



1

---

## 2 Functional Elements Specification

### 3 Committee Specifications 1.0, 27-May-2005

#### 4 Document identifier:

5 fws-fe-1.0-guidelines-spec-cs-01.doc

#### 6 Location:

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

#### 8 Editor:

9 Tan Puay Siew, Singapore Institute of Manufacturing Technology, SIMTech  
10 ([pstan@simtech.a-star.edu.sg](mailto:pstan@simtech.a-star.edu.sg))

11

#### 12 Contributor(s):

13 Cheng Huang Kheng, SIMTech ([jason@simtech.a-star.edu.sg](mailto:jason@simtech.a-star.edu.sg))

14 Lee Eng Wah, SIMTech ([ewlee@simtech.a-star.edu.sg](mailto:ewlee@simtech.a-star.edu.sg))

15 Christopher Haddad, Individual ([haddadc@cobia.net](mailto:haddadc@cobia.net))

16 Kenneth Lim, Crimson Logic Pte Ltd ([kennethlim@crimsonlogic.com](mailto:kennethlim@crimsonlogic.com))

17 Ravi Shankar, Crimson Logic Pte Ltd ([ravishankar@crimsonlogic.com](mailto:ravishankar@crimsonlogic.com))

18 Jagdip Talla, Crimson Logic Pte Ltd ([jagdip@crimsonlogic.com](mailto:jagdip@crimsonlogic.com))

19 Roberto Pascual, Infocomm Development Authority (IDA) of Singapore  
20 ([rbpascual@yahoo.com](mailto:rbpascual@yahoo.com))

21 Ang Chai Hong, SIMTech ([chang@simtech.a-star.edu.sg](mailto:chang@simtech.a-star.edu.sg))

22

23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48

**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](mailto:fwsifesc@lists.oasis-open.org) list. Others should subscribe to and send comments to the [fwsicomment@lists.oasis-open.org](mailto:fwsicomment@lists.oasis-open.org) list. To subscribe, send an email message to [fwsicomment-request@lists.oasis-open.org](mailto:fwsicomment-request@lists.oasis-open.org) with the word "subscribe" as the body of the message.

For information on whether any patents<sup>1</sup> have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the FWSI TC web page (<http://www.oasis-open.org/committees/fwsii/>).

---

<sup>1</sup> 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.

---

## Table of Contents

50	<b>1</b>	<b>Introduction</b> .....	7
51	1.1	Document Outline.....	7
52	1.2	Motivation.....	8
53	1.3	Terminology.....	8
54	<b>2</b>	<b>List of Functional Elements</b> .....	9
55	2.1	Event Handler Functional Element.....	9
56	2.1.1	Motivation.....	9
57	2.1.2	Terms Used.....	9
58	2.1.3	Key Features.....	11
59	2.1.4	Interdependencies.....	12
60	2.1.5	Related Technologies and Standards.....	12
61	2.1.6	Model.....	13
62	2.1.7	Usage Scenarios.....	14
63	2.2	Group Management Functional Element.....	36
64	2.2.1	Motivation.....	36
65	2.2.2	Terms Used.....	36
66	2.2.3	Key Features.....	37
67	2.2.4	Interdependency.....	38
68	2.2.5	Related Technologies and Standards.....	38
69	2.2.6	Model.....	38
70	2.2.7	Usage Scenarios.....	39
71	2.3	Identity Management Functional Element.....	44
72	2.3.1	Motivation.....	44
73	2.3.2	Terms Used.....	45
74	2.3.3	Key Features.....	46
75	2.3.4	Interdependencies.....	47
76	2.3.5	Related Technologies and Standards.....	47
77	2.3.6	Model.....	48
78	2.3.7	Usage Scenarios.....	49
79	2.4	Log Utility Functional Element.....	55
80	2.4.1	Motivation.....	55
81	2.4.2	Terms Used.....	55
82	2.4.3	Key Features.....	56
83	2.4.4	Interdependencies.....	56
84	2.4.5	Related Technologies and Standards.....	56
85	2.4.6	Model.....	57
86	2.4.7	Usage Scenarios.....	57
87	2.5	Notification Functional Element.....	65
88	2.5.1	Motivation.....	65
89	2.5.2	Terms Used.....	65
90	2.5.3	Key Features.....	66
91	2.5.4	Interdependencies.....	66
92	2.5.5	Related Technologies and Standards.....	67
93	2.5.6	Model.....	67

94	2.5.7 Usage Scenarios .....	68
95	2.6 Phase and Lifecycle Management Functional Element .....	74
96	2.6.1 Motivation .....	74
97	2.6.2 Terms Used .....	74
98	2.6.3 Key Features .....	75
99	2.6.4 Interdependencies .....	76
100	2.6.5 Related Technologies and Standards .....	76
101	2.6.6 Model .....	76
102	2.6.7 Usage Scenarios .....	77
103	2.7 Presentation Transformer Functional Element .....	83
104	2.7.1 Motivation .....	83
105	2.7.2 Terms Used .....	83
106	2.7.3 Key Features .....	83
107	2.7.4 Interdependencies .....	83
108	2.7.5 Related Technologies and Standards .....	83
109	2.7.6 Model .....	84
110	2.7.7 Usage Scenario .....	84
111	2.8 Role and Access Management Functional Element .....	86
112	2.8.1 Motivation .....	86
113	2.8.2 Terms Used .....	86
114	2.8.3 Key Features .....	87
115	2.8.4 Interdependencies .....	88
116	2.8.5 Related Technologies and Standards .....	88
117	2.8.6 Model .....	89
118	2.8.7 Usage Scenario .....	89
119	2.9 Search Functional Element .....	102
120	2.9.1 Motivation .....	102
121	2.9.2 Terms Used .....	102
122	2.9.3 Key Features .....	103
123	2.9.4 Interdependencies .....	104
124	2.9.5 Related Technologies and Standards .....	104
125	2.9.6 Model .....	104
126	2.9.7 Usage Scenario .....	104
127	2.10 Secure SOAP Management Functional Element .....	108
128	2.10.1 Motivation .....	108
129	2.10.2 Terms Used .....	108
130	2.10.3 Key Features .....	109
131	2.10.4 Interdependencies .....	109
132	2.10.5 Related Technologies and Standards .....	109
133	2.10.6 Model .....	110
134	2.10.7 Usage Scenarios .....	111
135	2.11 Sensory Functional Element .....	115
136	2.11.1 Motivation .....	115
137	2.11.2 Terms Used .....	115
138	2.11.3 Key Features .....	116
139	2.11.4 Interdependencies .....	116
140	2.11.5 Related Technologies and Standards .....	116
141	2.11.6 Model .....	116
142	2.11.7 Usage Scenarios .....	117
143	2.12 Service Management Functional Element .....	120

144	2.12.1 Motivation .....	120
145	2.12.2 Terms Used .....	120
146	2.12.3 Key Features .....	120
147	2.12.4 Interdependencies .....	121
148	2.12.5 Related Technologies and Standards .....	121
149	2.12.6 Model .....	122
150	2.12.7 Usage Scenarios .....	122
151	2.13 Service Registry Functional Element .....	129
152	2.13.1 Motivation .....	129
153	2.13.2 Terms Used .....	129
154	2.13.3 Key Features .....	130
155	2.13.4 Interdependencies .....	130
156	2.13.5 Related Technologies and Standards .....	130
157	2.13.6 Model .....	131
158	2.13.7 Usage Scenario .....	131
159	2.14 Service Tester Functional Element .....	143
160	2.14.1 Motivation .....	143
161	2.14.2 Terms Used .....	143
162	2.14.3 Key Features .....	143
163	2.14.4 Interdependencies .....	143
164	2.14.5 Related Technologies and Standards .....	143
165	2.14.6 Model .....	144
166	2.14.7 Usage Scenarios .....	144
167	2.15 User Management Functional Element .....	147
168	2.15.1 Motivation .....	147
169	2.15.2 Terms Used .....	147
170	2.15.3 Key Features .....	148
171	2.15.4 Interdependencies .....	149
172	2.15.5 Related Technologies and Standards .....	149
173	2.15.6 Model .....	149
174	2.15.7 Usage Scenarios .....	150
175	2.16 Web Service Aggregator Functional Element .....	159
176	2.16.1 Motivation .....	159
177	2.16.2 Terms Used .....	159
178	2.16.3 Key Features .....	160
179	2.16.4 Interdependencies .....	161
180	2.16.5 Related Technologies and Standards .....	161
181	2.16.6 Model .....	161
182	2.16.7 Usage Scenarios .....	162
183	3 Functional Elements Usage Scenario .....	168
184	3.1 Service Monitoring .....	169
185	3.2 Securing SOAP Messages .....	170
186	3.3 Decoupled User Access Management .....	171
187	4 References .....	173
188	Appendix A. Acknowledgments .....	175
189	Appendix B. Revision History .....	176

190 *Appendix C. Notices*..... 177  
191  
192

---

# 1 Introduction

193

194

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

203

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

215

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

224

## 1.1 Document Outline

225

226

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

231

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

235

- Motivation

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

239 • Terms Used

240 A glossary of the terms used. An explanation or illustration of the runtime capabilities of the  
241 Functional Element are also provided where appropriate.

242 • Key Features

243 A list of key features to be implemented are provided here and is expressed in the normative  
244 form.

245 • Interdependencies

246 In this section, the interdependencies between Functional Elements are provided to clarify  
247 the linkages between FEs (if any).

248 • Related Technologies and Standards

249 Here, the reliance of the Functional Elements on related technologies and specifications (or  
250 standards) are provided

251

252 Section 3 provides the examples of how the Functional Elements can be assembled to accelerate  
253 web service-enabled applications. From these Functional Elements, a variety of solutions can be  
254 built.

255

## 256 1.2 Motivation

257

258 In a Service-Oriented Architecture (SOA) environment, new applications/services are created  
259 through the assembly of existing services. One of the key advantages of this loosely coupled  
260 model is that it allows the new application/service to leverage on 3<sup>rd</sup> party services. As a typical  
261 3<sup>rd</sup> party's implementation of the services is done via the software component approach, this  
262 specification further proliferate new applications/services by defining a framework for Web  
263 Services implementation consisting of Functional Elements. Through these Functional Elements,  
264 which are implementation neutral, this Specification hopes to influence future software  
265 development towards assembly of services rather than 'pure built only'.

## 266 1.3 Terminology

267

268 Within this document the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL  
269 NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this  
270 document are to be interpreted as described in RFC2119 [3].

271

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

275

276

---

## 277 **2 List of Functional Elements**

### 278 **2.1 Event Handler Functional Element**

#### 279 **2.1.1 Motivation**

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

- 284 • Managing the information flow through a subscription based mechanism,
- 285 • Streamlining information into meaningful categories so as to improve relevancy to a  
286 potential consumer of the information, and
- 287 • Refining information flow via a filtering mechanism

288

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

- 291 • Primary Requirements
  - 292 • MANAGEMENT-111,
  - 293 • PROCESS-005, and
  - 294 • PROCESS-100 to PROCESS-117.
- 295 • Secondary Requirements
  - 296 • None

#### 297 **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.

298

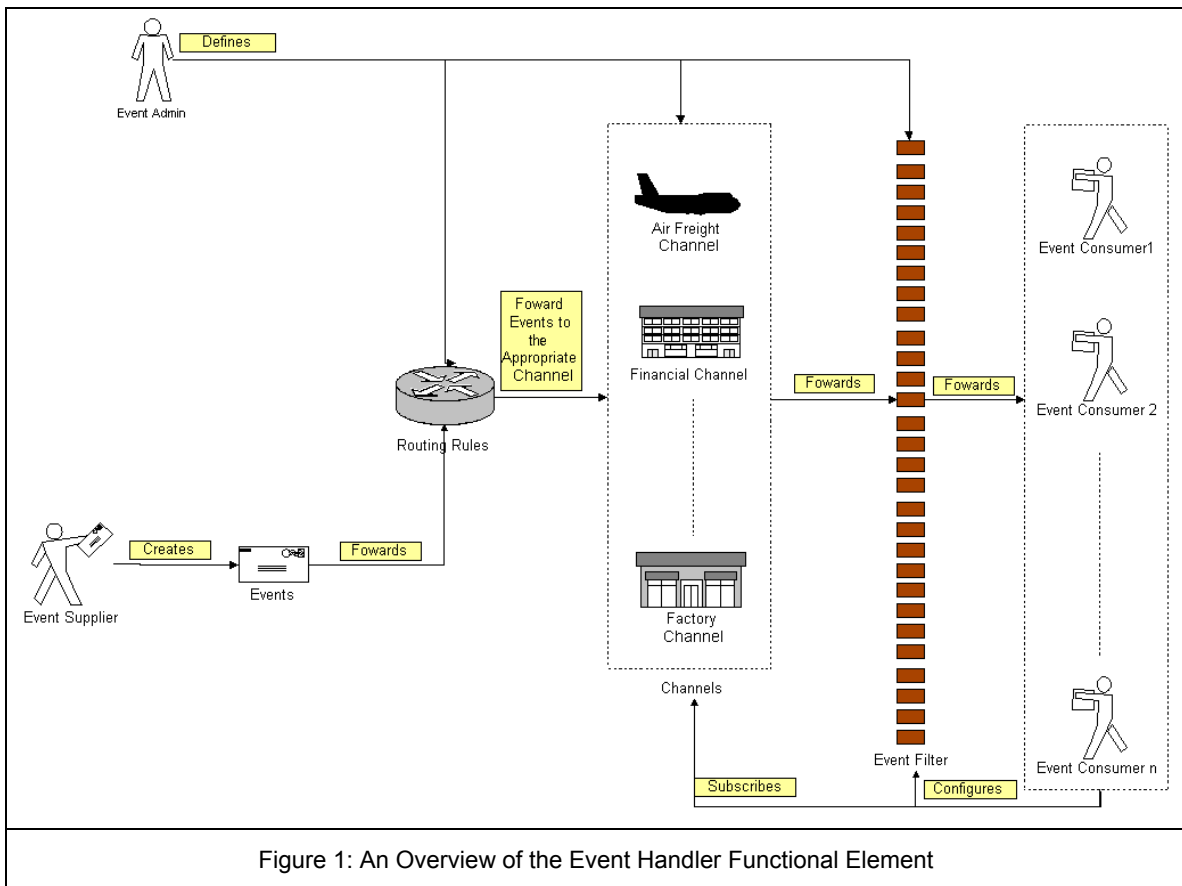


Figure 1: An Overview of the Event Handler Functional Element

299

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

### 307 2.1.3 Key Features

308 Implementations of the Event Handler Functional Element are expected to provide the following  
309 key features:

310 1. The Functional Element MUST provide the capability to manage the creation (or registration)  
311 and deletion of instances of the following concepts based on a pre-defined structure:

312 1.1. Event Supplier,

313 1.2. Event Consumer,

314 1.3. Event,

315 1.4. Filter,

316 1.5. Channel, and

317 1.6. Routing Rule.

318 2. The Functional Element MUST provide the capability to manage all the information (attribute  
319 values) stored in such concepts. This includes the capability to retrieve and update  
320 attribute's values belonging to the concepts mentioned in Key Feature (1).

321 3. The Functional Element MUST provide the capability to enable Event Suppliers to trigger  
322 relevant Events.

323 4. The Functional Element MUST provide a mechanism to associate/unassociate Routing  
324 Rules to an Event.

325 *Example: As shown in Figure 1, where an event can be routed to Air Freight or Financial*  
326 *Channel or even to all channels based on the Routing Rules that are associated*  
327 *with the Event.*

328 5. As part of Key Feature (3), the Routing Rules must be able to route an event to all, specified  
329 Channels or individual Event Consumers.

330 6. The Functional Element MUST enable Event Consumers to execute the following tasks to  
331 improve the relevancy of the incoming events”

332 6.1. Subscribe/Unsubscribe to relevant Channel(s), and

333 6.2. Apply a filter to the appropriate channel or event, which helps to refine the criteria of a  
334 useful event further.

335 7. The Functional Element MUST provide the capability to notify relevant Event Consumers  
336 when an event occurs.

337 Examples of notification types include SMS, email and Web Services invocations.

338 8. As part of Key Feature (6), the notification must be able to handle differing requirements  
339 arising from different notification formats.

340 *Example: If the incoming event contains 2 important attributes, the order or position of*  
341 *these 2 attributes must be configurable to suit the convenience of the Event*  
342 *Consumer. This is extremely important in the case of Web Service Invocations.*

343 10. The Functional Element MUST provide a mechanism for managing the concepts specified  
344 across different application domains.

345 *Example: Namespace control mechanism*

346

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

- 349 1. The Functional Element MAY provide a mechanism to enable active event detection.
- 350 2. If Key Feature (1) is implemented, then the Functional Element MUST provide the following
- 351 capabilities also:
- 352 2.1. Non-intrusive detection
- 353 *Example: The detection of a new event through periodic inspection of the audit log.*
- 354 2.2. Configurable event detection schedule
- 355 *Example: To inspect the audit log every 2 hours, where the duration between*
- 356 *inspections is configurable.*
- 357 2.3. Ability to retrieve relevant data from external source(s) for further event processing by
- 358 Event Handler
- 359 *Example: To retrieve Error Type and Message from audit log.*
- 360 3. The Functional Element MAY provide the capability to record event processing within the
- 361 Event Handler. The logging of event processing includes the occurrences of event, sending
- 362 of notifications, warning and error messages generated in the processing of events.
- 363 4. The Functional Element MAY provide the capability scheduled-based event notification.
- 364

#### 365 2.1.4 Interdependencies

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

366

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

367

#### 368 2.1.5 Related Technologies and Standards

369 None

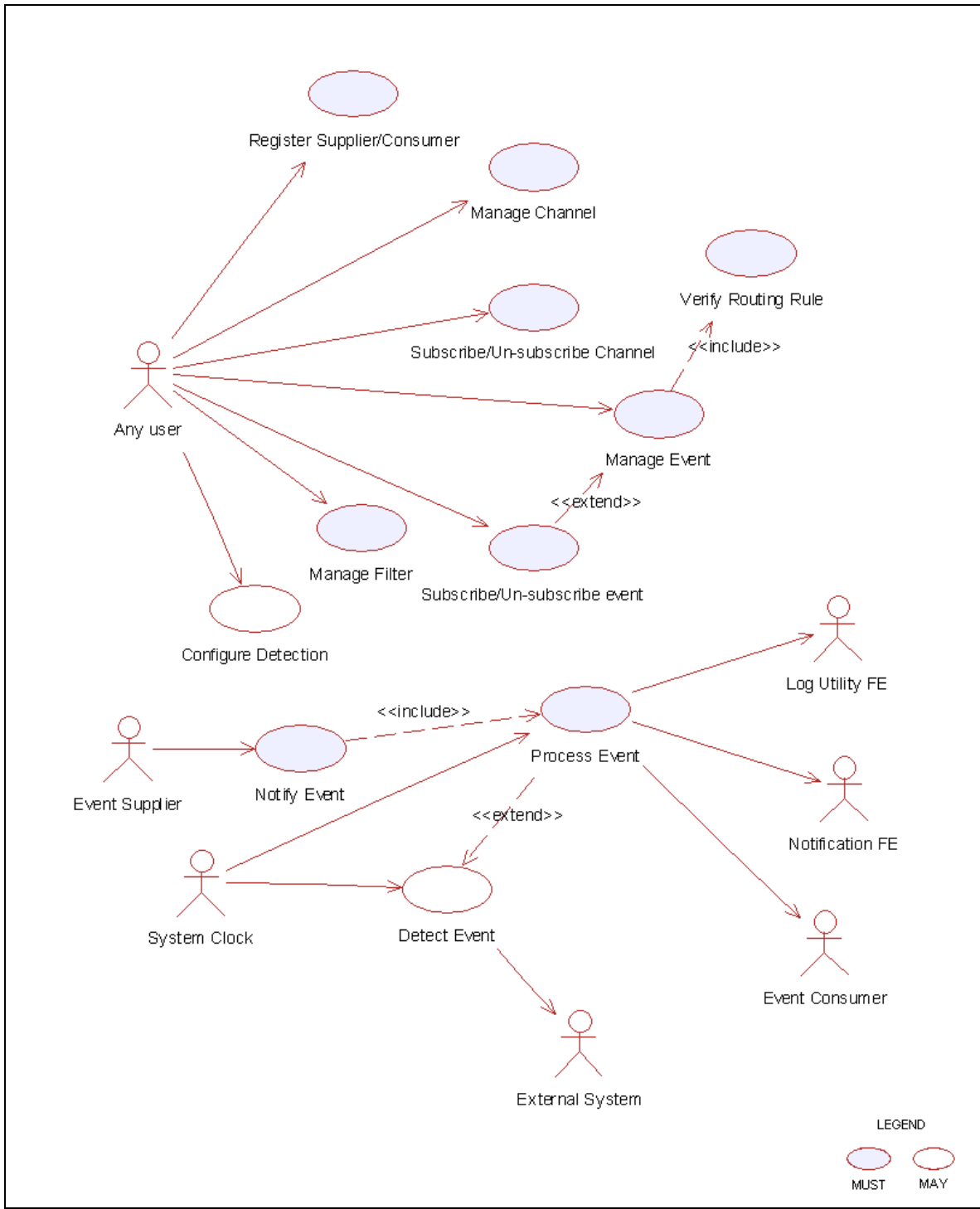


Figure 2: Model Of the Event Handler Functional Element [4]

371 **2.1.7 Usage Scenarios**

372 **2.1.7.1 Register Supplier/Consumer**

373 **2.1.7.1.1 Description**

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

376 **2.1.7.1.2 Flow of Events**

377 **2.1.7.1.2.1 Basic Flow**

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

380 1: The user sends a request to Event Handler together with its profile data and operation.

381 2: Based on the operation it specified, one of the following sub-flows is executed:

- 382
- If the operation is '**Register as supplier**', then sub-flow 2.1 is executed.
  - 383 • If the operation is '**Register as consumer**', then sub-flow 2.2 is executed.
  - 384 • If the operation is '**Un-register as supplier**', then sub-flow 2.3 is executed.
  - 385 • If the operation is '**Un-register as consumer**', then sub-flow 2.4 is executed.
  - 386 • If the operation is '**Update supplier**', then sub-flow 2.5 is executed.
  - 387 • If the operation is '**Update consumer**', then sub-flow 2.6 is executed.
  - 388 • If the operation is '**Retrieve supplier**', then sub-flow 2.7 is executed.
  - 389 • If the operation is '**Retrieve consumer**', then sub-flow 2.8 is executed.

390 2.1: Register as Supplier.

391 2.1.1: The Functional Element gets the user profile data, i.e. namespace, name,  
392 description and type.

393 2.1.2: The Functional Element registers the user as event supplier.

394 2.1.3: The Functional Element returns the Supplier Id to the user.

395 2.2: Register as Consumer.

396 2.2.1: The Functional Element gets the user profile data, i.e. namespace, name,  
397 description and type.

398 2.2.2: The Functional Element registers the user as event consumer.

399 2.2.3: The Functional Element returns the Consumer Id to the user.

400 2.3: Un-register as Supplier.

401 2.3.1: The Functional Element gets the user namespace and name or User Id.

402 2.3.2: The Functional Element checks whether the user is a supplier.

403 2.3.3: The Functional Element removes the user as supplier.

404 2.4: Un-register as Consumer.

405 2.4.1: The Functional Element gets the user namespace and name or User Id.

406 2.4.2: The Functional Element checks whether the user is a consumer.

407 2.4.3: The Functional Element removes the user as consumer.

408 2.5: Update Supplier.

409 2.5.1: The Functional Element gets the user namespace and name or User Id together  
410 with the user profile.

411 2.5.2: The Functional Element checks whether the user is a supplier.

412 2.5.2: The Functional Element updates the user profile.

413 2.6: Update Consumer.

414 2.6.1: The Functional Element gets the user namespace and name or User Id together  
415 with the user profile.

416 2.6.2: The Functional Element checks whether the user is a consumer.

417 2.6.3: The Functional Element updates the user profile.

418 2.7: Retrieve Supplier.

419 2.7.1: The Functional Element gets the user namespace and name or User Id.

420           2.7.2: The Functional Element checks whether the user is a supplier.

421           2.7.3: The Functional Element returns the user profile.

422           2.8: Retrieve Consumer.

423           2.8.1: The Functional Element gets the user namespace and name or User Id.

424           2.8.2: The Functional Element checks whether the user is a consumer.

425           2.8.3: The Functional Element returns the user profile.

426           3: The Functional Element returns the results to indicate the success or failure of this operation to  
427           the user and the use case ends.

#### 428           **2.1.7.1.2.2 Alternative Flows**

429           1: Supplier Already Registered.

430           1.1: If in the basic flow 2.1.2, the user already registered as supplier, Functional Element will  
431           return an error message to the user and the use case ends.

432           2: Consumer Already Registered.

433           2.1: If in the basic flow 2.2.2, the user already registered as consumer, Functional Element  
434           will return an error message to the user and the use case ends.

435           3: Supplier or Consumer Not Registered.

436           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  
437           registered, Functional Element will return an error message to the user and the use case  
438           ends.

439           4: Persistency Mechanism Error.

440           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  
441           perform data persistency, Functional Element will return an error message to the user and the  
442           use case ends.

443

#### 444           **2.1.7.1.3 Special Requirements**

445           None.

446 **2.1.7.1.4 Pre-Conditions**

447 None.

448 **2.1.7.1.5 Post-Conditions**

449 None.

450 **2.1.7.2 Manage Channel**

451 **2.1.7.2.1 Description**

452 This use case allows the user to manage channels.

453 **2.1.7.2.2 Flow of Events**

454 **2.1.7.2.2.1 Basic Flow**

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

456 1: The user sends request to manipulate a channel.

457 2: Based on the operation it specifies, one of the following sub-flows is executed:

- 458 • If the operation is '**Create Channel**', the sub-flow 2.1 is executed.
- 459 • If the operation is '**Retrieve Channel**', the sub-flow 2.2 is executed.
- 460 • If the operation is '**Update Channel**', the sub-flow 2.3 is executed.
- 461 • If the operation is '**Delete Channel**', the sub-flow 2.4 is executed.

462 2.1: Create Channel.

463 2.1.1: The Functional Element gets channel definition, i.e. namespace, channel name  
464 and description.

465 2.1.2: The Functional Element checks whether the channel exists.

466 2.1.3: The Functional Element creates the channel.

467 2.2: Retrieve Channel.

468 2.2.1: The Functional Element gets namespace, channel name and retrieve condition.

469 2.2.2: The Functional Element retrieves the channel's information according to the  
470 condition.

471 2.3: Update Channel.

472 2.3.1: The Functional Element gets namespace, channel name and description.

473 2.3.2: The Functional Element checks whether the channel exists.

474 2.3.3: The Functional Element updates the channel definition.

475 2.4: Delete Channel.

476 2.4.1: The Functional Element gets namespace and channel name.

477 2.4.2: The Functional Element checks whether the channel exists.

478 2.4.3: The Functional Element removes the channel from the Functional Element.

479 3: The Functional Element returns the results of the operation to the user and the use case ends.

#### 480 **2.1.7.2.2.2 Alternative Flows**

481 1: Channel Already Exists.

482 1.1: If in the basic flow 2.1.2, the channel is already defined, Functional Element returns an  
483 error message and the use case ends.

484 2: Conditional Retrieving.

485 2.1: In the basic flow 2.2.2:

486 2.1 1: If the condition is the retrieval by channel name and the channel does not exist,  
487 then it will go to Alternative Flow 3.

488 2.1.2: If the condition is the retrieval of one channel definition, it returns the definition of  
489 that channel and the use case ends.

490 2.1.3: If the condition is the retrieval of all channels' information, it returns all channels  
491 definition and the use case ends.

492 2.1.4: If the condition is the retrieval of channel through channel description, it will return  
493 all matched channels and the use case ends.

494 2.1.5: If the condition is the retrieval of registered consumers, it returns the list of  
495 consumer registered on the channel and the use case ends.

496 3: Channel Not Found.

497 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the channel does not exist, Functional  
498 Element will return an error message and the use case ends.

499 4: Consumer Not Found.

500 4.1: If in the basic flow 2.1.3, 2.5.3 and 2.6.3, the event consumer does not exist,  
501 Functional Element will return an error message and the use case ends.

502 5: Extension Point.

503 5.1: If in the basic flow 2.1.3, and 2.3.3, the event consumers that subscribed to the  
504 channel are provided, the use case Subscribe/un-subscribe channel will be extended.

### 505 **2.1.7.2.3 Special Requirements**

506 None.

### 507 **2.1.7.2.4 Pre-Conditions**

508 None.

### 509 **2.1.7.2.5 Post-Conditions**

510 None.

## 511 **2.1.7.3 Subscribe/Un-subscribe To Channel**

### 512 **2.1.7.3.1 Description**

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

### 514 **2.1.7.3.2 Flow of Events**

#### 515 **2.1.7.3.2.1 Basic Flow**

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

517 1: The user sends the request.

518 2: Based on the operation it specifies, one of the following sub-flows is executed:

519 • If the operation is '**Subscribe to Channel**', then sub-flow 2.1 is executed.

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

521 2.1: Subscribe To Channel.

522 2.1.1: The Functional Element gets event consumer Id, or consumer namespace and  
523 consumer name, together with channel namespace and channel name.

524 2.1.2: The Functional Element checks whether the channel exists.

525 2.1.3: The Functional Element adds the subscription of the consumer to the channel.

526 2.2: Un-Subscribe To Channel.

527 2.2.1: The Functional Element gets event consumer Id, or consumer namespace and  
528 consumer name, together with channel namespace and channel name.

529 2.2.2: The Functional Element checks whether the channel exists.

530 2.2.3: The Functional Element removes the subscription of the consumer to the channel.

531 3: The Functional Element returns the results of the operation to the user and the use case ends.

532 **2.1.7.3.2 Alternative Flows**

533 1: Channel Not Found.

534 1.1: If in the basic flow 2.1.2 and 2.2.2, the channel specified does not exist, Functional  
535 Element will return an error message to the user and the use case ends.

536 2: Event Consumer Not Found.

537 2.1: If in the basic flow 2.1.2 and 2.2.2, the event consumer related does not exist, Functional  
538 Element will return an error message to the user and the use case ends.

539 **2.1.7.3.3 Special Requirements**

540 None.

541 **2.1.7.3.4 Pre-Conditions**

542 None.

543 **2.1.7.3.5 Post-Conditions**

544 None.

## 545 **2.1.7.4 Manage Event**

### 546 **2.1.7.4.1 Description**

547 This use case describes the scenarios of managing events.

### 548 **2.1.7.4.2 Flow of Events**

#### 549 **2.1.7.4.2.1 Basic Flow**

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

551 1: The user sends a request to the Functional Element.

552 2: Based on the operation it specifies, one of the following sub-flows is executed:

- 553 • If the operation is '**Create Event**', then sub-flow 2.1 is executed.
- 554 • If the operation is '**Retrieve Event Information**', then sub-flow 2.2 is executed.
- 555 • If the operation is '**Update Event Definition**', then sub-flow 2.3 is executed.
- 556 • If the operation is '**Delete Event**', then sub-flow 2.4 is executed.
- 557 • If the operation is '**Assign Flow**', then sub-flow 2.5 is executed.
- 558 • If the operation is '**Un-Assign Flow**', then sub-flow 2.6 is executed.

559 2.1: Create Event

560 2.1.1: The Functional Element gets event definition including namespace, event name,  
561 event description, event routing rule, and event attributes definition.

562 2.1.2: The Functional Element verifies the parameters.

563 2.1.3: The Functional Element verifies the routing rule through use case verify routing  
564 rule.

565 2.1.4: The Functional Element creates event definition by recording the definition of  
566 event.

567 2.2: Retrieve Event.

568 2.2.1: The Functional Element gets namespace, event name, and condition.

569 2.2.2: The Functional Element retrieves the event definition according to the condition.

570 2.3: Update Event Definition

571 2.3.1: The Functional Element gets event definition including namespace, event name,  
572 event description, event routing rule, and event attributes definition.

573 2.3.2: The Functional Element verifies the parameters.

574 2.3.3: The Functional Element verifies the routing rule through use case verify routing  
575 rule.

576 2.3.4: The Functional Element updates the event definition.

577 2.4: Delete Event.

578 2.4.1: The Functional Element gets namespace and event name.

579 2.4.2: The Functional Element checks whether the event exists.

580 2.4.3: The Functional Element deletes the event definition.

581 2.5: Assign Flow.

582 2.5.1: The Functional Element gets namespace, event name and flow name.

583 2.5.2: The Functional Element checks whether the event exists and flow defined.

584 2.5.3: The Functional Element assigns the flow to the event.

585 2.6: Un-assign Flow.

586 2.6.1: The Functional Element gets namespace, event name and flow name.

587 2.6.2: The Functional Element checks whether the event exists and flow defined.

588 2.6.3: The Functional Element un-assigns the flow to the event.

589 3: The Functional Element returns the results of the operation to the user and the use case ends.

590 **2.1.7.4.2.2 Alternative Flows**

591 1: Event Already Exist.

592 1.1: If in the basic flow 2.1.2, the event already exists, Functional Element will return an error  
593 message to the user and the use case ends.

- 594 2: Parameters Are Invalid.
- 595 2.1: If in the basic flow 2.1.2 and 2.3.2, the parameters provided are invalid, Functional  
596 Element will return an error message to the user and the use case ends.
- 597 3: Event Not Found.
- 598 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the event does not exist, Functional Element  
599 will return an error message to the user and the use case ends.
- 600 4: Flow Not Defined.
- 601 4.1: If in the basic flow 2.1.2 and 2.3.2, the flow does not exist, Functional Element will return  
602 an error message to the user and the use case ends.
- 603 5: Condition Retrieve.
- 604 5.1: In the basic flow 2.2.2:
- 605 5.1.1: If the retrieving condition is the retrieval of event definition based on event name, it  
606 returns event definition and the use case ends.
- 607 5.1.2: If the retrieving condition is the retrieval of all event definition, it returns all event  
608 definition and the use case ends.
- 609 5.1.3: If the retrieving condition is the retrieval of events assigned to specified channel, it  
610 returns the list of event definitions.
- 611 5.1.4: If the retrieving condition is the retrieval of channels associated with specified  
612 event, it returns the list of channel definition.
- 613 6: Extension Point.
- 614 6.1: If in the basic flow 2.1.4, and 2.3.4, the event consumers that subscribed to the event are  
615 provided, the use case Subscribe/Un-subscribe event will be extended.
- 616 **2.1.7.4.3 Special Requirements**
- 617 None.
- 618 **2.1.7.4.4 Pre-Conditions**
- 619 None.

620 **2.1.7.4.5 Post-Conditions**

621 None.

622 **2.1.7.5 Subscribe/Un -subscribe To Event**

623 **2.1.7.5.1 Description**

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

625 **2.1.7.5.2 Flow of Events**

626 **2.1.7.5.2.1 Basic Flow**

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

628 1: The user sends a request.

629 2: Based on the operation it specifies, one of the following sub-flows is executed:

630 • If the operation is '**Subscribe to Event**', then sub-flow 2.1 is executed.

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

632 2.1: Subscribe To Event.

633 2.1.1: The Functional Element gets event consumer Id, or consumer namespace and  
634 consumer name, together with event namespace and event name.

635 2.1.2: The Functional Element checks whether the event exists.

636 2.1.3: The Functional Element adds the subscription of the consumer to the event.

637 2.2: Un-Subscribe To Event.

638 2.2.1: The Functional Element gets event consumer Id, or consumer namespace and  
639 consumer name, together with event namespace and event name.

640 2.2.2: The Functional Element checks whether the event exists.

641 2.2.3: The Functional Element removes the subscription of the consumer to the event.

642 3: The Functional Element returns the results of the operation to the user and the use case ends.

643 **2.1.7.5.2 Alternative Flows**

644 1: Event Not Found.

645 1.1: If in the basic flow 2.1.2 and 2.2.2, the event specified does not exist, Functional Element  
646 will return an error message to the user and the use case ends.

647 2: Event Consumer Not Found.

648 2.1: If in the basic flow 2.1.2 and 2.2.2, the event consumer related does not exist, Functional  
649 Element will return an error message to the user and the use case ends.

650 **2.1.7.5.3 Special Requirements**

651 None.

652 **2.1.7.5.4 Pre-Conditions**

653 None.

654 **2.1.7.5.5 Post-Conditions**

655 None.

656 **2.1.7.6 Verify Routing Rule**

657 **2.1.7.6.1 Description**

658 This use case verifies the syntax of routing rule.

659 **2.1.7.6.2 Flow of Events**

660 **2.1.7.6.2.1 Basic Flow**

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

662 1: The user sends a request.

663 2: The Functional Element gets the routing expression.

664 3: The Functional Element checks the syntax of routing expression.

665 4: The Functional Element verifies the parameters.

666 5: The Functional Element returns the status of the operation to the user and the use case ends.

667 **2.1.7.6.2 Alternative Flows**

668 1: Routing Rule Expression Syntax Error.

669 1.1: If in the basic flow 3, there is a syntax error, Functional Element will return an error  
670 message to the user and the use case ends.

671 2: Event Consumer Not Found.

672 2.1: If in the basic flow 4, the event consumer related does not exist, Functional Element will  
673 return an error message to the user and the use case ends.

674 **2.1.7.6.3 Special Requirements**

675 None.

676 **2.1.7.6.4 Pre-Conditions**

677 None.

678 **2.1.7.6.5 Post-Conditions**

679 None.

680 **2.1.7.7 Manage Filter**

681 **2.1.7.7.1 Description**

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

684 **2.1.7.7.2 Flow of Events**

685 **2.1.7.7.2.1 Basic Flow**

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

687 1: The user sends a request to manage a filter.

688 2: Based on the operation it specifies, one of the following sub-flows is executed:

- 689
- If the operation is '**Create Filter**', then sub-flow 2.1 is executed.

690

  - If the operation is '**Retrieve Filter**', then sub-flow 2.2 s executed.

691

  - If the operation is '**Update Filter**', then sub-flow 2.3 is executed.

- 692       • If the operation is '**Delete Filter**', then sub-flow 2.4 is executed.
- 693       2.1: Create Filter.
- 694           2.1.1: The Functional Element gets filter definition, i.e. consumer namespace, consumer  
695           name, filter name, description, event name or channel name.
- 696           2.1.2: The Functional Element checks whether the event or channel exists.
- 697           2.1.3: The Functional Element saves the filter definition.
- 698       2.2: Retrieve Filter.
- 699           2.2.1: The Functional Element gets the filter name.
- 700           2.2.2: The Functional Element retrieves the filter information according to the name.
- 701       2.3: Update Filter.
- 702           2.3.1: The Functional Element gets filter definition, i.e. consumer namespace, name, filter  
703           name, description, event name or channel name.
- 704           2.3.2: The Functional Element checks the parameters.
- 705           2.3.3: The Functional Element updates the filter definition.
- 706       2.4: Delete Filter.
- 707           2.4.1: The Functional Element gets namespace and filter name.
- 708           2.4.2: The Functional Element checks whether the filter exists.
- 709           2.4.3: The Functional Element removes the filter from the Functional Element.
- 710       3: The Functional Element returns the results of the operation to the user and the use case ends.

#### 711   **2.1.7.7.2.2 Alternative Flows**

##### 712   1: Filter Already Exists.

713       1.1: If in the basic flow 2.1.2, the filter is already defined, Functional Element will return an  
714       error message and the use case ends.

##### 715   2: Event Not Found.

716 2.1: If in the basic flow 2.1.2 and 2.3.2, the event used does not exist, Functional Element will  
717 return an error message and the use case ends.

718 3: Channel Not Found.

719 3.1: If in the basic flow 2.1.2 and 2.3.2, the channel used does not exist, Functional Element  
720 will return an error message and the use case ends.

721 4: Consumer Not Found.

722 4.1: If in the basic flow 2.1.3, 2.5.3, and 2.6.3, the event consumer does not exist, Functional  
723 Element will return an error message and the use case ends.

### 724 **2.1.7.7.3 Special Requirements**

725 None.

### 726 **2.1.7.7.4 Pre-Conditions**

727 None.

### 728 **2.1.7.7.5 Post-Conditions**

729 None.

## 730 **2.1.7.8 Notify Event**

### 731 **2.1.7.8.1 Description**

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

### 735 **2.1.7.8.2 Flow of Events**

#### 736 **2.1.7.8.2.1 Basic Flow**

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

738 1: The user sends a notification.

739 2: The Functional Element receives the notification with parameters, i.e. event supplier id or event  
740 supplier namespace and name.

741 3: The Functional Element checks whether the event is defined and event supplier is registered.

742 4: Include use case Process Event to process the notification of event.

743 5: The Functional Element returns the status of the operation to the user and the use case ends.

#### 744 **2.1.7.8.2.2 Alternative Flows**

745 1: User Is Not Registered.

746 1.1: If in the basic flow 3, the user is not registered, Functional Element will return an error  
747 message to the user and the use case ends.

748 2: Event Not Defined.

749 2.1: If in the basic flow 3, the event is not defined, Functional Element will return an error  
750 message to the user and the use case ends.

751 3: Error Returned.

752 3.1: If in the basic flow 4, an error is returned by use case Process event, Functional Element  
753 will return an error message to the user and the use case ends.

#### 754 **2.1.7.8.3 Special Requirements**

755 None.

#### 756 **2.1.7.8.4 Pre-Conditions**

757 None.

#### 758 **2.1.7.8.5 Post-Conditions**

759 None.

#### 760 **2.1.7.9 Configure Monitoring**

##### 761 **2.1.7.9.1 Description**

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

##### 764 **2.1.7.9.2 Flow of Events**

###### 765 **2.1.7.9.2.1 Basic Flow**

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

767 1: The user sends a request to manage a filter.

768 2: Based on the operation it specifies, one of the following sub-flows is executed:

769     • If the operation is 'Add Configuration', then sub-flow 2.1 is executed.

770     • If the operation is 'Remove Configuration', then sub-flow 2.2 is executed.

771 2.1: Add Configuration.

772     2.1.1: The Functional Element gets configuration definition, i.e. configuration name,  
773     namespace, event name, connection parameters, condition that signifies the events and  
774     schedule.

775     2.1.2: The Functional Element saves filter definition.

776 2.2: Remove Configuration.

777     2.2.1: The Functional Element gets configuration name.

778     2.2.2: The Functional Element removes the configuration.

779 3: The Functional Element returns the results of the operation to the user and the use case ends.

780 **2.1.7.9.2 Alternative Flows**

781 1: Configuration Exist.

782     1.1: If in the basic flow 2.1.2, the configuration already exists, Functional Element will return  
783     an error message and the use case ends.

784 **2.1.7.9.3 Special Requirements**

785 None.

786 **2.1.7.9.4 Pre-Conditions**

787 None.

788 **2.1.7.9.5 Post-Conditions**

789 None.

790 **2.1.7.10 Detect Event**

791 **2.1.7.10.1 Description**

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

794 **2.1.7.10.2 Flow of Events**

795 **2.1.7.10.2.1 Basic Flow**

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

797 1: The Functional Element clock generates a trigger.

798 2: The Functional Element receives the trigger and checks the condition for pre-defined  
799 monitoring sources.

800 3: The Functional Element checks whether the event happens.

801 4: The Functional Element returns the results of the operation and the use case ends.

802 **2.1.7.10.2.2 Alternative Flows**

803 1: External Functional Element Not Available.

804 1.1: If in the basic flow 3, the external Functional Element is not available and the Event  
805 Handler cannot make a connection, Functional Element will return an error message and the  
806 use case ends.

807 2: Data Not Available.

808 2.1: If in the basic flow 3, the data that signifies the event cannot be accessed, Functional  
809 Element will return an error message and the use case ends.

810 3: Extension Point.

811 3.1: If in the basic flow 3, the event happens, Functional Element will extend to use case  
812 Process event.

813 **2.1.7.10.3 Special Requirements**

814 None.

815 **2.1.7.10.4 Pre-Conditions**

816 None.

817 **2.1.7.10.5 Post-Conditions**

818 None.

## 819 2.1.7.11 Process Event

### 820 2.1.7.11.1 Description

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

### 824 2.1.7.11.2 Flow of Events

#### 825 2.1.7.11.2.1 Basic Flow

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

827 1: The user sends a request to process an event.

828 2: Based on the actor of this use case, one of the sub-flows is executed.

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

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

833 2.1: Initiated By Functional Element Clock.

834 2.1.1: The Functional Element looks up scheduled events defined to find out time-due  
835 notification.

836 2.1.2: The Functional Element retrieves the routing rule for the event.

837 2.1.3: The Functional Element looks up the corresponding consumers based on the  
838 routing rule.

839 2.1.4: The Functional Element retrieves filters defined and find out the event receivers.

840 2.1.5: The Functional Element notifies or invokes the event consumers based on the  
841 routing rule defined.

842 2.2: Initiated By Any User.

843 2.2.1: The Functional Element retrieves the routing rule for the event.

844 2.2.2: The Functional Element looks up the corresponding consumers.

845 2.2.3: The Functional Element retrieves filters defined and find out the event receivers.

846 2.2.4: The Functional Element notifies or invokes the event consumers based on the  
847 routing rule defined.

848 3: The Functional Element logs the notification of event and the use case ends.

#### 849 **2.1.7.11.2.2 Alternative Flows**

850 1: Notify Event.

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

- 852 • If the consumer type is '**SMTP**', then sub-flow Notify via SMTP is executed.
- 853 • If the consumer type is '**SMS Gateway**', then sub-flow Notify via SMS Gateway is  
854 executed.
- 855 • If the consumer type is '**Notify RPC-Web Service**', then sub-flow Notify RPC-Web  
856 Service is executed.
- 857 • If the consumer type is '**Notify Document Style Web Service**' then sub-flow Notify  
858 Document style Web Service is executed.

859 1.1: Notify via SMTP.

860 1.1.1: The Functional Element gets the pre-defined message for event and forms the  
861 parameters.

862 1.1.2: The Functional Element gets the parameters for SMTP server.

863 1.1.3: The Functional Element sends out the pre-defined message and the use case  
864 ends.

865 1.2: Notify via SMS Gateway.

866 1.2.1: The Functional Element gets the pre-defined message for event and forms the  
867 parameters.

868 1.2.2: The Functional Element gets the parameters for the SMS gateway.

869 1.2.3: The Functional Element sends out the pre-defined message and the use case  
870 ends.

871 1.3: Notify RPC-Web Service.

872 1.3.1: The Functional Element gets the operation parameter.

873 1.3.2: The Functional Element gets Web Services endpoint parameters.

874 1.3.3: The Functional Element dynamically invokes the Web Service and the use case  
875 ends.

876 1.4: Notify Document Style Web Service.

877 1.4.1: The Functional Element gets the operation parameter.

878 1.4.2: The Functional Element gets Web Services endpoint parameters.

879 1.4.3: The Functional Element dynamically generates the SOAP message and sends to  
880 the Web Services and the use case ends.

881 2: Flow Is Defined.

882 If in the basic flow 2.1.2 and 2.2.1, a flow is defined for the event, Functional Element will perform  
883 the following steps:

884 2.1: The Functional Element retrieves all the intended event consumers defined in the flow.

885 2.2: The Functional Element will go to basic flow 2.2.

886 2.3: The Functional Element will resume the execution from basic flow 2.1.2 or 2.2.1.

887 3: Log Utility Not Available.

888 3.1: If in the basic flow 3, the Log Utility Functional Element is not available, Functional  
889 Element will return an error message to the user and the use case ends.

890 4: SMS Gateway Not Available.

891 4.1: If in the Alternative Flow 1.2.3, the SMS Gateway is not available, Functional Element will  
892 return an error message to the user and the use case ends.

893 5: SMTP Server Not Available.

894 5.1: If in the Alternative Flow 1.1.3, the SMTP server is not available, Functional Element will  
895 return an error message to the user and the use case ends.

896 6: RPC Web Service Not Available.

897 6.1: If in the Alternative Flow 1.3.3, the Web Service is not available, Functional Element will  
898 return an error message to the user and the use case ends.

899 7: Document Style Web Service Not Available.

900 7.1: If in the Alternative Flow 1.4.3, document style Web Service is not available, Functional  
901 Element will return an error message to the user and the use case ends.

### 902 **2.1.7.11.3 Special Requirements**

#### 903 **2.1.7.11.3.1 Supportability**

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

#### 905 **2.1.7.11.4 Pre-Conditions**

906 None.

#### 907 **2.1.7.11.5 Post-Conditions**

908 None.

909

910

911

912

913

## 914 2.2 Group Management Functional Element

### 915 2.2.1 Motivation

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

921

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

- 924 • Primary Requirements
  - 925 • MANAGEMENT-050 to MANAGEMENT-053, and
  - 926 • MANAGEMENT-078
- 927 • Secondary Requirements
- 928 • None

### 929 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	<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"> <li>• Defining a set of basic user information that should be stored in any enterprise application.</li> <li>• Providing a means to extend this basic set of user information when needed.</li> <li>• Simplifying management by grouping related users together through certain criteria.</li> <li>• Having the flexibility of adopting both coarse and fine grain access controls.</li> </ul>
------------------------------	--

930

931 **2.2.3 Key Features**

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

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

950 *Example: Namespace control mechanism*

951

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

- 954 1. The Functional Element MAY provide a mechanism to enable different Groups to be related  
955 to one another.
- 956 2. The Functional Element MAY also provide a mechanism to enable hierarchical relationships  
957 between Groups.

958 *Example: Parent and Child Relationship.*

959 3. As an extension of Key Feature (2), the Functional Element MAY also provide the capability  
 960 to enable Groups to be part of the collection of “users” of another Group.

961 *Example: Adding of Group “Dept-A” to “Company-XYZ” – “Dept-A” is a Group, and also part*  
 962 *of the collection of Group “Company-XYZ”.*

963 4. The Functional Element MAY provide validity checks when managing information stored in a  
 964 Group.

965 *Example: Adding of User “john” – A validity check could be imposed to ensure that a user*  
 966 *“john” exists before adding to into the Group.*

967

968 **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.

969

970 **2.2.5 Related Technologies and Standards**

971 None.

972

973 **2.2.6 Model**

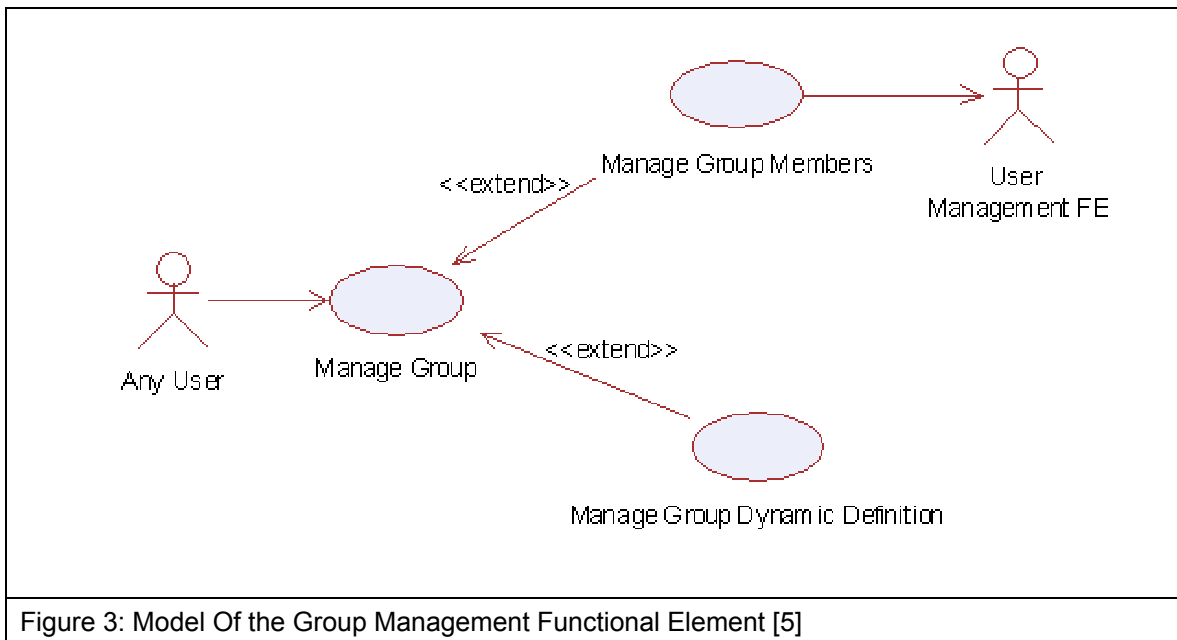


Figure 3: Model Of the Group Management Functional Element [5]

974

975 **2.2.7 Usage Scenarios**

976 **2.2.7.1 Manage Group**

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

979 **2.2.7.1.1 Flow of Events**

980 **2.2.7.1.1.1 Basic Flow**

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

- 982 • If user wants to '**Create Group**', then basic flow 1 is executed.
- 983 • If user wants to '**Retrieve Group**', then basic flow 2 is executed.
- 984 • If user wants to '**Update Group**', then basic flow 3 is executed.
- 985 • If user wants to '**Delete Group**', then basic flow 4 is executed.

986 1: Create Group.

987 1.1: User provides the basic information that is necessary for creating a group.

988 1.2: Functional Element creates the group and the use case ends.

989 2: Retrieve Group.

990 2.1: User provides the necessary information for retrieving the complete group's attributes.

991 2.2: Functional Element returns the group's information and the use case ends.

992 3: Update Group.

993 3.1: User provides the necessary information for updating the group's attributes.

994 3.2: Functional Element updates the group and the use case ends.

995 4: Delete Group.

996 4.1: User provides the necessary information for deleting a particular group.

997 4.2: Functional Element deletes the group and the use case ends.

998 **2.2.7.1.1.2 Alternative Flows**

999 1: Group Exist.

1000 1.1: In basic flow 1.2, Functional Element detects an identical group. Functional Element  
1001 returns an error message and the use case ends.

1002 2: Group Does Not Exist.

1003 2.1: In basic flow 2.2, 3.2 and 4.2, Functional Element cannot find a group that matches the  
1004 user's criteria. Functional Element returns an error message and the use case ends.

1005 3: Save Updated Information.

1006 3.1: In basic flow 1.2, 2.2, 3.2 and 4.2, Functional Element fails to save the updated  
1007 information. Functional Element returns an error message and the use case ends.

1008 **2.2.7.1.2 Special Requirements**

1009 None.

1010 **2.2.7.1.3 Pre-Conditions**

1011 None.

1012 **2.2.7.1.4 Post-Conditions**

1013 None.

1014 **2.2.7.2 Manage Group Members**

1015 **2.2.7.2.1 Description**

1016 This use case is an extension of the manage group use case. Specifically, it describes the  
1017 scenarios to manage members in the group.

1018 **2.2.7.2.2 Flow of Events**

1019 **2.2.7.2.2.1 Basic Flow**

1020 This use case starts when the user wants to manage members in a group.

- 1021
- If user wants to '**Create Members In A Group**', then basic flow 1 is executed.
- 1022
- If user wants to '**Retrieve Members From A Group**', then basic flow 2 is executed.
- 1023
- If user wants to '**Delete Members From A Group**', then basic flow 3 is executed.

- 1024 1: Create Members In A Group.
- 1025 1.1: User provides the necessary information for retrieving the group.
- 1026 1.2: Functional Element adds members to the group and the use case ends.
- 1027 2: Retrieve Members In A Group.
- 1028 2.1: User provides the necessary information for retrieving the group.
- 1029 2.2: Functional Element returns the members and the use case ends.
- 1030 3: Delete Members From Group.
- 1031 3.1: User provides the necessary information for retrieving the group.
- 1032 3.2: User provides the necessary information for deleting members in the group.
- 1033 3.3: Functional Element deletes members from group and the use case ends.
- 1034 **2.2.7.2.2 Alternative Flows**
- 1035 1: Group Does Not Exist.
- 1036 1.1: In basic flow 1.1, 2.1 and 3.1, Functional Element cannot find the group requested.  
1037 Functional Element returns an error message and the use case ends.
- 1038 2: Members Does Not Exist
- 1039 2.1: In basic flow 3.3, the Functional Element attempts to delete a non-existence member.  
1040 Functional Element returns an error message and the use case ends.
- 1041 **2.2.7.2.3 Special Requirements**
- 1042 None.
- 1043 **2.2.7.2.4 Pre-Conditions**
- 1044 None.
- 1045 **2.2.7.2.5 Post-Conditions**
- 1046 None.

1047 **2.2.7.3 Manage Group Dynamic Definition**

1048 **2.2.7.3.1 Description**

1049 This use case describes scenario involved in managing the dynamic group definition.

1050 **2.2.7.3.2 Flow of Events**

1051 **2.2.7.3.2.1 Basic Flow**

1052 This use case starts when the user wants to manage dynamic group definition. This include  
1053 create, retrieve, update and delete dynamic group definition.

- 1054 • If user wants to '**Create Dynamic Definition For A Group**', then basic flow 1 is  
1055 executed.
- 1056 • If user wants to '**Retrieve Dynamic Definition For A Group**', then basic flow 2 is  
1057 executed.
- 1058 • If user wants to '**Delete Dynamic Definition For A Group**', then basic flow 3 is  
1059 executed.
- 1060 • If user wants to '**Update Dynamic Definition For A Group**', then basic flow 4 is  
1061 executed.

1062

1063 1: Create Dynamic Definition For A Group.

1064 1.1: User provides the additional definition for the group.

1065 1.2: Functional Element creates the additional definition for the group and the use case ends.

1066 2: Retrieve Dynamic Definition For A Group.

1067 2.1: User provides the necessary information to retrieve a particular group.

1068 2.2: Functional Element returns the additional definition for the group and the use case ends.

1069 3: Delete Dynamic Definition For Group.

1070 3.1: User provides the necessary information to retrieve a particular group.

1071 3.2: Functional Element deletes the dynamic definition belonging to the group and the use  
1072 case ends.

1073 4: Update Dynamic Definition For Group.

- 1074 4.1: User provides the necessary information to retrieve a particular group.
- 1075 4.2: User provides the necessary dynamic definition that needs to be updated.
- 1076 4.3: Functional Element update the dynamic definition and the use case ends.
- 1077 **2.2.7.3.2 Alternative Flows**
- 1078 1: Group Does Not Exist.
- 1079 1.1: In basic flow 1.1, 2.1, 3.1 and 4.1, Functional Element cannot find the group specified.  
1080 Functional Element returns an error message and the use case ends.
- 1081 2: Dynamic Group Definition Already Exists.
- 1082 2.1: In basic flow 1.2, Functional Element returns the error message and the use case ends.
- 1083 3: Dynamic Group Definition Does Not Exist.
- 1084 3.1: In basic flow 4.3, Functional Element cannot update the dynamic group definition.  
1085 Functional Element returns an error message and the use case ends.
- 1086 **2.2.7.3.3 Special Requirements**
- 1087 None.
- 1088 **2.2.7.3.4 Pre-Conditions**
- 1089 None.
- 1090 **2.2.7.3.5 Post-Conditions**
- 1091 None.

## 1092 **2.3 Identity Management Functional Element**

### 1093 **2.3.1 Motivation**

1094 As secured Web Services become rampant, with each having its own authentication and  
1095 authorisation management, users are finding it difficult to keep track of their accounts and  
1096 passwords. Through the use of Identity Management, users can now voluntarily establish links  
1097 between their accounts so that they need not sign in multiple times to access enterprise-level  
1098 Web Services. This mechanism is known as Single Sign-On (SSO). SSO can further be extended  
1099 to access Web Services from across different business organisations that have prior agreements  
1100 to trust and transact with each other (also known as a circle of trust). This mechanism, which  
1101 involves federating and signing-in of identity's accounts across different trusted organisations, is  
1102 known as Federated Identity Single Sign-On.

1103

1104 Identity Management is about the management of information pertaining to an entity as well as  
1105 the process of identification, authentication and authorization of resources to that entity.

1106

1107 Identity management generally covers the following aspects:

- 1108 • Basic user accounts management facilities
- 1109 • User authentication mechanism(s)
- 1110 • User authorisation mechanism(s)
- 1111 • Generation of audit trails for user activities

1112

1113 This Functional Element fulfills the following requirements from the Functional Elements  
1114 Requirements, Working Draft 01a:

- 1115 • Primary Requirements
  - 1116 • SECURITY-001,
  - 1117 • SECURITY-003 (all),
  - 1118 • SECURITY -004 (all),
  - 1119 • SECURITY -040 and
  - 1120 • SECURITY -041.
- 1121 • Secondary Requirements
  - 1122 • None

1123

## 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.

1125

1126 The following terms mentioned in this document are used in accordance with the terms defined in  
 1127 the Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) v1.1  
 1128 specification.

- 1129
- Assertion [section 2.3.2]

- 1130 • AudienceRestrictionCondition [section 2.3.2.1.3]
- 1131 • AuthenticationQuery [section 3.3.3]
- 1132 • AuthenticationStatement [section 2.4.3]
- 1133 • KeyInfo [section 5.4.5]
- 1134 • Request [section 3.2.2]
- 1135 • Response [section 3.4.2]
- 1136 • Subject [section 2.4.2.1]

1137

### 1138 **2.3.3 Key Features**

1139 Implementations of the Identity Management Functional Element are expected to provide the  
1140 following key features:

- 1141 5. The Functional Element MUST have the mechanism to access an Identity Repository.
- 1142 6. The Functional Element MUST provide the capability to manage the creation and deletion of  
1143 instances of Identity in the said Identity Repository.
- 1144 7. The Functional Element MUST have the mechanisms to manage all the information  
1145 (attribute values) stored in such Identities. This includes the capability to:
  - 1146 7.1. Retrieve and update attribute's values belonging to a Identity,
  - 1147 7.2. Encrypt sensitive user information,
  - 1148 7.3. Authenticate a user, and
  - 1149 7.4. Assign/Unassign Access Policy (or Policies).
- 1150 *Example: Different levels of privileges to access protected resources.*
- 1151 8. As part of Key Feature (3.3), the authentication of an Identity MUST be achieved at least  
1152 through the use of a password.
- 1153 9. As part of Key Feature (3.3), the Functional Element MUST also provide the capability to  
1154 use an Assertion Authority for Single Sign-On (SSO) authentication.
- 1155 10. As part of Key Feature (5), the SSO message exchange and protocol MUST use an  
1156 approved standard.
- 1157 11. As part of Key Feature (3.4), a mechanism MUST be provided to verify the Identity's Access  
1158 Policy on protected Resources.
- 1159 12. The Functional Element MUST provide the capability to create audit trails.

1160 *Example: Timestamp of an Identity's access to Resources.*

1161

1162 In addition, the following key features could be provided to enhance the Functional Element  
1163 further:

- 1164 1. The Functional Element MAY provide an Identity Repository.

- 1165 2. If Key Feature (1) is provided, the Functional Element MUST provide the capability to  
1166 manage the creation and deletion of instances of Identities based on a pre-defined structure.
- 1167 3. The Functional Element MAY provide additional storage in the Identity Repository for an  
1168 Identity to customise its preferences.
- 1169 *Example: Identity's preferred subscription of notifications/alerts for news.*
- 1170 4. The Functional Element MAY provide a capability to use an Identity Provider for Federated  
1171 Identity SSO authentication.
- 1172 5. If Key Feature (4) is provided, the Federated Identity SSO message exchange and protocol  
1173 MUST use an approved standard.
- 1174

1175 **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.

1176

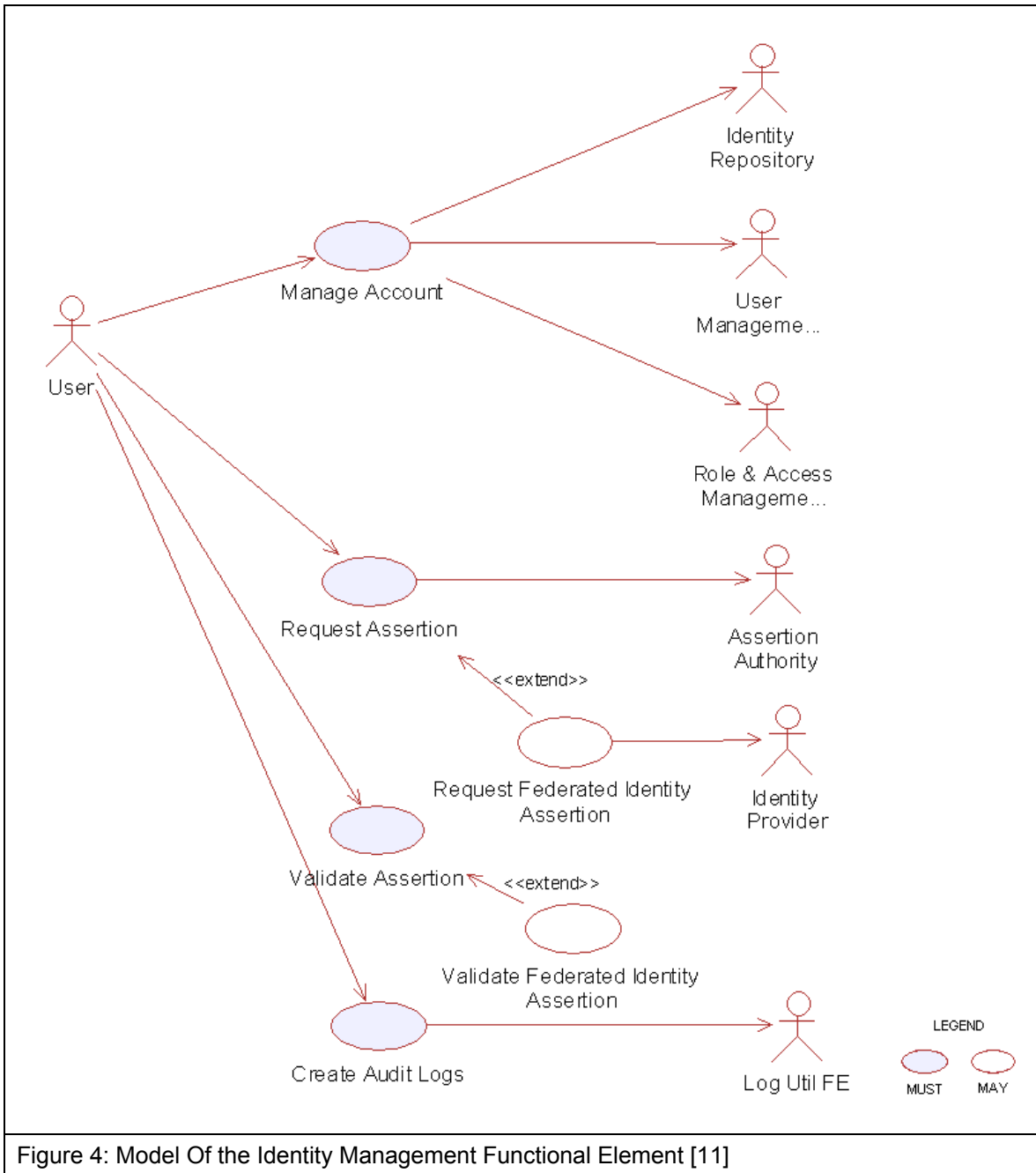
1177 **2.3.5 Related Technologies and Standards**

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"> <li>• Assertion Schema</li> <li>• Protocol Schema</li> </ul>
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

1178

1179

2.3.6 Model



1182 **2.3.7 Usage Scenarios**

1183 **2.3.7.1 Manage Account**

1184 **2.3.7.1.1 Description**

1185 This use case describes the creation/retrieval/update/deletion of an identity's account. An  
1186 identity's account usually consists of two elements: i) the user information and ii) the associated  
1187 access policy.

1188 As Identity Management Functional Element leverages on the User Management and Role-and-  
1189 Access Management Functional Element to provide for these functionalities, please refer to  
1190 sections 2.15 User Management Functional Element and 2.8 Role and Access Management  
1191 Functional Element use cases for details.

1192 **2.3.7.2 Request Assertion**

1193 **2.3.7.2.1 Description**

1194 This use case describes the composition of either 1) an authentication query or 2) an  
1195 authorisation decision query and sending it to the assertion authority.

1196 **2.3.7.2.2 Flow of Events**

1197 **2.3.7.2.2.1 Basic Flow**

1198 This use case starts when the user wants to compose a query to the assertion authority.

1199 If the user requests for an authentication query, then sub-flow 1 is executed.

1200 If the user requests for an authorisation decision query, then sub-flow 2 is executed.

1201 1: Request for Authentication Assertion

1202 1.1: The user composes a valid SAML Request with an AuthenticationQuery and sends it to  
1203 the assertion authority.

1204 1.2: The user waits for an SAML Response from the assertion authority.

1205 1.3: The user obtains the SAML Assertion from the SAML Response and use case ends.

1206 2: Request for Authorisation Decision Assertion

- 1207 2.1: The user composes a valid SAML Request with an AuthorizationDecisionQuery and  
1208 sends it to the assertion authority.
- 1209 2.2: The user waits for an SAML Response from the assertion authority.
- 1210 2.3: The user obtains the SAML Assertion from the SAML Response and use case ends.
- 1211 **2.3.7.2.2 Alternative Flows**
- 1212 1: Invalid Request
- 1213 1.1: If in basic flow 1.1 or 2.1, if any of the parameters passed into the request is invalid, the  
1214 Functional Element flag an exception and use case ends.
- 1215 2: Error message from assertion authority
- 1216 2.1: If in basic flow 1.3 or 2.3, the assertion authority is unable to return an assertion (e.g.  
1217 user has not logged on etc.), it returns an error code and an error message.
- 1218 2.2: The Functional Element flag an error with the error message attached and use case  
1219 ends.
- 1220 **2.3.7.2.3 Special Requirements**
- 1221 None.
- 1222 **2.3.7.2.4 Pre-Conditions**
- 1223 None.
- 1224 **2.3.7.2.5 Post-Conditions**
- 1225 None.
- 1226 **2.3.7.3 Validate Assertion**
- 1227 **2.3.7.3.1 Description**
- 1228 This use case describes the validation of either 1) the Authentication Assertion or 2) the  
1229 Authorisation Decision Assertion

1230 **2.3.7.3.2 Flow of Events**

1231 **2.3.7.3.2.1 Basic Flow**

1232 This use case starts when the user wants to check if the assertion it is a valid assertion from the  
1233 assertion authority.

1234 1: The user passes the assertion to the Functional Element for validation.

1235 2: The Functional Element checks if the assertion is signed by the assertion authority.

1236 3: The Functional Element checks for an un-expired assertion.

1237 4: The Functional Element checks if the assertion has an AudienceRestrictionCondition and  
1238 verifies that the service provider using the Functional Element is in the audience list.

1239 5: Based on the type of assertion, one of the sub-flows is executed.

1240 • If the user wants to check for a valid authentication assertion, then sub-flow 5.1 is executed.

1241 • If the user wants to check for a valid authorisation decision assertion, then sub-flow 5.2 is  
1242 executed.

1243 5.1: Validate Authentication Statement

1244 5.1.1: The Functional Element checks if the assertion has indeed an  
1245 AuthenticationStatement.

1246 5.1.2: The Functional Element checks if the Subject in the AuthenticationStatement  
1247 matches the userid of the principal.

1248 5.1.3: The Functional Element verifies the Subject with its KeyInfo.

1249 5.1.4: The Functional Element returns the status code to the user and use case ends.

1250 5.2: Validate Authorisation Decision Statement

1251 5.2.1: The Functional Element checks if the assertion has indeed an  
1252 AuthorizationDecisionStatement.

1253 5.2.2: The Functional Element checks if the Resource in the  
1254 AuthorizationDecisionStatement matches the id of the requested resource.

1255 5.2.3: The Functional Element determines if the decision is Permit.

- 1256            5.2.4: The Functional Element returns the status code to the user and use case ends.
- 1257    **2.3.7.3.2.2 Alternative Flows**
- 1258    1: Signature Error
- 1259            1.1: If in basic flow 2, the Functional Element is unable to verify that the signature is from the  
1260            assertion authority, it returns an error and use case ends.
- 1261    2: Expired Assertion
- 1262            2.1: If in basic flow 3, the Functional Element finds that the assertion has already expired, it  
1263            returns an error and use case ends.
- 1264    3: Audience Error
- 1265            3.1: If in basic flow 4, the service provider is not in the AudienceRestrictionCondition, the  
1266            Functional Element returns an error and use case ends.
- 1267    4: Invalid Authentication Assertion
- 1268            4.1: If in basic flow 5.1.1, the Functional Element is unable to find an  
1269            AuthenticationStatement, it returns an error and use case ends.
- 1270    5: Mismatch Subject
- 1271            5.1: If in basic flow 5.1.2, the Functional Element is unable to match the Subject in  
1272            AuthenticationStatement, it returns an error and use case ends.
- 1273    6: Subject Error
- 1274            6.1: If in basic flow 5.1.3, the Functional Element is unable to verify the Subject with the  
1275            KeyInfo, it returns an error and use case ends.
- 1276    7: Invalid Authorisation Decision Assertion
- 1277            7.1: If in basic flow 5.2.1, the Functional Element is unable to find an  
1278            AuthorizationDecisionStatement, it returns an error and use case ends.
- 1279    8: Mismatch Resource
- 1280            8.1: If in basic flow 5.2.2, the Functional Element is unable to match the resource in  
1281            AuthorizationDecisionStatement, it returns an error and use case ends.

1282 **2.3.7.3.3 Special Requirements**

1283 None.

1284 **2.3.7.3.4 Pre-Conditions**

1285 None.

1286 **2.3.7.3.5 Post-Conditions**

1287 None.

1288 **2.3.7.4 Create Audit Logs**

1289 **2.3.7.4.1 Description**

1290 This use case describes logging all identity management activities for audit purposes.

1291 **2.3.7.4.2 Flow of Events**

1292 **2.3.7.4.2.1 Basic Flow**

1293 This use case starts when any of other Functional Element use cases are triggered.

1294 1: The Functional Element opens an audit log file.

1295 2: The Functional Element writes a timestamp identity management activity message into the  
1296 audit log file.

1297 3: The Functional Element closes the audit log file and the use case ends.

1298 **2.3.7.4.2.2 Alternative Flows**

1299 1: Log File Not Created

1300 1.1: If in the basic flow 1, the Functional Element cannot open the audit file, it creates a new  
1301 audit file and use case continues.

1302 2: Error Writing Log

1303 2.1: If in the basic flow 2, the Functional Element has error writing to file, it will flag an  
1304 exception and the use case ends.

1305 **2.3.7.4.3 Special Requirements**

1306 None.

1307 **2.3.7.4.4 Pre-Conditions**

1308 None.

1309 **2.3.7.4.5 Post-Conditions**

1310 None.

## 1311 2.4 Log Utility Functional Element

### 1312 2.4.1 Motivation

1313 In a Web Service-enabled implementation, the Log Utility Functional Element can help to  
1314 organise the diagnostic output that may be generated by the implementation. In order to achieve  
1315 that, the following capabilities should be provided. They include:

- 1316 • Logging information into different data sources,
- 1317 • Allowing user defined log format to be used,
- 1318 • Capability for storing log information, and
- 1319 • Providing the capability to analyse the information log.

1320

1321 This Functional Element fulfills the following requirements from the Functional Elements  
1322 Requirements, Working Draft 01a:

- 1323 • Primary Requirements
  - 1324 • MANAGEMENT-007, [*\*To be fulfilled in next working draft*]
  - 1325 • MANAGEMENT-110,
  - 1326 • MANAGEMENT-112 to MANAGEMENT-114, and
  - 1327 • PROCESS-009.
- 1328 • Secondary Requirements
  - 1329 • MANAGEMENT-006,
  - 1330 • MANAGEMENT-095,
  - 1331 • MANAGEMENT-111,
  - 1332 • PROCESS-008,
  - 1333 • PROCESS-115, and
  - 1334 • PROCESS-118.

1335

### 1336 2.4.2 Terms Use d

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.
--------------	---

1337

### 1338 **2.4.3 Key Features**

1339 Implementations of the Log Utility Functional Element are expected to provide the following key  
1340 features:

- 1341 1. The Functional Element MUST provide the capability to define a Log Category and manage  
1342 it. This includes:
  - 1343 1.1. The capability to define the format of the log information,
  - 1344 1.2. The capability to choose the data source to logged to, and
  - 1345 1.3. The capability to define the name of the log category.
- 1346 2. The Functional Element MUST provide the capability to manage logging of events/records.  
1347 This includes:
  - 1348 2.1. The capability to insert a new record into the log,  
1349 *Examples of a log record could include events, transactions status, usages status or*  
1350 *users' activities.*
  - 1351 2.2. The capability to search and return result sets of search on log records, and
  - 1352 2.3. The capability to archive or delete obsolete log records.

1353

1354 In addition, the following key features could be provided to enhance the Functional Element  
1355 further:

- 1356 1. The Functional Element MAY also provide the capability to perform conditional search or  
1357 viewing of log records.
- 1358 2. The Functional Element MAY provide the capability to perform basic statistical analysis on  
1359 log records. Basic statistical analysis capabilities include:
  - 1360 2.1. Minimum and maximum value calculations on numerical values,
  - 1361 2.2. Mean values calculations on numerical values, and
  - 1362 2.3. Standard deviation calculations on numerical values.

1363 *Note: Report Structure Creation, Generation and Notification are expected to be added in the*  
1364 *next Working Draft version under this optional key features.*

1365

### 1366 **2.4.4 Interdependencies**

1367 None

### 1368 **2.4.5 Related Technologies and Standards**

1369 None

1370 **2.4.6 Model**

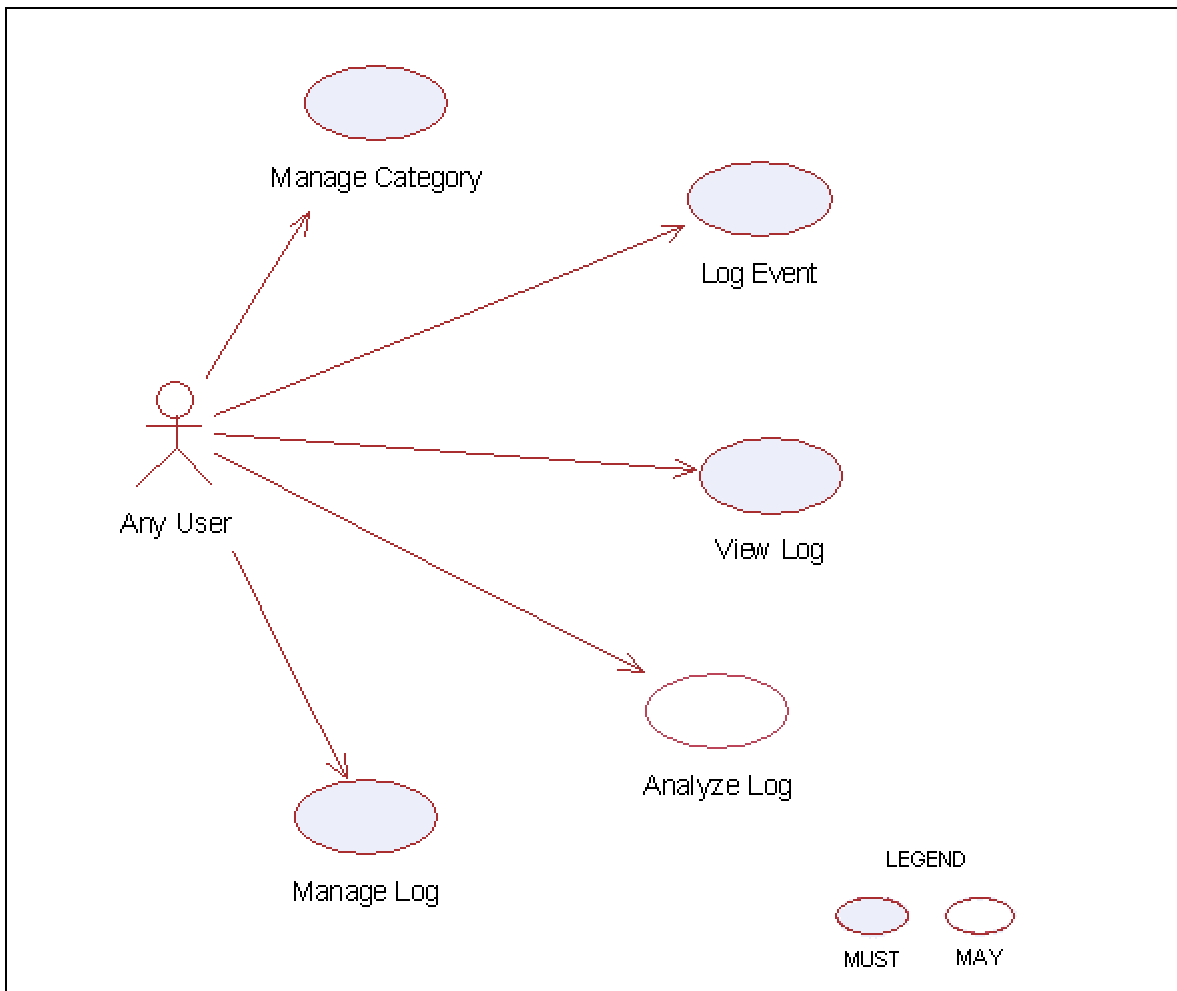


Figure 5: Model Of the Log Utility Functional Element [12]

1371

1372 **2.4.7 Usage Scenarios**

1373 **2.4.7.1 Manage Category**

1374 **2.4.7.1.1 Description**

1375 This use case allows any user to manage log category. Log category defines the data fields that  
1376 the user wants to log.

1377 **2.4.7.1.2 Flow of Events**

1378 **2.4.7.1.2.1 Basic Flow**

1379 This use case starts when users wants to manage the log category.

1380 1: The users send the request to the Functional Element. The request contains the operations  
1381 the users want to perform.

1382 2: The Functional Element receives the request. Based on the operation specified, one of the  
1383 following sub-flows is executed.

- 1384 • If the operation is '**Create Log Category**', then sub-flow 2.1 is executed.
- 1385 • If the operation is '**Retrieve Log Category Information**', then sub-flow 2.2 is executed.
- 1386 • If the operation is '**Delete Log Category**', then sub-flow 2.3 is executed.

1387 2.1: Create Log Category.

1388 2.1.1: The Functional Element gets the following data from the users.

- 1389 • Category name
- 1390 • The definition of category
- 1391 • The data source where the log is located

1392 2.1.2: The Functional Element checks the uniqueness of the category name.

1393 2.1.3: The Functional Element connects to the data source according to the specified  
1394 data source.

1395 2.1.4: The Functional Element creates the empty log in the data source.

1396 2.1.5: The Functional Element writes the category name and its definition in its own  
1397 category definition record and the use case end.

1398 2.2: Retrieve Log Category Information.

1399 2.2.1: The Functional Element gets the category name.

1400 2.2.2: The Functional Element checks the existence of this category.

1401 2.2.3: The Functional Element retrieves the definition of this category.

1402 2.2.4: The Functional Element returns the definition of this category to the user and the  
1403 use case ends.

1404 2.3: Delete Log Category.

1405 2.3.1: The Functional Element gets the category name.

- 1406 2.3.2: The Functional Element checks the existence of this category.
- 1407 2.3.3: The Functional Element deletes its own records of category definition and the use  
1408 case ends.
- 1409 **2.4.7.1.2.2 Alternative Flows**
- 1410 1: Category Already Exists.
- 1411 1.1: In sub-flow 2.1.2, if the category name is already used by others, the Functional Element  
1412 returns an error message and the use case ends.
- 1413 2: Data Source Not Available.
- 1414 2.1: In sub-flow 2.1.3, if the data source is not available, the Functional Element returns an  
1415 error message and the use case ends.
- 1416 3: Create Log Error.
- 1417 3.1: In sub-flow 2.1.4, if the log cannot be created on the specified data source, the  
1418 Functional Element returns an error message and the use case ends.
- 1419 4: Category Does Not Exist.
- 1420 4.1: In sub-flow 2.2.1 and 2.3.1, the category cannot be found in Functional Element category  
1421 definition, the Functional Element returns an error message and the use case ends.
- 1422 5: Delete Category Error.
- 1423 5.1: In sub-flow 2.3.3, the log category cannot be deleted, the Functional Element returns an  
1424 error message and the use case ends.
- 1425 **2.4.7.1.3 Special Requirements**
- 1426 None
- 1427 **2.4.7.1.4 Pre-Conditions**
- 1428 None.
- 1429 **2.4.7.1.5 Post-Conditions**
- 1430 If the use case was successful, the category definition is saved to the Functional Element and an  
1431 empty log is created in the specified data source. Otherwise, the Functional Element's state is  
1432 unchanged.

1433 **2.4.7.2 Log Event**

1434 **2.4.7.2.1 Description**

1435 The use case allows any user to log any event.

1436 **2.4.7.2.2 Flow of Events**

1437 **2.4.7.2.2.1 Basic Flow**

1438 This use case starts when users want to write to a log.

1439 1: The users provide the event data, category name he/she wants to log to the Functional  
1440 Element.

1441 2: The Functional Element gets the definition of the category.

1442 3: The Functional Element connects the log data source.

1443 4: The Functional Element writes the log record into the end of the log file and the use case ends.

1444 **2.4.7.2.2.2 Alternative Flows**

1445 1: Category Does Not Exist.

1446 1.1: If in basic flow 2, the category that the users want to write does not exist, the Functional  
1447 Element returns an error message and the use case ends.

1448 2: Data Source Not Available.

1449 2.1: If in basic flow 3, the data source is not available, the Functional Element returns an error  
1450 message and the use case ends.

1451 3: Data Not Match.

1452 3.1: If in basic flow 4, the data provided by the users for logging does not match with the  
1453 category definition in the Functional Element, the Functional Element returns an error  
1454 message and the use case ends.

1455 **2.4.7.2.3 Special Requirements**

1456 None.

1457 **2.4.7.2.4 Pre-Conditions**

1458 None.

1459 **2.4.7.2.5 Post-Conditions**

1460 If the use case was successful, the log record is saved to the Functional Element. Otherwise, the  
1461 Functional Element's state is unchanged.

1462 **2.4.7.3 View Log**

1463 **2.4.7.3.1 Description**

1464 The use case allows users to retrieve the log content.

1465 **2.4.7.3.2 Flow of Events**

1466 **2.4.7.3.2.1 Basic Flow**

1467 This use case starts when users want to view a log.

1468 1: The users specify the category name and the search criteria, such as searching by event type  
1469 or searching by time period (starting time and end time).

1470 2: The Functional Element connects to the data storage where the log records are stored.

1471 3: The Functional Element retrieves the log content and returns to the service users and the use  
1472 case ends.

1473 **2.4.7.3.2.2 Alternative Flows**

1474 1: Search Criteria Not Valid.

1475 1.1: If in basic flow 1 and 3, the search criteria specified by the users is invalid for Search  
1476 Service, the Functional Element returns an error message and the use case ends.

1477 **2.4.7.3.3 Special Requirements**

1478 None.

1479 **2.4.7.3.4 Pre-Conditions**

1480 None.

1481 **2.4.7.3.5 Post-Conditions**

1482 None.

1483 **2.4.7.4 Analyze Log Data**

1484 **2.4.7.4.1 Description**

1485 The use case allows users to analyze the log data, i.e., to get statistics of certain event. The  
1486 service users may get statistical results on the log data, such as the cumulative events and mean  
1487 of two numerical values.

1488 **2.4.7.4.2 Flow of Events**

1489 **2.4.7.4.2.1 Basic Flow**

1490 This use case starts when users want to analyze the log data.

1491 1: The users specify the items to analyze, i.e. field name and category name.

1492 2: The users specify the analysis method, option among max, min and mean.

1493 3: The Functional Element retrieves the definition of the category and validates the parameters  
1494 provided by the users.

1495 4: The Functional Element connects to the data source and retrieves the log data.

1496 5: The Functional Element analyses the log data and does statistics on the data with respect to  
1497 what is specified in Step 1 and 2.

1498 6: The Functional Element returns the analyzed result and the use case ends.

1499 **2.4.7.4.2.2 Alternative Flows**

1500 1: Invalid Item Specified.

1501 1.1: If in basic flow 1, the analyze items specified by the users are invalid, i.e. invalid field and  
1502 invalid data source, the Functional Element returns an error message and the use case ends.

1503 2: Category Does Not Exist.

1504 2.1: If in basic flow 3, the category that the users want to write to does not exist, the  
1505 Functional Element returns an error message and the use case ends.

1506 3: Data Source Not Available.

1507 3.1: If in basic flow 4, the data source is not available, the Functional Element returns an error  
1508 message and the use case ends.

1509 **2.4.7.4.3 Special Requirements**

1510 **2.4.7.4.3.1 Supportability**

1511 Only basic statistic methods of numerical value are supported.

1512 **2.4.7.4.4 Pre-Conditions**

1513 None.

1514 **2.4.7.4.5 Post-Conditions**

1515 None.

1516 **2.4.7.5 Manage Log**

1517 **2.4.7.5.1 Description**

1518 The use case allows users to drop log and backup log.

1519 **2.4.7.5.2 Flow of Events**

1520 **2.4.7.5.2.1 Basic Flow**

1521 The use case starts when the users want to drop and backup a log of a specific data source.

1522 1: The users specify the function name to the Functional Element.

1523 2: Based on the operation specified, one of the following sub-flows is executed.

1524 • If the operation is '**Delete Log**', then sub-flow 2.1 is executed.

1525 • If the operation is '**Backup Log**', then sub-flow 2.2 is executed.

1526 2.1: Delete Log

1527 2.1.1: The Functional Element gets category name from the users.

1528 2.1.2: The Functional Element retrieves the definition of the category.

1529 2.1.3: The Functional Element connects to the corresponding data source.

1530 2.1.4: The Functional Element deletes the log from the data source.

1531 2.2: Backup Log

1532 2.2.1: The Functional Element gets the category name and the destination file name from  
1533 the users.

1534 2.2.2: The Functional Element retrieves the definition of the category.

1535 2.2.3: The Functional Element connects to the corresponding data source.

1536 2.2.4: The Functional Element read the original log and writes it to the destination file.

1537 **2.4.7.5.2 Alternative Flows**

1538 1: Category Does Not Exist.

1539 1.1: If in basic flow 2.1.2 and 2.2.2 the category that the users want to write does not exist,  
1540 the Functional Element returns an error message and the use case ends.

1541 2: Data Source Not Available.

1542 2.1: If in basic flow 2.1.4 and 2.2.4, the data source is not available, the Functional Element  
1543 returns an error message and the use case ends.

1544 **2.4.7.5.3 Special Requirements**

1545 None.

1546 **2.4.7.5.4 Pre-Conditions**

1547 None.

1548 **2.4.7.5.5 Post-Conditions**

1549 None.

1550

## 1551 2.5 Notification Functional Element

### 1552 2.5.1 Motivation

1553 In a Web Service-enabled implementation, timely information is crucial for the management of  
1554 resources that it encompasses. Other uses of this Functional Element include broadcasting of  
1555 information to other services and this could span across both the wired and wireless medium. In  
1556 order to fulfill these needs, this Functional Element will cover the following aspects which include:

- 1557 • Providing the capability to configure and link with the various gateways so as to enable  
1558 messages dissemination, and
- 1559 • Providing the capability to send instantaneous or scheduled messages to the intended  
1560 audiences.

1561

1562 This Functional Element fulfills the following requirements from the Functional Elements  
1563 Requirements, Working Draft 01a:

- 1564 • Primary Requirements
  - 1565 • DELIVERY-003, and
  - 1566 • PROCESS-118.
- 1567 • Secondary Requirements
  - 1568 • MANAGEMENT-205,
  - 1569 • PROCESS-005,
  - 1570 • PROCESS-102,
  - 1571 • PROCESS-107, and
  - 1572 • PROCESS-110.

1573

### 1574 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.

1575

### 1576 2.5.3 Key Features

1577 Implementations of the Notification Functional Element are expected to provide the following key  
1578 features:

- 1579 1. The Functional Element MUST support notifications using both the SMS and SMTP  
1580 protocols.
- 1581 2. The Functional Element MUST provide the capability to configure supported SMS  
1582 gateway(s) and the SMTP servers where applicable.  
1583 *Example: The capability to configure the username and password for SMTP server's*  
1584 *authentication.*
- 1585 3. The Functional Element MUST provide the capability to send notifications to single and  
1586 multiple recipients.
- 1587 4. The Functional Element MUST provide the capability to structure a notification based on the  
1588 selected protocol(s).

1589

1590 In addition, the following key features could be provided to enhance the Functional Element  
1591 further:

- 1592 1. The Functional Element MAY provide the capability to send notifications either instantly or  
1593 based on a pre-defined schedule.
- 1594 2. If Key Feature (1) is provided, the Functional Element MAY also provide the capability to  
1595 send scheduled messages in the following manner:
  - 1596 2.1. Hourly,
  - 1597 2.2. Daily,
  - 1598 2.3. Weekly, and
  - 1599 2.4. Monthly (based on a particular date or particular day of the week).

1600 *Note: The next working draft version will attempt to look at other available protocols.*

1601

### 1602 2.5.4 Interdependencies

1603 None

1604

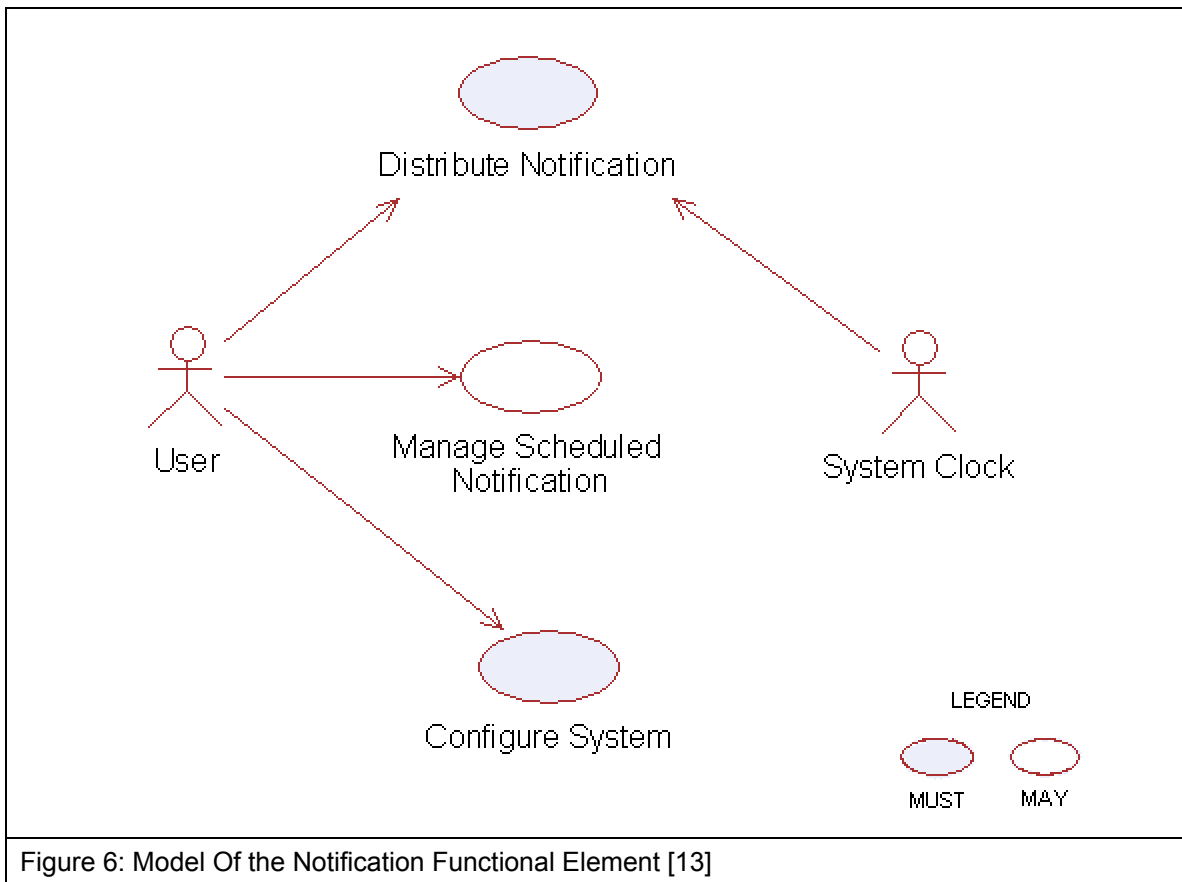
## 2.5.5 Related Technologies and Standards

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.

1605

1606

## 2.5.6 Model



1607 **2.5.7 Usage Scenarios**

1608 **2.5.7.1 Distribute Notification**

1609 **2.5.7.1.1 Description**

1610 This use case allows the Functional Element to distribute messages to intended recipients.

1611 **2.5.7.1.2 Flow of Events**

1612 **2.5.7.1.2.1 Basic Flow**

1613 This use case starts when the service user or system clock wishes to send message to recipient.

1614 1: The Functional Element decides to send messages to recipients. Based on the operation  
1615 specified, one of the following sub-flows is executed.

- 1616
- If the request is '**Initiated By The User**', then sub-flow 1.1 is executed.
  - If the request is '**Initiated By The System Clock**' then sub-flow 1.2 is executed.
- 1617

1618 1.1: Initiated By The User

1619 1.1.1: The Functional Element receives the request from the service user.

1620 1.1.2: The Functional Element validates passed parameters such as message type,  
1621 recipient address, and message key and message length.

1622 1.1.3: The Functional Element checks the availability of the connection.

1623 1.1.4: The Functional Element sends message to recipient(s) and the use case end

1624 1.2 : Initiated By The System Clock

1625 1.2.1: The Functional Element checks scheduled message(s) and end date for scheduled  
1626 message.

1627 1.2.2: Once the Functional Element detects scheduled messages, one of the sub-flows is  
1628 executed.

- 1629
- If the Functional Element detects the scheduled notification is once, the '**Activate Once Notification**' sub-flow 1.2.2.1 is executed.
  - If the Functional Element detects the scheduled notification is daily, the '**Activate Daily Notification**' sub-flow 1.2.2.2 is executed.
- 1631
- 1632

- 1633                   • If the Functional Element detects the scheduled notification is weekly, the  
1634                    '**Activate Weekly Notification**' sub-flow 1.2.2.3 is executed.
- 1635                   • If the Functional Element detects the scheduled notification is Monthly, the  
1636                    '**Activate Monthly Notification**' sub-flow 1.2.2.4 is executed.
- 1637                   1.2.2.1: Activate Once Notification.
- 1638                   1.2.2.1.1: The Functional Element compares the system time with the scheduled  
1639                    message's time and gets notification details if both times are match.
- 1640                   1.2.2.2: Activate Daily Notification.
- 1641                   1.2.2.2.1: The Functional Element compares the system time with the scheduled  
1642                    message's time and gets notification details if both times are match.
- 1643                   1.2.2.3: Activate Weekly Notification.
- 1644                   1.2.2.3.1: The Functional Element compares the system date and time with the  
1645                    scheduled message's date and time and gets notification details if both date &  
1646                    time are match.
- 1647                   1.2.2.4: Activate Monthly Notification.
- 1648                   1.2.2.4.1: The Functional Element compares the system date and time with the  
1649                    scheduled message's date and time and gets notification ID if both date & time  
1650                    are match.
- 1651                   1.2.3: The Functional Element extracts the list of recipient(s) and message(s).
- 1652                   1.2.4: The Functional Element checks the availability of connection.
- 1653                   1.2.5: The Functional Element sends message to recipient(s) and the use case ends.

1654    **2.5.7.1.2.2 Alternative Flows**

1655    1: Unsupported Message Type/Recipient Address/Message.

1656           1.1: If in basic flow 1.1.2, Functional Element detects unsupported message type, recipient  
1657            address or message, the Functional Element returns an error message and the use case  
1658            ends.

1659    2: Connection Fail.

1660           2.1: If in basic flow 1.1.3 and 1.2.4, the Functional Element is unable to detect connection  
1661            type, the Functional Element returns an error message and the use case ends

1662 3: Delete Scheduled Message.

1663 3.1: If in basic flow 1.2.1, if the Functional Element detects that the scheduled message has  
1664 expired, the Functional Element will proceed to delete those messages.

### 1665 **2.5.7.1.3 Special Requirements**

#### 1666 **2.5.7.1.3.1 Supportability**

1667 None

#### 1668 **2.5.7.1.4 Pre-Conditions**

1669 None.

#### 1670 **2.5.7.1.5 Post-Conditions**

1671 None.

### 1672 **2.5.7.2 Manage Scheduled Notification**

#### 1673 **2.5.7.2.1 Description**

1674 This use case allows the service user to maintain the notification information. This includes  
1675 adding, changing and deleting notification information from the Functional Element.

#### 1676 **2.5.7.2.2 Flow of Events**

##### 1677 **2.5.7.2.2.1 Basic Flow**

1678 This use case starts when the service user wishes to schedule notification message(s).

1679 1: The Functional Element requests the service user to specify the function he/she would like to  
1680 perform (such as create, update and delete notification message).

1681 2: Once the Functional Element user provides the requested information, one of the sub-flows is  
1682 executed.

1683 • If the service user provides '**Create Notification**', then sub-flow 2.1 is executed.

1684 • If the service user provides '**Delete Notification**', then sub-flow 2.2 is executed.

1685 2.1 Create Notification

1686 2.1.1: The Functional Element receives the request from the service user.

1687 2.1.2: The Functional Element validates passed parameters such as schedule type,  
1688 message type, recipient address, message key and the message length.

1689 2.1.3: The Functional Element generates and assigns a unique Notification ID and adds  
1690 the notification information to the Functional Element and ends use case.

1691 2.2: Delete Notification

1692 2.2.1: The Functional Element requests the service user to provide the Notification  
1693 information.

1694 2.2.2: The Functional Element retrieves the existing Notification information.

1695 2.2.3: The Functional Element deletes the Notification record and use case ends.

1696 **2.5.7.2.2 Alternative Flows**

1697 1: Invalid Parameters.

1698 1.1: If in basic flow 2.1.2, if the Functional Element detects invalid parameters such as  
1699 schedule type, date & time, recipient address, message key and message, the Functional  
1700 Element returns an error message and the use case ends.

1701 **2.5.7.2.3 Special Requirements**

1702 None.

1703 **2.5.7.2.4 Pre-Conditions**

1704 None.

1705 **2.5.7.2.5 Post-Conditions**

1706 If the use case was successful, the schedule message information is added to Functional  
1707 Element. Otherwise, the Functional Element's state is unchanged.

1708 **2.5.7.3 Configure System**

1709 **2.5.7.3.1 Description**

1710 This use case allows the service user to maintain the notification Functional Element behaviors.  
1711 This includes configuration of supported Notification Channel, Default Notification Channel,  
1712 Schedule Types, and SMS and SMTP Gateway.

1713 **2.5.7.3.2 Flow of Events**

1714 **2.5.7.3.2.1 Basic Flow**

1715 1: The Functional Element requests the service user to specify or configure the function he/she  
1716 would like to perform (such as create, update and delete configuration parameters).

1717 2: Once the Functional Element user provides the requested information, one of the sub-flows is  
1718 executed.

- 1719 • If user wishes to configure '**Notification Channel**', then sub-flow 2.1 is executed.
- 1720 • If user wishes to configure '**Default Notification Channel**', then sub-flow 2.2 is executed.
- 1721 • If user wishes to configure '**Schedule Types**', then sub-flow 2.3 is executed.
- 1722 • If user wishes to configure '**SMTP server and SMS Gateway**', then sub-flow 2.4 is  
1723 executed.

1724 2.1 Notification Channel.

1725 2.1.1: The Functional Element receives the request from the service user.

1726 2.1.2: The Functional Element validates passed parameters such as Notification Channel  
1727 information.

1728 2.1.3: The Functional Element generates and assigns a unique Notification Channel ID  
1729 and adds the notification information to the Functional Element and the use case ends.

1730 2.2: Default Notification Channel.

1731 2.2.1: The Functional Element requests the service user to provide the Default  
1732 Notification information.

1733 2.2.2: The Functional Element validates passed parameters such as Default Notification  
1734 Channel information.

1735 2.2.3: The Functional Element updates existing Default Notification or create new Default  
1736 Notification information and the use case ends.

1737 2.3 Schedule Types.

1738 2.3.1: The Functional Element receives the request from the service user.

1739 2.3.2: The Functional Element validates passed parameters such as Schedule Type.

1740 2.3.3: The Functional Element generates and assigns a unique Schedule Type ID and  
1741 adds the Schedule Type information to the Functional Element and the use case ends.

1742 2.4: SMTP server and SMS Gateway.

1743 2.4.1: The Functional Element requests the service user to provide the SMTP server and  
1744 SMS Gateway information.

1745 2.4.2: The Functional Element validates passed parameters such as SMTP server and  
1746 SMS Gateway information.

1747 2.4.3: The Functional Element updates existing SMTP server and SMS Gateway or  
1748 create new SMTP server and SMS Gateway information and the use case ends.

1749 **2.5.7.3.2 Alternative Flows**

1750 1: Invalid Parameters.

1751 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  
1752 parameters such as Notification Channel, Default Notification Channel, and SMTP server,  
1753 Schedule Types and SMS Gateway information, the Functional Element returns an error  
1754 message and the use case ends

1755 **2.5.7.3.3 Special Requirements**

1756 None.

1757 **2.5.7.3.4 Pre-Conditions**

1758 None.

1759 **2.5.7.3.5 Post-Conditions**

1760 None

## 1761 2.6 Phase and Lifecycle Management Functional Element

### 1762 2.6.1 Motivation

1763 The Phase and Lifecycle Management Functional Element is expected to be an integral part of  
1764 the User Access Management (UAM) functionalities that is expected to be needed by a Web  
1765 Service-enabled implementation. This FE is expected to fulfill the needs arising out of managing  
1766 the dynamic status of user information across the whole lifecycle. As such it will cover aspects  
1767 that include:

- 1768 • Basic lifecycle management facilities,
- 1769 • Basic phase management facilities, and
- 1770 • Management of user information in phases across the whole lifecycle.

1771

1772 This Functional Element fulfills the following requirements from the Functional Elements  
1773 Requirements, Working Draft 01a:

- 1774 • Primary Requirements
  - 1775 • MANAGEMENT-070 to MANAGEMENT-078
- 1776 • Secondary Requirements
  - 1777 • None

1778

### 1779 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	<p>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.</p> <ul style="list-style-type: none"> <li>• Requirements Analysis</li> <li>• Design, Construction</li> <li>• Testing (Validation)</li> <li>• Installation</li> <li>• Operation</li> <li>• Maintenance</li> <li>• Retirement.</li> </ul>
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"> <li>• Defining a set of basic user information that should be stored in any enterprise application.</li> <li>• Providing a means to extend this basic set of user information when needed..</li> <li>• Simplifying management by grouping related users together through certain criteria.</li> <li>• Having the flexibility of adopting both coarse/fine grain access control.</li> </ul>

1780

### 1781 **2.6.3 Key Features**

1782 Implementations of the Phase and Lifecycle Management Functional Element are expected to  
 1783 provide the following key features:

- 1784 1. The Functional Element MUST provide basic structures based on a set of pre-defined  
 1785 attributes for Lifecycle and Phase.
- 1786 2. The Functional Element MUST provide the capability to manage the creation and deletion of  
 1787 instances of Lifecycle and Phase based on the pre-defined structures.
- 1788 3. The Functional Element MUST provide a means to manage the lifecycles and phases  
 1789 contained within. This includes:
  - 1790 3.1. The capability to retrieve and update a lifecycle or phase
  - 1791 3.2. The capability to add/remove phases from a lifecycle

- 1792 4. The Functional Element MUST provide a mechanism to manage the collection of users in a  
 1793 Phase. This includes:
- 1794 4.1. The capability to assign and un-assign users belonging to a Phase.
- 1795 4.2. The users could be individual Users or Groups.
- 1796 5. The Functional Element MUST provide a mechanism for managing Groups across different  
 1797 application domains.
- 1798 *Example: Namespace control mechanism*
- 1799

1800 **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..

1801

1802 **2.6.5 Related Technologies and Standards**

1803 None.

1804 **2.6.6 Model**

1805

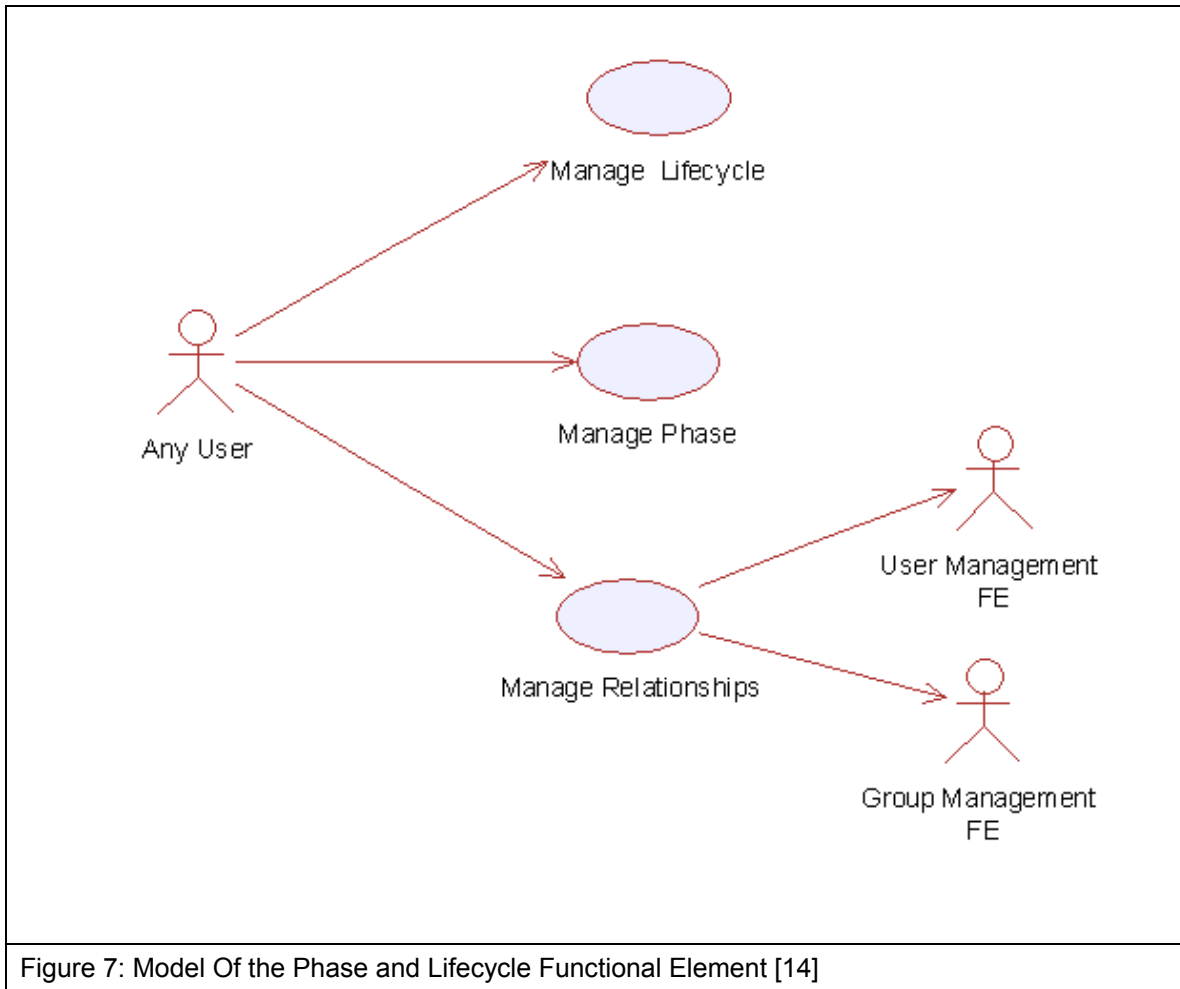


Figure 7: Model Of the Phase and Lifecycle Functional Element [14]

1806 **2.6.7 Usage Scenarios**

1807 **2.6.7.1 Manage Lifecycle**

1808 **2.6.7.1.1 Description**

1809 This use case is used to create, update, retrieve and delete the lifecycle.

1810 **2.6.7.1.2 Flow of Events**

1811 **2.6.7.1.2.1 Basic Flow**

1812 This use case starts when the user wants to manage phase in lifecycle.

- 1813 • If user wants to **'Create Lifecycle'**, then basic flow 1 is executed.
- 1814 • If user wants to **'Retrieve Lifecycle'**, then basic flow 2 is executed.

- 1815       • If user wants to '**Update Lifecycle**', then basic flow 3 is executed.
- 1816       • If user wants to '**Delete Lifecycle**', then basic flow 4 is executed.
- 1817   1: Create Lifecycle.
- 1818       1.1: User provides information to create lifecycle.
- 1819       1.2: Functional Element creates lifecycle and the use case ends.
- 1820   2: Retrieve Lifecycle
- 1821       2.1: User provides the lifecycle name, lifecycle namespace.
- 1822       2.2: Functional Element returns the lifecycle information and the use case ends.
- 1823   3: Update Lifecycle.
- 1824       3.1: User provides the lifecycle information.
- 1825       3.2: Functional Element updates the lifecycle-phase and the use case ends.
- 1826   4: Delete Lifecycle.
- 1827       4.1: User provides lifecycle name and lifecycle namespace.
- 1828       4.2: Functional Element deletes the lifecycle and the use case ends.

1829   **2.6.7.1.2.2 Alternative Flows**

- 1830   1: Lifecycle Does Not Exist.
- 1831       1.1: In basic flow 2.1, 3.1 and 4.1, if lifecycle can not be found based on lifecycle name and  
1832       lifecycle namespace provided by user, Functional Element returns an error message and the  
1833       use case ends.
- 1834   2: Creation Of Lifecycle Fails.
- 1835       2.1: In basic flow 1.2, if lifecycle cannot be created, the Functional Element returns an error  
1836       message and the use case ends

1837   **2.6.7.1.3 Special Requirements**

- 1838   None.

1839 **2.6.7.1.4 Pre-Conditions**

1840 None.

1841 **2.6.7.1.5 Post-Conditions**

1842 None.

1843 **2.6.7.2 Manage Phase**

1844 **2.6.7.2.1 Description**

1845 This use case describes the management of different phases in a project.

1846 **2.6.7.2.2 Flow of Events**

1847 **2.6.7.2.2.1 Basic Flow**

1848 This use case starts when the user wants to manage phase.

- 1849 • If user wants to '**Create Phase**', then basic flow 1 is executed.
- 1850 • If user wants to '**Retrieve Phase**', then basic flow 2 is executed.
- 1851 • If user wants to '**Update Phase**', then basic flow 3 is executed.
- 1852 • If user wants to '**Delete Phase**', then basic flow 4 is executed.

1853 1: Create Phase.

1854 1.1: User provides information to create phase.

1855 1.2: Functional Element creates phase and the use case ends.

1856 2: Retrieve Phase.

1857 2.1: User provides phase name, lifecycle name and lifecycle namespace.

1858 2.2: Functional Element returns the phase information and the use case ends.

1859 3: Update Phase.

1860 3.1: User provides the phase information.

1861 3.2: Functional Element updates the phase and the use case ends.

1862 4: Delete Phase.

1863 4.1: User provides phase name, lifecycle name and lifecycle namespace

1864 4.2: Functional Element deletes phase and the use case ends.

#### 1865 **2.6.7.2.2 Alternative Flows**

1866 1: Phase Does Not Exist.

1867 1.1: In basic flow 2.1, 3.1 and 4.1 if phase cannot be found based on phase name, lifecycle  
1868 name and lifecycle namespace provided by user, Functional Element returns an error  
1869 message and the use case ends.

1870 2: Creation of phase fails.

1871 2.1: In basic flow 1.2, if phase cannot be created, the Functional Element returns an error  
1872 message and the use case ends

#### 1873 **2.6.7.2.3 Special Requirements**

1874 None.

#### 1875 **2.6.7.2.4 Pre-Conditions**

1876 None.

#### 1877 **2.6.7.2.5 Post-Conditions**

1878 None.

### 1879 **2.6.7.3 Manage Relationship**

#### 1880 **2.6.7.3.1 Description**

1881 This use case describes the management of the relationship between user/group and phase in a  
1882 lifecycle.

#### 1883 **2.6.7.3.2 Flow of Events**

##### 1884 **2.6.7.3.2.1 Basic Flow**

1885 This use case starts when the user wants to manage the relationship between the user/group and  
1886 phase.

1887 

- If user refers to 'Create Relationship', basic flow 1 is executed.

- 1888       • If user refers to '**Update Relationship**', basic flow 2 is executed.
- 1889       • If user wants to '**Retrieve Relationship**', basic flow 3 is executed.
- 1890       • If user refers to '**Delete Relationship**', basic flow 4 is executed.
- 1891       1: Create Relationship.
- 1892           1.1: User provides user/group, phase and phase information.
- 1893           1.2: Functional Element creates relationship and the use case ends.
- 1894       2: Update Relationship.
- 1895           2.1: User provides user/group name and user/group namespace.
- 1896           2.2: Functional Element updates the relationship and the use case ends.
- 1897       3: Retrieve Relationship.
- 1898           3.1: User provides user/group name and user/group namespace.
- 1899           3.2: Functional Element returns the relationship and the use case ends.
- 1900       4: Delete Relationship.
- 1901           4.1: User provides user/group name and user/group namespace.
- 1902           4.2: Functional Element deletes relationship between phases and users/groups and the use  
1903           case ends.

1904       **2.6.7.3.2.2 Alternative Flows**

- 1905       1: Phase Does Not Exist.
- 1906           1.1: In basic flow 1,2, 2.2, 3.2 and 4.2, if the phase does not exist, the Functional Element  
1907           returns an error message and the use case ends.
- 1908       2: User/Group Does Not Exist.
- 1909           1.1: In basic flow 1,2, 2.2, 3.2 and 4.2, if the user/group does not exist, the Functional  
1910           Element returns an error message and the use case ends.

1911 **2.6.7.3.3 Special Requirements**

1912 None.

1913 **2.6.7.3.4 Pre-Conditions**

1914 None.

1915 **2.6.7.3.5 Post-Conditions**

1916 None.

1917

## 1918 **2.7 Presentation Transformer Functional Element**

### 1919 **2.7.1 Motivation**

1920 In a Web Service implementation, there exists the need to render the eventual presentation  
1921 layout to different consumers depending on their receiving capabilities. As such, there is a need  
1922 to dynamically generate the appropriate output at runtime.

1923

1924 This Functional Element fulfills the following requirements from the Functional Elements  
1925 Requirements, Working Draft 01a:

- 1926 • Primary Requirements
  - 1927 • DELIVERY-001, and
  - 1928 • DELIVERY-005 to DELIVERY-007.
- 1929 • Secondary Requirements
  - 1930 • None

### 1931 **2.7.2 Terms Used**

Terms	Description
XSL	Extensible Stylesheet Language

### 1932 **2.7.3 Key Features**

1933 Implementations of the Presentation Transformer Functional Element are expected to provide the  
1934 following key features:

- 1935 1. The Functional Element MUST be able to transform an XML document into a required  
1936 presentation format (output).
- 1937 2. The Functional Element MUST be able to understand a XSL document for the specifications  
1938 of the required presentation format.

1939

### 1940 **2.7.4 Interdependencies**

1941 None

### 1942 **2.7.5 Related Technologies and Standards**

1943 None

1944

1945 **2.7.6 Model**

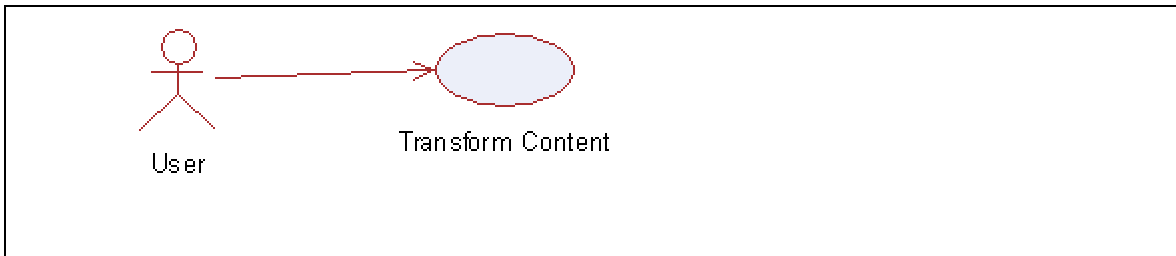


Figure 8: Model Of the Presentation Functional Element [15]

1946

1947 **2.7.7 Usage Scenario**

1948 **2.7.7.1 Transform Content**

1949 **2.7.7.1.1 Description**

1950 This use case allows the service user to transform the content to the appropriate mark-up  
1951 language.

1952 **2.7.7.1.2 Flow of Events**

1953 **2.7.7.1.2.1 Basic Flow**

1954 This use case starts when the service user wishes to transform the content to the appropriate  
1955 mark-up language.

1956 1: The Functional Element receives the request from the service user to transform the content to  
1957 the appropriate mark-up language.

1958 2: The Functional Element detects the type of mark-up language to be transformed.

1959 3: The Functional Element extracts mark-up language type from either the XML document or  
1960 parameters.

1961 4: The Functional Element retrieves appropriate mark-up language style sheet and transform it  
1962 into appropriate mark-up language.

1963 5: The Functional Element returns transformed appropriate result and the use case ends.

1964 **2.7.7.1.2.2 Alternative Flows**

1965 1: Unsupported Content.

1966 1.1: If in basic flow 2, the Functional Element is unable to detect the content type, the  
1967 Functional Element returns an error message and the use case ends.

1968 2: Unsupported Mark-up Language.

1969 2.1: If in basic flow 2, the Functional Element is unable to detect the supported mark-up  
1970 language type, the Functional Element returns an error message and the use case ends.

1971 **2.7.7.1.3 Pre-Conditions**

1972 None.

1973 **2.7.7.1.4 Post-Conditions**

1974 None.

1975

1976 **2.8 Role and Access Management Functional Element**

1977 **2.8.1 Motivation**

1978 The Role and Access Management Functional Element is expected to be an integral part of the  
1979 User Access Management (UAM) functionalities that is expected to be needed by a Web Service-  
1980 enabled implementation. This Functional Element is expected to fulfill the needs arising out of  
1981 managing access to resources within an application, based on role-based access control  
1982 mechanism. As such it will cover aspects that include:

- 1983       • Management of roles and access privileges, and
- 1984       • Assignment of roles to entities that will be accessing the resources that is being  
1985 managed.

1986

1987 This Functional Element fulfills the following requirements from the Functional Elements  
1988 Requirements, Working Draft 01a:

- 1989       • Primary Requirements
- 1990               • MANAGEMENT-030 to MANAGEMENT-034, and
- 1991               • MANAGEMENT-200 to MANAGEMENT-205.
- 1992       • Secondary Requirements
- 1993               • SECURITY-040 to SECURITY-041.

1994

1995 **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"> <li>• Determine who (user) is requesting access.</li> <li>• Determine the role(s) of the user</li> <li>• Determine the type of access that is allowed based on the role(s) of the user</li> </ul> <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"> <li>• Defining a set of basic user information that should be stored in any enterprise application.</li> <li>• Providing a means to extend this basic set of user information when needed..</li> <li>• Simplifying management by grouping related users together through certain criteria.</li> <li>• Having the flexibility of adopting both coarse/fine grain access controls.</li> </ul>

1996

1997 **2.8.3 Key Features**

1998 Implementations of the Secure SOAP Functional Element are expected to provide the following  
1999 key features:

- 2000 1. The Functional Element MUST provide the capability to manage the creation and deletion of  
2001 instances of the following concepts based on a pre-defined structure:
- 2002 1.1. Role,  
2003 1.2. Access, and  
2004 1.3. Resource
- 2005 2. The Functional Element MUST provide the capability to manage all the information (attribute  
2006 values) stored in such concepts. This includes the capability to retrieve and update  
2007 attribute's values belonging to a concept like Role, Access or Resource.
- 2008 3. The Functional Element MUST provide the capability to associate a Role to its access  
2009 privileges through the Access structure.
- 2010 4. The Functional Element MUST provide the capability to determine a Role's accessibility to  
2011 Resources based on the access privileges that have been assigned.

- 2012 5. The Functional Element MUST provide the ability to manage the association of users to  
2013 Roles via assignments of Roles to users. This will include:
- 2014 5.1. Assignment/Un-assignment of Roles to individual Users, and
- 2015 5.2. Assignment/Un-assignment of Roles to Groups.
- 2016 This will provide an indirect linkage between the accessibility of specific Users to Resources  
2017 through the concept of Role and Access.
- 2018 6. The Functional Element MUST provide a mechanism for managing the concepts of Role,  
2019 Access and Resource across different application domains.
- 2020 *Example: Namespace control mechanism*

2021

2022 In addition, the following key features could be provided to enhance the Functional Element  
2023 further:

- 2024 1. The Functional Element MAY provide a mechanism to enable different Access instances to  
2025 be related to one another.
- 2026 2. The Functional Element MAY also provide a mechanism to enable hierarchical  
2027 relationships between Access instances.
- 2028 *Example: Parent and Child Relationship*
- 2029 3. The Functional Element MAY provide the ability for Roles to be temporal sensitive.
- 2030 *Example: A Role is assigned to a particular Phase in a Lifecycle.*

2031

## 2032 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.

## 2033 2.8.5 Related Technologies and Standards

2034 None

2035

2036 **2.8.6 Model**

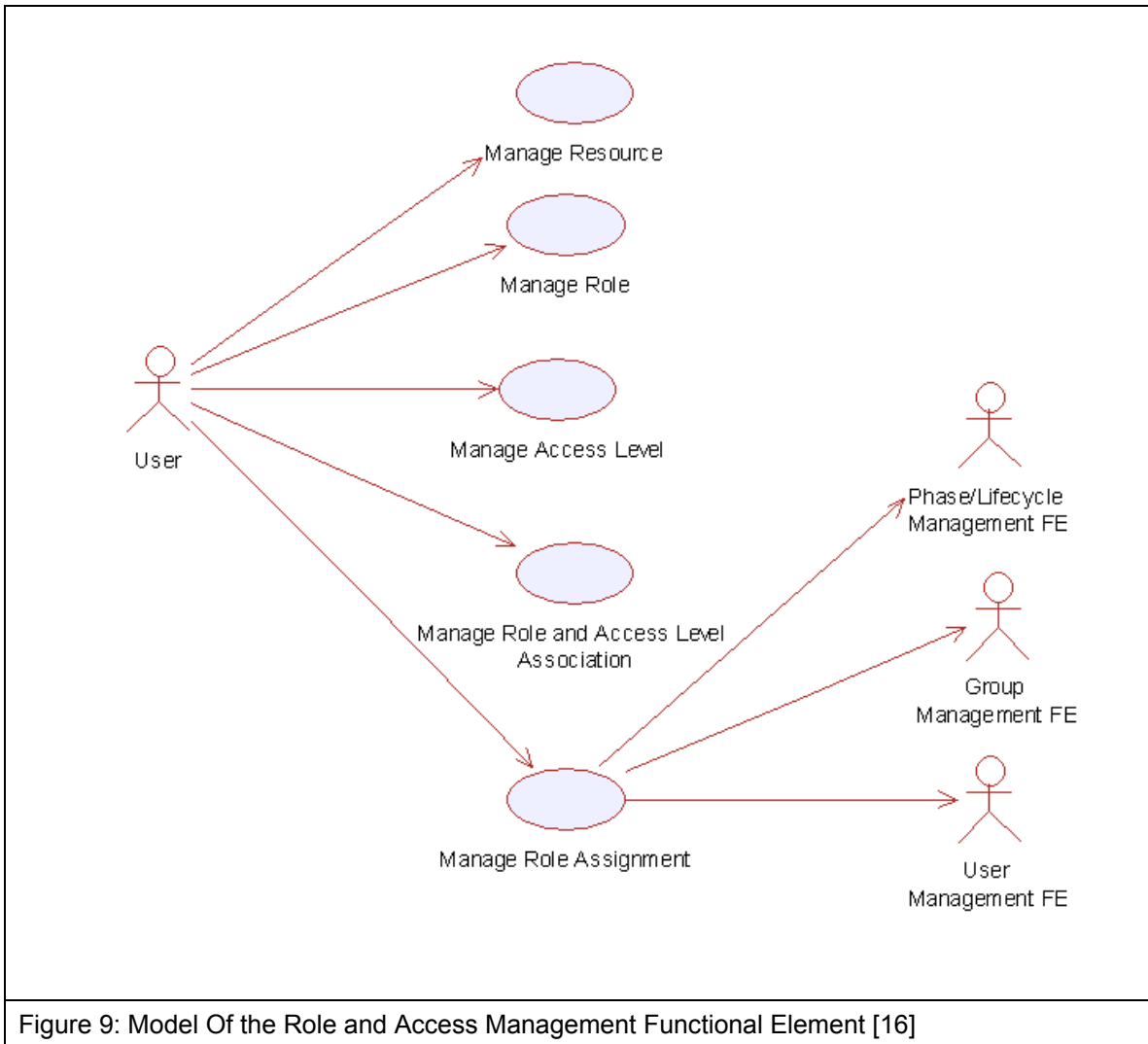


Figure 9: Model Of the Role and Access Management Functional Element [16]

2037

2038 **2.8.7 Usage Scenario**

2039 **2.8.7.1 Manage Role**

2040 **2.8.7.1.1 Description**

2041 This use case allows the service user to manipulate the role information such as adding,  
2042 changing and deleting role information in the Functional Element.

2043 **2.8.7.1.2 Flow of Events**

2044 **2.8.7.1.2.1 Basic Flow**

2045 This use case starts when any user wants to create, change or delete a role.

2046 1: Service user specifies the function it would like to perform (either create a role, update a role or  
2047 delete a role).

2048 2: Once the service user provides the requested information, one of the sub-flows is executed.

- 2049
- If the service user provides '**Create a Role**', then sub-flow 2.1 is executed.
  - 2050 • If the service user provides '**Retrieve a Role**', then sub-flow 2.2 is executed.
  - 2051 • If the service user provides '**Update a Role**', then sub-flow 2.3 is executed.
  - 2052 • If the service user provides '**Delete a Role**', then sub-flow 2.4 is executed.

2053 2.1: Create a Role.

2054 2.1.1: The service user specifies role information such as the role name and description.

2055 2.1.2: The Functional Element connects to the data storage.

2056 2.1.3: The Functional Element checks whether the role exists in the Functional Element  
2057 or not, saves the role information in the data storage and the use case ends.

2058 2.2: Retrieve a Role.

2059 2.2.1: The service user specifies the role name for retrieval.

2060 2.2.2: The Functional Element connects to the data storage.

2061 2.2.3: The Functional Element retrieves the role information in the data storage and the  
2062 use case ends.

2063 2.3: Update a Role.

2064 2.3.1: The service user specifies the role name to update.

2065 2.3.2: The service user specifies the target field name and value of the role.

2066 2.3.3: The Functional Element connects to the data storage.

2067 2.3.4: The Functional Element updates the role information in the data storage and the  
2068 use case ends.

2069 2.4: Delete a Role.

2070 2.4.1: The service user specifies the role name to delete.

2071 2.4.2: The Functional Element connects to the data storage.

2072 2.4.3: The Functional Element removes the record of the role in the data storage and the  
2073 use case ends.

2074 **2.8.7.1.2.2 Alternative Flows**

2075 1: Data Storage Not Available.

2076 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  
2077 available, an error message is returned and the use case ends.

2078 2: Role Already Exists.

2079 2.1: If in basic flow 2.1.3, the Functional Element checks that the role already exists in the  
2080 data storage, an error message is returned and the use case ends.

2081 3: Role Does Not Exist.

2082 3.1: If in basic flow 2.2.3, 2.3.4 and 2.4.3, the Functional Element checks that the role does  
2083 not exist in the data storage, an error message is returned and the use case ends.

2084 4: Role Cannot Be Deleted.

2085 4.1: If in basic flow 2.4.3, the other information associated with the role, such as any access  
2086 level assigned, still exists, the role information may not be removed. An error message is  
2087 returned and the use case ends.

2088 **2.8.7.1.3 Special Requirements**

2089 None

2090 **2.8.7.1.4 Pre-Conditions**

2091 None.

2092 **2.8.7.1.5 Post-Conditions**

2093 If the use case was successful, the role is saved/updated/removed in the Functional Element.  
2094 Otherwise, the Functional Element state is unchanged.

## 2095 **2.8.7.2 Manage Resource**

### 2096 **2.8.7.2.1 Description**

2097 This use case allows the service user to manipulate the resource information such as adding,  
2098 changing and deleting resource information in the Functional Element.

### 2099 **2.8.7.2.2 Flow of Events**

#### 2100 **2.8.7.2.2.1 Basic Flow**

2101 This use case starts when any user wants to create, change or delete a resource.

2102 1: The user specifies the function it would like to perform.

2103 2: The user provides the requested information, one of the sub-flows is executed.

2104 • If the user provides '**Create a Resource**', then sub-flow 2.1 is executed.

2105 • If the user provides '**Retrieve a Resource**', then sub-flow 2.2 is executed.

2106 • If the user provides '**Update a Resource**', then sub-flow 2.3 is executed.

2107 • If the user provides '**Delete a Resource**', then sub-flow 2.4 is executed.

2108 2.1: Create a Resource.

2109 2.1.1: The user specifies resource information such as the resource name and  
2110 description.

2111 2.1.2: The Functional Element connects to the data storage.

2112 2.1.3: The Functional Element checks whether the resource exists in the Functional  
2113 Element, saves the resource information in the data storage and the use case ends.

2114 2.2: Retrieve a Resource.

2115 2.2.1: The service user specifies the resource name for retrieval.

2116 2.2.2: The Functional Element connects to the data storage.

2117 2.2.3: The Functional Element retrieves the resource information in the data storage and  
2118 the use case ends.

2119 2.3: Update a Resource.

- 2120 2.3.1: The service user specifies the resource name to update.
- 2121 2.3.2: The Functional Element connects to the data storage.
- 2122 2.3.3: The Functional Element updates the resource information in the data storage and  
2123 the use case ends.
- 2124 2.4: Delete a Resource.
- 2125 2.4.1: The service user specifies the resource name to delete.
- 2126 2.4.2: The Functional Element connects to the data storage.
- 2127 2.4.3: The Functional Element removes the record of the resource in the data storage  
2128 and the use case ends.
- 2129 **2.8.7.2.2 Alternative Flows**
- 2130 1: Data Storage Not Available.
- 2131 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  
2132 is not available, an error message is returned and the use case ends.
- 2133 2: Resource Already Exists.
- 2134 2.1: If in basic flow 2.1.3, the Functional Element checks that the resource already exists in  
2135 the data storage, an error message is returned and the use case ends.
- 2136 3: Resource Does Not Exist.
- 2137 3.1: If in basic flow 2.2.3, 2.3.3 and 2.4.3, the Functional Element checks that the resource  
2138 does not exist in the data storage, an error message is returned and the use case ends.
- 2139 **2.8.7.2.3 Special Requirements**
- 2140 None
- 2141 **2.8.7.2.4 Pre-Conditions**
- 2142 None.
- 2143 **2.8.7.2.5 Post-Conditions**
- 2144 None

## 2145 **2.8.7.3 Manage Access Level**

### 2146 **2.8.7.3.1 Description**

2147 This use case allows service user to manage the creation/retrieval/modification/deletion of access  
2148 level.

### 2149 **2.8.7.3.2 Flow of Events**

#### 2150 **2.8.7.3.2.1 Basic Flow**

2151 This use case starts when service user wants to manage the access levels.

2152 1: The service user specifies the function it would like to perform (add, update or delete an  
2153 access level).

2154 2: Once the service user provides the requested information, one of the sub-flows is executed.

- 2155 • If the service user provides '**Add an Access Level**', then sub-flow 2.1 is executed.
- 2156 • If the service user provides '**Retrieve an Access Level**', then sub-flow 2.2 is activated.
- 2157 • If the service user provides '**Update an Access Level**', then sub-flow 2.3 is activated.
- 2158 • If the service user provides '**Delete an Access Level**', then sub-flow 2.4 is executed.

2159 2.1: Add an Access Level.

2160 2.1.1: The service user specifies the access level information, which includes: name,  
2161 description, name of parent access level and group of resources that the access level is  
2162 associated with.

2163 2.1.2: The Functional Element connects to the data storage.

2164 2.1.3: The Functional Element check whether the access level and its parent access level  
2165 exist in the Functional Element, saves the access level information in the data storage  
2166 and the use case ends.

2167 2.2: Retrieve an Access Level.

2168 2.2.1: The service user specifies the access level name to retrieve.

2169 2.2.2: The Functional Element connects to the data storage.

2170 2.2.3: The Functional Element gets access level information from the data storage and  
2171 returns to the service user and the use case ends.

- 2172 2.3: Update an Access Level.
- 2173 2.3.1: The service user specifies the access level name.
- 2174 2.3.2: The service user specifies the field(s) and new value(s) to update.
- 2175 2.3.3: The Functional Element connects to the data storage.
- 2176 2.3.4: The Functional Element updates the access level information in the data storage  
2177 with the value specified in 2.3.2 and the use case ends.
- 2178 2.4: Delete an Access Level.
- 2179 2.4.1: The service user specifies the access level name to delete.
- 2180 2.4.2: The Functional Element connects to the data storage.
- 2181 2.4.3: The Functional Element removes the record of the access level in the data storage  
2182 and the use case ends.
- 2183 **2.8.7.3.2 Alternative Flows**
- 2184 1: Data Storage Not Available.
- 2185 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  
2186 information is not available, an error message is returned and the use case ends.
- 2187 2: Access Level Already Exists.
- 2188 2.1: If in basic flow 2.1.3, the Functional Element checks that the access level already exists  
2189 in the data storage, an error message is returned and the use case ends.
- 2190 3: Access Level Cannot Be Deleted.
- 2191 3.1: If in basic flow 2.4.3, the other information associated with the Access Level, such as  
2192 roles to which the access level is assigned and the parent access level still exists, the access  
2193 level information may not be removed. An error message is returned and the use case ends.
- 2194 4: Parent Access Level Not Exist.
- 2195 4.1: If in basic flow 2.1.3, the parent access level does not exist, an error message is returned  
2196 and the use case ends.

2197 **2.8.7.3.3 Special Requirements**

2198 None

2199 **2.8.7.3.4 Pre-Conditions**

2200 None.

2201 **2.8.7.3.5 Post-Conditions**

2202 None

2203 **2.8.7.4 Manage Role and Access Level Association**

2204 **2.8.7.4.1 Description**

2205 This use case allows service user to assign, update and remove the access level assigned to  
2206 role.

2207 **2.8.7.4.2 Flow of Events**

2208 **2.8.7.4.2.1 Basic Flow**

2209 This use case starts when service user wants to manage the relationship between access level  
2210 and role.

2211 1: The service user specifies a role and the function he/she would like to perform on the role  
2212 (either assign an access level to role, update role access level, or delete role access level).

2213 2: Once the service user provides the requested information, one of the sub-flows is executed.

2214 • If the user provides '**Assign an Access Level to Role**', then sub-flow 2.1 is executed.

2215 • If the user provides '**Update Access Level for Role**', then sub-flow 2.2 is executed.

2216 • If the user provides '**Delete Access Level for Role**', then sub-flow 2.3 is executed.

2217 • If the user provides '**Retrieve Access Level for Role**', then sub-flow 2.4 is executed.

2218 • If the service user provides '**Retrieve Role for Access Level**', then sub-flow 2.5 is  
2219 executed.

2220 2.1: Assign an Access Level to Role.

2221 2.1.1: The service user specifies access level that will be assigned to the role.

2222 2.1.2: The Functional Element connects to the data storage.

2223 2.1.3: The Functional Element checks whether the access level has been assigned to the  
2224 role. Functional Element saves the access level reference in the role record in the data  
2225 storage and the use case ends.

2226 2.2: Update Access Level for Role.

2227 2.2.1: The service user specifies the access level to update and the new access level  
2228 information.

2229 2.2.2: The Functional Element connects to the data storage.

2230 2.2.3: The Functional Element updates the access level reference in the role record in the  
2231 data storage and the use case ends.

2232 2.3: Delete Access Level to Role.

2233 2.3.1: The service user specifies the access level to delete.

2234 2.3.2: The Functional Element connects to the data storage.

2235 2.3.3: The Functional Element removes the access level reference from the record of the  
2236 role in the data storage and the use case ends.

2237 2.4: Retrieve Access Level for Role.

2238 2.4.1: The service user specifies the role to retrieve the access levels associated with it.

2239 2.4.2: The Functional Element connects to the data storage.

2240 2.4.3: The Functional Element retrieves the access level assigned to the role in the data  
2241 storage and the use case ends.

2242 2.5: Retrieve Role for Access Level.

2243 2.5.1: The service user specifies the access level to retrieve roles associated to it.

2244 2.5.2: The Functional Element connects to the data storage.

2245 2.5.3: The Functional Element retrieves roles associated to the access level in the data  
2246 storage and the use case ends.

#### 2247 **2.8.7.4.2.2 Alternative Flows**

2248 1: Data Storage Not Available.

2249 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  
2250 not available, an error message is returned and the use case ends.

2251 2: Access Level Assignment Already Exists.

2252 2.1: If in basic flow 2.1.3, the Functional Element checks that the access level already exists  
2253 in the role record in the data storage, an error message is returned and the use case ends.

2254 3: Access Level Assignment Not Exist.

2255 3.1: If in basic flow 2.3.3, the access level assignment does not exist, an error message is  
2256 returned and the use case ends.

2257 4: Access Level Not Exist.

2258 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  
2259 error message is returned and the use case ends.

2260 5: Role Not Exist.

2261 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  
2262 message is returned and the use case ends.

2263 **2.8.7.4.3 Special Requirements**

2264 None.

2265 **2.8.7.4.4 Pre-Conditions**

2266 None.

2267 **2.8.7.4.5 Post-Conditions**

2268 None.

2269 **2.8.7.5 Manage Role Assignment**

2270 **2.8.7.5.1 Description**

2271 The use case allows service user to assign a role to a user, a group, a phase in a lifecycle, to  
2272 change or to delete such assignment.

2273 **2.8.7.5.2 Flow of Events**

2274 **2.8.7.5.2.1 Basic Flow**

2275 This use case starts when the service user wants to manage the assignment of a role. This role  
2276 can be assigned to a user, group, phase and lifecycle.

2277 1: Service user specifies a role and an operation to perform on the role.

2278 2: Once the service user provides the requested information, one of the sub-flows is executed.

2279 • If the user provides '**Assign Role**', then sub-flow 2.1 is executed.

2280 • If the user provides '**Retrieve Role**', then sub-flow 2.2 is executed.

2281 • If the user provides '**Un-assign Role**', then user sub-flow 2.3 is executed.

2282 2.1: Assign Role.

2283 2.1.1: The service user specifies a user/group/phase/lifecycle to which the role will be  
2284 assigned.

2285 2.1.2: Depending of target of the assignment, the Functional Element will check for the  
2286 presence of one of the following Functional Elements.

2287 • User Management Functional Element

2288 • Group Management Functional Element

2289 • Phase and Lifecycle Management Functional Element

2290 2.1.3: The Functional Element checks whether the role has been assigned to the  
2291 intended target

2292 2.1.4: The Functional Element saves the relationship between the role and the target and  
2293 the use case ends.

2294 2.2: Retrieve Role.

2295 2.2.1: The service user specifies a user/group/phase/lifecycle to retrieve all roles  
2296 assigned

2297 2.2.2: Depending of target of the assignment, the Functional Element will check for the  
2298 presence of one of the following Functional Elements.

2299 • User Management Functional Element

2300 • Group Management Functional Element

- 2301                   • Phase and Lifecycle Management Functional Element
- 2302                   2.2.3: The Functional Element gets the roles that are assigned to the target.
- 2303                   2.2.4: The Functional Element returns the results to the service user and the use case  
2304                   ends.
- 2305                   2.3: Un-assign Role.
- 2306                   2.3.1: The service user specifies a user/group/phase/lifecycle and the role that is to be  
2307                   un-assigned.
- 2308                   2.3.2: Depending of target of this un-assignment, the Functional Element will check for  
2309                   the presence of one of the following Functional Elements.
- 2310                   • User Management Functional Element
- 2311                   • Group Management Functional Element
- 2312                   • Phase and Lifecycle Management Functional Element
- 2313                   2.3.3: The Functional Element checks if the roles have been assigned to the target in the  
2314                   first place.
- 2315                   2.3.4: The Functional Element removes the role assigned and the use case ends.
- 2316                   **2.8.7.5.2.2 Alternative Flows**
- 2317                   1: Dependent Functional Element not available.
- 2318                   1.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the dependent Functional Elements are not  
2319                   available, an error message is returned and the use case ends.
- 2320                   2: Invalid User/Group/Phase/Lifecycle Account.
- 2321                   2.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the dependent Functional Elements are available  
2322                   but an invalid account is provided, an error message is returned and the use case ends.
- 2323                   3: Data Storage Not Available.
- 2324                   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  
2325                   storage, an error message is provided and the use case ends.
- 2326

2327 **2.8.7.5.3 Special Requirements**

2328 None.

2329 **2.8.7.5.4 Pre-Conditions**

2330 None.

2331 **2.8.7.5.5 Post-Conditions**

2332 None.

2333 **2.9 Search Functional Element**

2334 **2.9.1 Motivation**

2335 In a Web Service-enabled implementation, information is distributed across different sites and this  
2336 makes searching and collating information difficult. Against this backdrop, this Functional  
2337 Element is expected to fulfill the needs identified within an application by covering the following  
2338 aspects.

- 2339 • Providing the capability for configuration of different types of data sources for information  
2340 search,
- 2341 • Providing the facility to provide a concrete definition of data source classification for  
2342 information search,
- 2343 • Providing the ability to define different search scopes for various data source  
2344 classification,
- 2345 • Performing information search on those pre-configured different types of data sources  
2346 and
- 2347 • Providing the provision to consolidate the return result arising from the search operation.

2348

2349 This Functional Element fulfills the following requirements from the Functional Elements  
2350 Requirements, Working Draft 01a:

- 2351 • Primary Requirements
  - 2352 • MANAGEMENT-009,
  - 2353 • PROCESS-030 to PROCESS-031, and
  - 2354 • PROCESS-034.
- 2355 • Secondary Requirements
  - 2356 • None

2357

2358 **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

### 2359 **2.9.3 Key Features**

2360 Implementations of the Search Functional Element are expected to provide the following key  
2361 features:

- 2362 1. The Functional Element MUST provide a mechanism to define and manage Search  
2363 Categories.
- 2364 2. The Functional Element MUST provide the capability to configure and store information  
2365 about targeted data sources for a particular Search Category.  
*Example: Some of the stored information would include Location, Type, Name, Data Fields (of interest to the search) and access control (typically username and password) of the targeted data source.*
- 2366 3. As part of Key Feature (2), the Functional Element MUST also provide the ability to  
2367 configure the scope of search and returned results.  
2368
- 2369 4. The Functional Element MUST also provide a mechanism to link the Search Categories to  
2370 configured target data sources.
- 2371 5. The Functional Element MUST provide the ability to search multiple data sources for a  
2372 defined Search Category.  
*Example: Some of the common data sources would include RDBMS, XML DB, LDAP servers and flat files like XML files, text files and HTML files*
- 2373 6. The Functional Element MUST provide the ability to perform searches based on a given set  
2374 of keyword(s).  
2375  
2376  
2377  
2378

2379

2380 In addition, the following key features could be provided to enhance the Functional Element  
2381 further:

- 2382 1. The Functional Element MAY also provide the ability to perform conditional and parametric  
2383 searches.
- 2384 2. The Functional Element MAY also provide the ability to restrict the scope of a search.  
*Example: By providing a particular Search Category or types of data sources for the search.*

2385  
2386  
2387

2388 **2.9.4 Interdependencies**

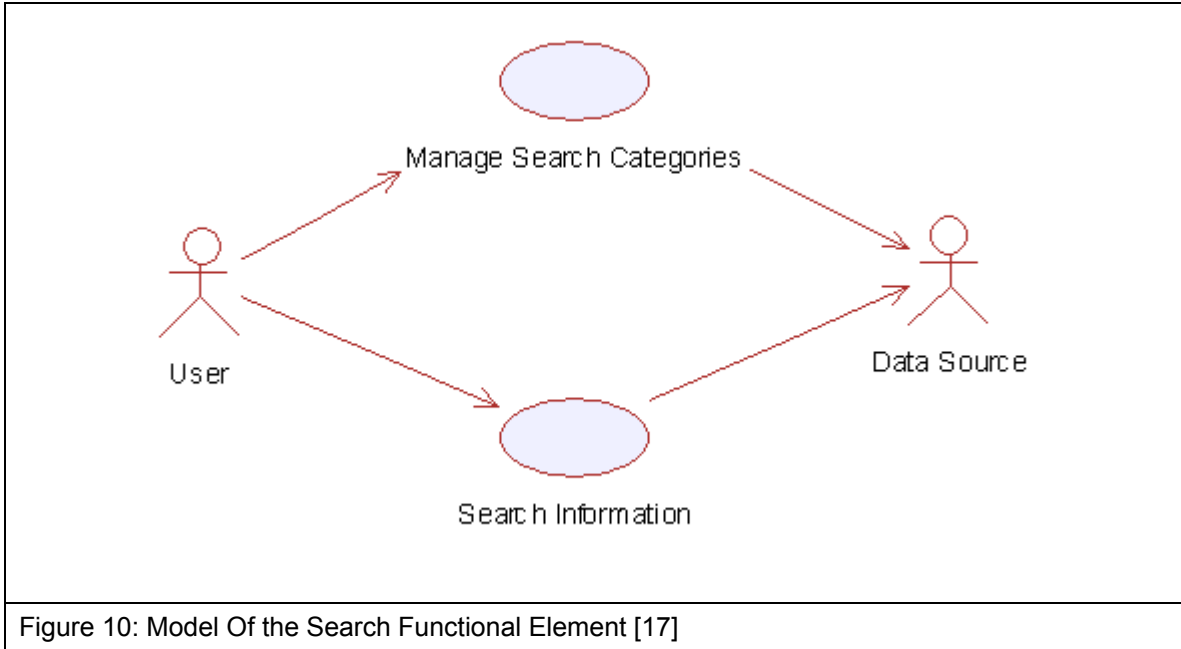
2389 None

2390

2391 **2.9.5 Related Technologies and Standards**

2392 None

2393 **2.9.6 Model**



2394 **2.9.7 Usage Scenario**

2395 **2.9.7.1 Manage Search Categories**

2396 **2.9.7.1.1 Description**

2397 This use case allows the users to manage the different search categories.

2398 **2.9.7.1.2 Flow of Events**

2399 **2.9.7.1.2.1 Basic Flow**

2400 This use case starts when the user wishes to manage the different data sources for search to be  
2401 performed on it.

2402 1: The users initiates a request to configure data source(s) and type(s) by providing the data  
2403 source information and type to be added, removed or retrieved.

- 2404 2: The Functional Element checks whether the data source configuration file exists.
- 2405 3: The Functional Element checks the request. Based on the type of request, one of the sub-  
2406 flows is executed.
- 2407 • If the request is to 'Create Data Source And Type', then sub-flow 3.1 is executed.
- 2408 • If the request is to 'View Data Sources And Types', then sub-flow 3.2 is executed.
- 2409 • If the request is to 'Delete Data Source And Type', then sub-flow 3.3 is executed.
- 2410 3.1: Create Data Source and Type.
- 2411 3.1.1: The Functional Element checks whether the same data source and type has been  
2412 created.
- 2413 3.1.2: The Functional Element appends the new data source and type in the data source  
2414 configuration file specified.
- 2415 3.2: View Data Source and Type.
- 2416 3.2.1: The Functional Element retrieves all the data source and type information from the  
2417 data source configuration file.
- 2418 3.2.2: The Functional Element returns the data source(s) and type(s).
- 2419 3.3: Delete Data Source and Type.
- 2420 3.3.1: The Functional Element checks whether the data source and type exist in the data  
2421 source configuration based on data source id from the data source configuration file.
- 2422 3.3.2: The Functional Element removes the old data source and type from the data  
2423 source configuration file.
- 2424 4: The Functional Element returns a success or failure flag indicating the status of the operation  
2425 being performed and use case ends.
- 2426 **2.9.7.1.2.2 Alternative Flows**
- 2427 1: Data Source Configuration File Not Found.
- 2428 1.1: If in basic flow 2, the data source configuration file does not exist, the Functional Element  
2429 creates an empty data source configuration file.
- 2430 2: Duplicate Data Source and Type.

- 2431 2.1: If in basic flow 3.1.1, the same data source and type have been configured, the  
2432 Functional Element returns an error message and the use case end.
- 2433 3: Data Source and Type Do Not Exist.
- 2434 3.1: If in basic flow 3.2.1 and 3.3.1, a particular data source and type cannot be found in the  
2435 specified data source configuration file, the Functional Element returns an error message and  
2436 the use case end.
- 2437 **2.9.7.1.3 Special Requirements**
- 2438 None.
- 2439 **2.9.7.1.4 Pre-Conditions**
- 2440 None.
- 2441 **2.9.7.1.5 Post-Conditions**
- 2442 None.
- 2443 **2.9.7.2 Search Information**
- 2444 **2.9.7.2.1 Description**
- 2445 This use case allows any users to perform search on various disparate data sources and types  
2446 configured to be searched and returns the matching results.
- 2447 **2.9.7.2.2 Flow of Events**
- 2448 **2.9.7.2.2.1 Basic Flow**
- 2449 This use case starts when users wishes to perform information search on a data source.
- 2450 1: Users initiates a request to perform information search on a given data source by providing  
2451 information to be searched, location of the data source(s) and the data source type(s).
- 2452 2: The Functional Element checks for the existence of the specified data source(s).
- 2453 3: The Functional Element validates the data source type(s) against the set of supported data  
2454 type(s) configured within the Functional Element that are available for information search.
- 2455 4: The Functional Element performs information search based on the search parameters given by  
2456 the users or the other Functional Elements.
- 2457 5: The Functional Element returns the result of the information search performed to the users or  
2458 other Functional Elements and use case ends.

2459 **2.9.7.2.2 Alternative Flows**

2460 1: Data Source(s) Are Not Available.

2461 1.1: In basic flow 2, if the identified data source is not available, the Functional Element  
2462 returns an error message and the use case ends.

2463 2: Invalid Configuration Instructions

2464 2.1: In basic flow 2, if the input inform by the user is incomplete, the Functional Element  
2465 returns an error message and the use case ends.

2466 3: Invalid Data Source Type.

2467 3.1: In basic flow 3, if the data source type is invalid, the Functional Element returns an error  
2468 message and the use case ends.

2469 4: No Matching Result.

2470 4.1: In basic flow 4, if the search results in no matching results, the Functional Element  
2471 returns an error message and the use case ends..

2472 **2.9.7.2.3 Special Requirements**

2473 None

2474 **2.9.7.2.4 Pre-Conditions**

2475 None.

2476 **2.9.7.2.5 Post-Conditions**

2477 None.

2478

## 2479 **2.10 Secure SOAP Management Functional Element**

### 2480 **2.10.1 Motivation**

2481 In a Web Services implementation, it is envisaged that confidential information is being exchanged  
2482 all the time. Against this backdrop, it is imperative that an application in such an environment is  
2483 equipped with the capability to guard sensitive information from prying eyes. Secure SOAP  
2484 Management fulfills this need by covering the following areas.

- 2485 • The facility of digitally signing SOAP message,
- 2486 • The facility of encrypting SOAP message, and
- 2487 • The capability to generate the original SOAP message after signing or encrypting the  
2488 message.

2489

2490 This Functional Element fulfills the following requirements from the Functional Elements  
2491 Requirements, Working Draft 01a:

- 2492 • Primary Requirements
  - 2493 • SECURITY-003 (SECURITY-003-3 only),
  - 2494 • SECURITY-020 (all), and
  - 2495 • SECURITY-022, and
  - 2496 • SECURITY-026.
- 2497 • Secondary Requirements
  - 2498 • None

2499

### 2500 **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.
---	---

2501

### 2502 **2.10.3 Key Features**

2503 Implementations of the Group Management Functional Element are expected to provide the  
2504 following key features:

- 2505 1. The Functional Element MUST provide the capability to digitally sign SOAP messages  
2506 completely or partially using XML-Signature Syntax and Processing, W3C Recommendation  
2507 12 February 2002.
- 2508 2. The Functional Element MUST provide the capability to validate a signed SOAP message.
- 2509 3. The Functional Element MUST provide the capability to encrypt SOAP messages  
2510 completely or partially using XML-Encryption Syntax and Processing, W3C  
2511 Recommendation 10 December 2002.
- 2512 4. The Functional Element MUST provide the capability to decrypt encrypted SOAP messages.
- 2513 5. The Functional Element MUST support PKCS12 compatible digital certificates.
- 2514 6. The Functional Element MUST be able to verify the validity and authenticity of digital  
2515 certificates used.

2516

2517 In addition, the following key features could be provided to enhance the Functional Element  
2518 further:

- 2519 1. The Functional Element MAY also support PKCS11 compatible tokens.
- 2520 2. The Functional Element MAY also provide log support as part of the audit trails for its  
2521 transaction records.

2522

### 2523 **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.
--------------------------------	--

### 2524 **2.10.5 Related Technologies and Standards**

<b>Standards / Specifications</b>	<b>Specific References</b>
-----------------------------------	----------------------------

Public Key Infrastructure (PKI)	<p>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</p> <p>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.</p>
XML-Signature Syntax and Processing, W3C Recommendation 12 <sup>th</sup> Feb 2002 [18]	<p>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.</p> <p>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.</p>
XML-Encryption Syntax and Processing, W3C Recommendation 10 <sup>th</sup> Dec 2002 [19]	<p>This specification addresses data privacy by defining a process for encrypting data and representing the result in XML document.</p> <p>In this Functional Element, the encryption and decryption of SOAP messages are done based on this specification.</p>

2525

2526

2527 **2.10.6 Model**

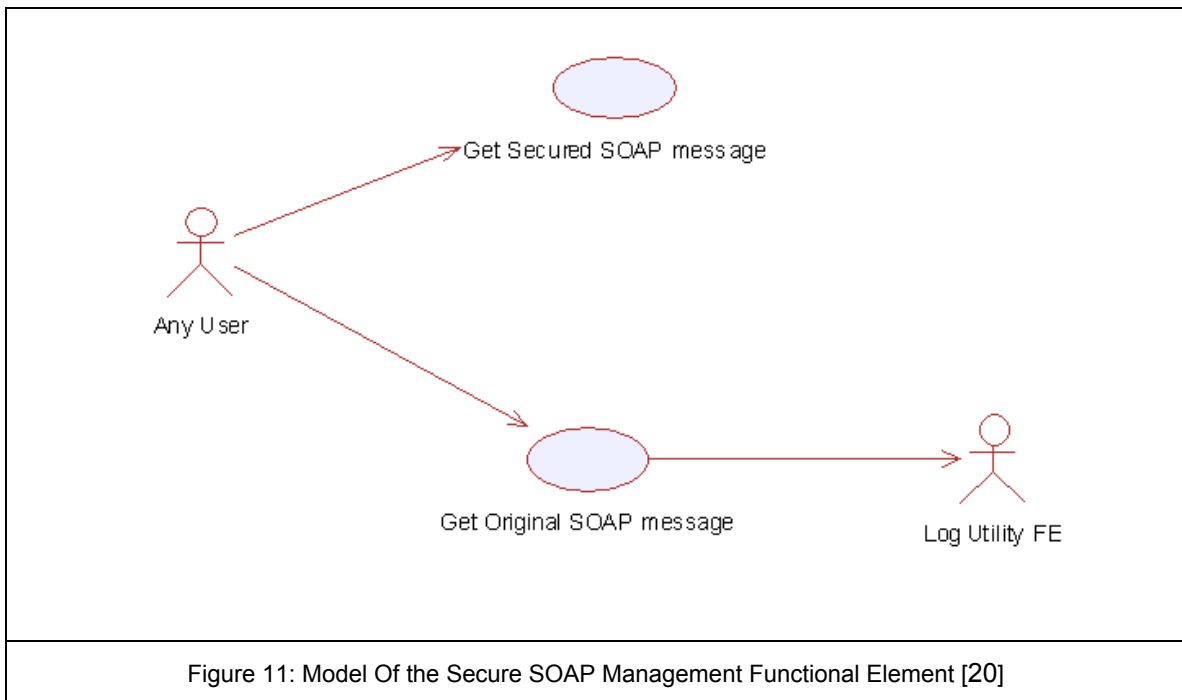


Figure 11: Model Of the Secure SOAP Management Functional Element [20]

2528 **2.10.7 Usage Scenarios**

2529 **2.10.7.1 Get Secured SOAP message**

2530 **2.10.7.1.1 Description**

2531 This Functional Element describes the process to generate secured SOAP message.

2532 **2.10.7.1.2 Flow of Events**

2533 **2.10.7.1.2.1 Basic Flow**

2534 This use case starts when the user wants to secure the SOAP message.

- 2535 • If user wants to '**Sign SOAP message**', then basic flow 1 is executed.
- 2536 • If user wants to '**Encrypt and Sign the SOAP message**', then basic flow 2 is executed.

2537 1: Sign SOAP Message.

2538 1.1: User sends the SOAP message, digital certificate and specifies the element name that  
2539 needs to be signed.

2540 1.2: Functional Element gets the key information from the digital certificate.

2541 *Note: The private key will be used to sign the SOAP message and the public key will be*  
2542 *added to the SOAP message after the signing.*

2543 1.3: Functional Element signs the element.

2544 *Note: The digital signature format is expected to be based on XML-Digital Signature Syntax*  
2545 *mentioned in section 3.10.5.*

2546 1.4: Functional Element parses the secure SOAP message and regenerates the SOAP  
2547 message.

2548 1.5: Functional Element returns the secured SOAP message to user and the use case ends.

2549 2: Encrypt And Sign SOAP Message.

2550 2.1: User sends the SOAP message, digital certificate and specify the element name that  
2551 needs to be encrypted.

2552 2.2: User sends the receiver's public key information to Functional Element.

2553 *Note: Receiver's public key will be used to encrypt the session key, which was then used to*  
2554 *encrypt the content of the element in the SOAP message.*

2555 2.3: Functional Element gets key information from the user's digital certificate.

2556 *Note: Private key is used to sign the SOAP message and public key is used to add into the*  
2557 *SOAP message after the signing.*

2558 2.4: Functional Element generates the session key.

2559 *Note: Session key is used to encrypt the content of the element.*

2560 2.5: Functional Element encrypts the content of element with the session key.

2561 2.6: Functional Element encrypts session key with the receiver's public key.

2562 2.7: Functional Element signs the SOAP message after encryption.

2563 2.8: Functional Element regenerates the SOAP message.

2564 *Note: Functional Element adds the encrypted content of the element, encrypted session key*  
2565 *information, the receiver's public key information and the signature to the SOAP message.*

2566 2.9: Functional Element returns the SOAP message and the use case ends.

#### 2567 **2.10.7.1.2.2 Alternative Flows**

2568 1: Cannot Get Key.

2569 1.1: In basic flow 1.2 and 2.3, Functional Element cannot get the key information from the  
2570 digital certificate. The Functional Element returns an error message and the use case ends.

2571 2: Cannot Sign

2572 2.1: In basic flow 1.3, Functional Element cannot sign the SOAP message. The Functional  
2573 Element returns an error message and the use case ends.

2574 3: Cannot Encrypt

2575 3.1: In basic flow 2.5, Functional Element cannot encrypt the SOAP message. The Functional  
2576 Element returns an error message and the use case ends.

#### 2577 **2.10.7.1.3 Special Requirements**

2578 None.

2579 **2.10.7.1.4 Pre-Conditions**

2580 None.

2581 **2.10.7.1.5 Post-Conditions**

2582 None.

2583 **2.10.7.2 Get Original SOAP Message**

2584 **2.10.7.2.1 Description**

2585 This use case allows users to get original SOAP message.

2586 **2.10.7.2.2 Flow of Events**

2587 **2.10.7.2.2.1 Basic Flow**

2588 This use case starts when the user wants to get the original SOAP message.

- 2589
- If the user wants to '**Verify the SOAP message**', then basic flow 1 is executed.
- 2590
- If the user wants to '**Decrypt and Verify the SOAP message**', then basic flow 2 is
- 2591
- executed.

2592 1: Verify SOAP Message.

2593 1.1: User sends the SOAP message and sender's digital certificate.

2594 1.2: Functional Element verifies the SOAP message.

2595 *Note: The sender's certificate information will be used to verify the signature.*

2596 1.3: Functional Element gets the original SOAP message, returns to user and the use case  
2597 ends.

2598 2: Decrypt And Verify The SOAP Message.

2599 2.1: User sends the SOAP message, user's digital certificate and sender's certificate.

2600 2.2: Functional Element verifies the SOAP message.

2601 *Note: The sender's certificate information will be used to verify the signature.*

2602 2.3: Functional Element gets the user's key information from the user's digital certificate.

- 2603 *Note: The user's private key will be used to decrypt the session key.*
- 2604 2.4: Functional Element decrypts the session key.
- 2605 2.5: Functional Element decrypts the content of the element with the session key.
- 2606 2.6: Functional Element regenerates the SOAP message.
- 2607 *Note: Functional Element removes the session key information and the digital signature*  
2608 *information from the SOAP message and gets the original one.*
- 2609 2.7: Functional Element returns the original SOAP message to user and the use case ends.
- 2610 **2.10.7.2.2 Alternative Flows**
- 2611 1: Verification Fails.
- 2612 1.1: In basic flow 1.3 and 2.3, if verification fails, the Functional Element returns an error  
2613 message and the use case ends.
- 2614 2: Decryption of Content Fails.
- 2615 2.1: In basic flow 2.5, the Functional Element cannot decrypt the content of the element. The  
2616 Functional Element returns an error message and the use case ends.
- 2617 **2.10.7.2.3 Special Requirements**
- 2618 None
- 2619 **2.10.7.2.4 Pre-Conditions**
- 2620 None.
- 2621 **2.10.7.2.5 Post-Conditions**
- 2622 None.

## 2623 2.11 Sensory Functional Element

### 2624 2.11.1 Motivation

2625 In a Web Service implementation where the presentation capabilities of clients differ, there is a  
2626 need to determine the exact ability of the end devices so that the appropriate contents may be  
2627 forwarded. The Sensory Functional Element can help to play this role by covering the following  
2628 aspects within an application:

- 2629 • Determining the presentation capabilities by inspecting incoming headers, and
- 2630 • Determining the presentation capabilities by extracting MIME information from the  
2631 relevant headers.

2632

2633 This Functional Element fulfills the following requirements from the Functional Elements  
2634 Requirements, Working Draft 01a:

- 2635 • Primary Requirements
  - 2636 • DELIVERY-001,
  - 2637 • DELIVERY-005 to DELIVERY-006, and
  - 2638 • DELIVERY-009.
- 2639 • Secondary Requirements
  - 2640 • MANAGEMENT-011, and
  - 2641 • MANAGEMENT-096.

2642

### 2643 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.

2644

2645 **2.11.3 Key Features**

2646 Implementations of the Sensory Functional Element are expected to provide the following key  
 2647 features:

- 2648 1. The Functional Element MUST intercept HTTP requests from client and determines existing  
 2649 supportability of the request's MIME type.
- 2650 2. The Functional Element MUST provide the mechanism to manage MIME types, including  
 2651 the ability to add, delete and retrieve supported MIME types.

2652

2653 In addition, the following key features could be provided to enhance the Functional Element  
 2654 further:

- 2655 1. The Functional Element MAY provide a mechanism to enable Location Based Services  
 2656 (LBS).

2657 **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.

2658 **2.11.5 Related Technologies and Standards**

2659 None.

2660

2661 **2.11.6 Model**

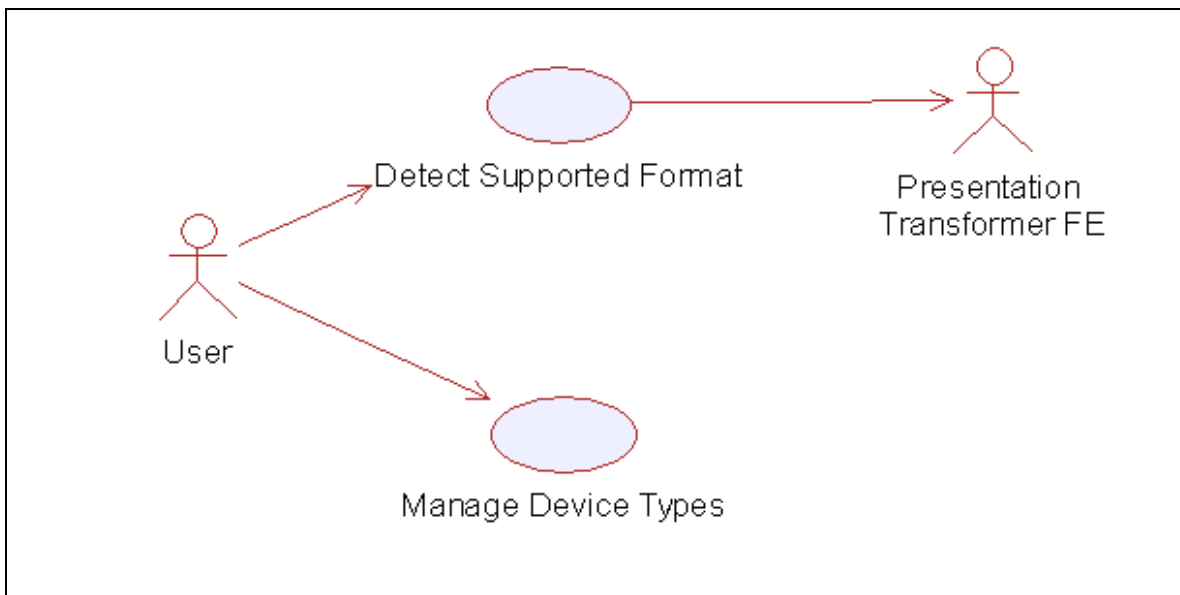


Figure 12: Model Of the Sensory Functional Element [21]

2662 **2.11.7 Usage Scenarios**

2663 **2.11.7.1 Detect Supported Format**

2664 **2.11.7.1.1 Description**

2665 This use case allows the service user (user/other service) to make request and based on that  
2666 request it detects service user's device capabilities.

2667 **2.11.7.1.2 Flow of Events**

2668 **2.11.7.1.2.1 Basic Flow**

2669 This use case starts when the service user wishes to use any service provided by the service  
2670 provider.

2671 1: The Functional Element receives the request from the service user.

2672 2: The Functional Element extracts MIME name and MIME type from the service user's HTTP  
2673 request (even from SOAP request).

2674 3: The Functional Element uses MIME name and MIME TYPE to check with the registered MIME  
2675 type.

2676 4: The Functional Element sends device capabilities to service user and ends the use case.

2677 **2.11.7.1.2.2 Alternative Flows**

2678 1: Unsupported Device.

2679 1.1 If in the basic flow 2, the Functional Element is unable to detect the service user' device  
2680 capability, the Functional Element returns a error message and the use case ends.

2681 **2.11.7.1.3 Special Requirements**

2682 None

2683 **2.11.7.1.3.1 Supportability**

2684 The edge devices must be able to support the HTTP request.

2685 **2.11.7.1.4 Pre-Conditions**

2686 None.

2687 **2.11.7.1.5 Post-Conditions**

2688 None.

2689 **2.11.7.2 Manage Device Types**

2690 **2.11.7.2.1 Description**

2691 This use case allows the service user to maintain the device (MIME Type information). This  
2692 includes adding, changing and deleting device information from the Functional Element.

2693 **2.11.7.2.2 Flow of Events**

2694 **2.11.7.2.2.1 Basic Flow**

2695 This use case starts when the service user wishes to add or delete either device or service  
2696 information from the Functional Element.

2697 1: The Functional Element requests that the service user specify the function he/she would like to  
2698 perform (either add, update or delete device or service).

2699 2: Once the service user provides the requested information, one of the sub-flows is executed.

2700 • If the service user provides '**Add Device Types**', then sub-flow 2.1 is executed.

2701 • If the service user provides '**Delete Device Types**', then sub-flow 2.2 is executed.

2702 2.1: Add Device Type.

2703 2.1.1: The Functional Element requests that the service user provide the device  
2704 information. This includes: MIME Name, MIME Description, Supported MIME type.

2705 2.1.2: Once the service user provides the requested information, the Functional Element  
2706 generates and assigns a unique MIME Id number to the device.

2707 2.2: Delete Device Type.

2708 2.2.1: The Functional Element requests that the service user provide the Device ID.

2709 2.2.2: The Functional Element retrieves the existing device information based on the  
2710 Device ID.

2711 2.2.3: The service user provides the delete device information and the Functional  
2712 Element deletes the device record from the Functional Element.

2713 3: The use case ends when the service user provides the requested information or decided to  
2714 end use case.

2715 **2.11.7.2.2.2 Alternative Flows**

2716 1: Invalid Device Information.

2717 1.1: If in the sub-flow 2.1.2, the requested information provided by the user is invalid, the  
2718 Functional Element returns an error message and the use case ends

2719 2: Device Not Found.

2720 2.1 If in the basic flows 2.2.2, the device information with the specified device is not found or  
2721 does not exist, the Functional Element returns an error message and the use case ends.

2722 **2.11.7.2.3 Special Requirements**

2723 **2.11.7.2.3.1 Supportability**

2724 Manage Device Types supports the most widespread MIME types used today.

2725 **2.11.7.2.4 Pre-Conditions**

2726 None.

2727 **2.11.7.2.5 Post-Conditions**

2728 If the use case was successful, the device information is added, updated or deleted from the  
2729 Functional Element. Otherwise, the Functional Element's state is unchanged.

## 2730 2.12 Service Management Functional Element

### 2731 2.12.1 Motivation

2732 The ability to monitor Web Services invocation is crucial towards the adoption of this technology  
2733 from the security and performance standpoints. A security framework should incorporate an  
2734 authentication and authorisation mechanism together with an audit trail. These twin  
2735 considerations will serve to discourage resource misuse and in addition, will help to promote the  
2736 “pay-as-you-use” concept. Service throughput on the server end is another important parameter  
2737 that must be monitored. Administrators of services, which are sluggish, should be notified  
2738 immediately via any electronic means.

2739

2740 This Functional Element fulfills the following requirements from the Functional Elements  
2741 Requirements, Working Draft 01a:

- 2742 • Primary Requirements
  - 2743 • MANAGEMENT-090, and
  - 2744 • MANAGEMENT-093 to MANAGEMENT-096.
- 2745 • Secondary Requirements
- 2746 • None

### 2747 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

2748

### 2749 2.12.3 Key Features

2750 Implementations of the Service Management Functional Element are expected to provide the  
2751 following key features:

- 2752 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)*
- 2753 2. The Functional Element MUST provide the capability to discover services that are under the  
2754 Management Domain.
- 2755 3. The Functional Element MUST provide the capability to configure Performance Parameters  
2756 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.*

2757 4. The Functional Element MUST provide a means to log Performance Parameters.

2758

2759 In addition, the following key feature could be provided to enhance the Functional Element  
2760 further:

2761 1. The Functional Element MAY provide the capability to configure additional attributes that is  
2762 tagged along with a particular Web Service.

*Example: The access permission for invoking the service.*

2763 2. The Functional Element MAY provide verification services to block unauthorized Web  
2764 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.*

2765 **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

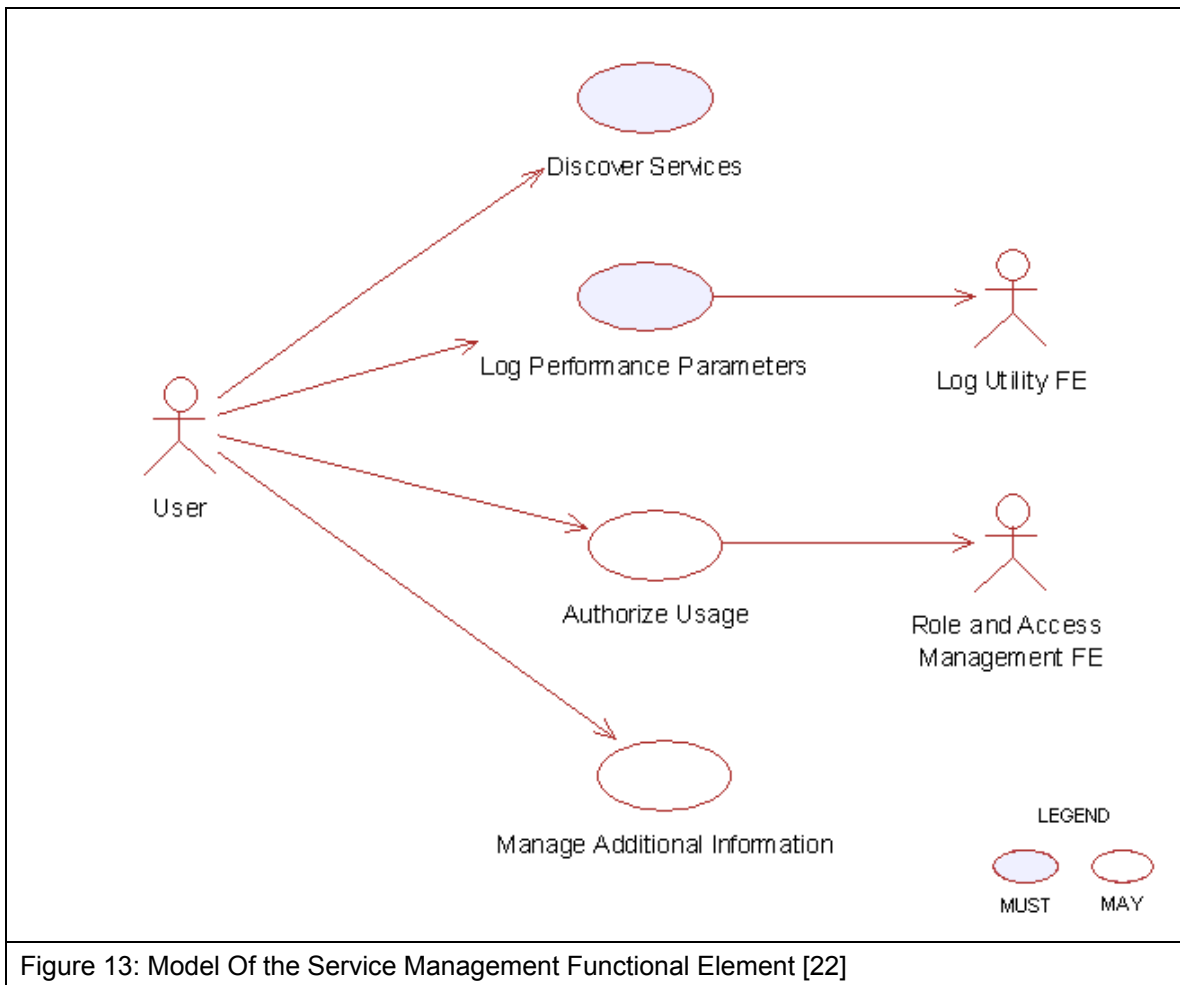
2766

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.

2767 **2.12.5 Related Technologies and Standards**

2768 None

## 2.12.6 Model



## 2770 2.12.7 Usage Scenarios

### 2771 2.12.7.1 Discover Services

#### 2772 2.12.7.1.1 Description

2773 This use case describes the scenario surrounding the automatic discovery of services hosted in  
2774 the Management Domain.

#### 2775 2.12.7.1.2 Flow of Events

##### 2776 2.12.7.1.2.1 Basic Flow

2777 The use case begins when the user wants to retrieve a list of services URLs from the  
2778 Management Domain.

- 2779 1: The user sends a request to retrieve the list of services URLs from the Management Domain.
- 2780 2: The Functional Element reads from a configuration file to so as to determine the exact  
2781 boundaries of the Management Domain.
- 2782 3: The Functional Element retrieves from each of the servers as stated in the configuration file a  
2783 list of service URLs that it is hosting
- 2784 4: The Functional Element returns the list of service URLs back to the user and the use case  
2785 ends.

2786 **2.12.7.1.2.2 Alternative Flows**

2787 1: Configuration File Does Not Exist

2788 1.1: In basic flow 2, the Functional Element fails to read boundaries from the configuration  
2789 file. The Functional Element in turn return an error message and the use case end.

2790 2: Fail To Communicate With the Server

2791 2.1: In basic flow 3, the Functional Element fails to communicate with the servers hosting the  
2792 services. The Functional Element in turn return an error message and the use case end.

2793 **2.12.7.1.3 Special Requirements**

2794 The protocol of communicating with a server hosting the services is not standardized. Each  
2795 server may offer different mechanism for retrieving the list of services hosted and as such, the  
2796 extensibility this approach is severely limited.

2797 **2.12.7.1.4 Pre-Conditions**

2798 None.

2799 **2.12.7.1.5 Post-Conditions**

2800 None

2801 **2.12.7.2 Log Performance Parameters**

2802 **2.12.7.2.1 Description**

2803 This use case allows the user to log the performance parameters of all the Web Services that is  
2804 being hosted by an application that contains the Service Management Functional Element.

2805 **2.12.7.2.2 Flow of Events**

2806 **2.12.7.2.2.1 Basic Flow**

2807 The use case begins when the user wants to log the performance parameters of all the Web  
2808 Services that is being hosted by an application that contains the Service Management Functional  
2809 Element.

2810 1: The user sends a request to log the performance parameters of all the Web Services hosted.

2811 2: The Functional Element reads from a configuration file the performance parameter to be  
2812 logged.

2813 3: The Functional Element extracts the performance parameters for the incoming message and  
2814 stores them into the data store

2815 4: The Functional Element next extracts the performance parameters for the outgoing message  
2816 and stores them into the data store

2817 5: The Functional Element stores the necessary information into the data store.

2818 **2.12.7.2.2.2 Alternative Flows**

2819 1: No Performance Parameter Found.

2820 1.1: In basic flow 2, the Functional Element discovers that the performance parameter to be  
2821 logged is not configured. The Functional Element returns an error message and the use case  
2822 ends.

2823 2: Data Store Not Available.

2824 2.1: In basic flow 5, the Functional Element detects that the data store is not available. The  
2825 Functional Element returns an error message and the use case ends.

2826 **2.12.7.2.3 Special Requirements**

2827 None.

2828 **2.12.7.2.4 Pre-Conditions**

2829 None.

2830 **2.12.7.2.5 Post-Conditions**

2831 None.

2832 **2.12.7.3 Authorize Usage**

2833 **2.12.7.3.1 Description**

2834 This use case describes the authentication process for invoking a Web Service that is being  
2835 hosted by an application that contains the Service Management Functional Element.

2836 **2.12.7.3.2 Flow of Events**

2837 **2.12.7.3.2.1 Basic Flow**

2838 The use case starts when a user accesses a service.

2839 1: The user sends a request to invoke a particular Web Service.

2840 2: The Functional Element extracts the following information from the incoming message

2841 2.1: The username attribute that resides in the header of the incoming message

2842 3: The Functional Element extracts the access privilege associated with the service from the data  
2843 store

2844 4: The Functional Element uses the Role and Access Management Functional Element to retrieve  
2845 the role of the user.

2846 5: The Functional Element looks up the data store to determine if the user is authorized to access  
2847 the service

2848 6: The Functional Element allows the request to be process and the use case ends.

2849 **2.12.7.3.2.2 Alternative Flow**

2850 1: Username header not found.

2851 1.1: In basic flow 2, the username attribute is not found in the header.

2852 1.2: The Functional Element denies access to the requested Web Service and returns an  
2853 error message.

2854 2: Web Service access privilege not set.

2855 2.1: In basic flow 3, the Functional Element could not find the access privilege for the Web  
2856 Service.

2857 2.2: The Functional Element denies access to the requested Web Service and returns an  
2858 error message.

2859 3: Role and Access Management Functional Element not available

2860 3.1: In basic flow 4, the Functional Element could not find the Role and Access Management  
2861 Functional Element.

2862 3.2: The Functional Element denies access to the requested Web Service and returns an  
2863 error message.

2864 4: User not authorize

2865 4.1: In basic flow 5, the Functional Element looks up the data source and determines that the  
2866 user does not have the required privilege to access the service.

2867 4.2: The Functional Element denies access to the requested Web Service and returns an  
2868 error message.

2869 **2.12.7.3.3 Special Requirements**

2870 None.

2871 **2.12.7.3.4 Pre-Conditions**

2872 None.

2873 **2.12.7.3.5 Post-Conditions**

2874 None.

2875 **2.12.7.4 Manage Additional Information**

2876 **2.12.7.4.1 Description**

2877 This use case helps to maintain the following attributes of a Web Service that is useful in  
2878 determining if a particular user has the privilege to invoke it.

2879

- Service Name. This is the name of the service to monitor

2880

- Access level. This refers to the access level of the Web Services hosted

2881

- Role Names. If a user's role matches any of the roles contained here, then he/she has  
2882 the privilege to access the Web Service.

2883 **2.12.7.4.2 Flow of Events**

2884 **2.12.7.4.2.1 Basic Flow**

2885 This use case starts when user wants to manage services.

2886 1: The user specifies the additional information that he wants to create/update/delete/retrieve.

2887 2: Once the user provides the requested information, one of the sub-flows is executed.

2888 • If the user provides '**Create Service Parameter**', then sub-flow 2.1 is executed.

2889 • If the user provides '**Update Service Parameter**', then sub-flow 2.2 is executed.

2890 • If the user provides '**Delete Service Parameter**', then sub-flow 2.3 is executed.

2891 • If the user provides '**Retrieve Service Parameter**', then sub-flow 2.4 is executed.

2892 2.1: Create Service Parameter.

2893 2.1.1: The user specifies the service to create with the appropriate additional information.

2894 2.1.2: The Functional Element connects to the data store.

2895 2.1.3: The Functional Element saves the new service in the data store and the use case  
2896 ends.

2897 2.2: Update Service Parameter.

2898 2.2.1: The user specifies the service to update with the appropriate additional information.

2899 2.2.2: The Functional Element connects to the data store.

2900 2.2.3: The Functional Element updates the service in the data store and the use case  
2901 ends.

2902 2.3: Delete Service Parameter.

2903 2.3.1: The user specifies the service to delete.

2904 2.3.2: The Functional Element connects to the data store.

2905 2.3.3: The Functional Element deletes the service in the data store and the use case  
2906 ends.

- 2907 2.4: Retrieve Service Parameter.
- 2908 2.4.1: The user specifies the service to retrieve.
- 2909 2.4.2: The Functional Element connects to the data store.
- 2910 2.4.3: The Functional Element retrieves the service from the data store and the use case  
2911 ends.
- 2912 **2.12.7.4.2.2 Alternative Flows**
- 2913 1: Data Store Not Available.
- 2914 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  
2915 is returned and the use case ends.
- 2916 **2.12.7.4.3 Special Requirements**
- 2917 None.
- 2918 **2.12.7.4.4 Pre-Conditions**
- 2919 None.
- 2920 **2.12.7.4.5 Post-Conditions**
- 2921 None.

## 2922 **2.13 Service Registry Functional Element**

### 2923 **2.13.1 Motivation**

2924 In a Web Service-enabled implementation, there exist the needs to maintain a central repository  
2925 of all the services that are available. This facilitates service lookups as well as management of  
2926 Web Services within the application that contains the Functional Element. In order to achieve  
2927 these expectations, the Functional Element will cover the following aspects.

- 2928 • Simplify management of information in a XML registry server like UDDI and ebXML, and
- 2929 • Simplify information publish and query from a XML registry server like UDDI and ebXML.

2930

2931 This Functional Element fulfills the following requirements from the Functional Elements  
2932 Requirements, Working Draft 01a:

- 2933 • Primary Requirements
  - 2934 • PROCESS-031 to PROCESS-032,
  - 2935 • PROCESS-035, and
  - 2936 • MANAGEMENT-097 to MANAGEMENT-100
- 2937 • Secondary Requirements
  - 2938 • PROCESS-014.

2939

### 2940 **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

2941 **2.13.3 Key Features**

2942 Implementations of the Service Registry Functional Element are expected to provide the following  
2943 key features:

- 2944 1. The Functional Element MUST provide the capability to facilitate the management of the  
2945 following information in a UDDI or an ebXML compliant registry server.
- 2946 1.1. Organisation
- 2947 1.2. Classification / Taxonomy
- 2948 1.3. Web Service
- 2949 1.4. tModel
- 2950 1.5. Service Binding
- 2951 The management of this information includes registering, updating, deleting and searching.
- 2952 2. As part of Key Feature (1), the Functional Element MUST provide the ability to perform the  
2953 operations specified across multiple registry servers.

2954

2955 In addition, the following key feature could be provided to enhance the Functional Element  
2956 further:

- 2957 1. The Functional Element MAY provide a mechanism to enable single step publishing of  
2958 services into registry servers.

2959

2960 **2.13.4 Interdependencies**

2961 None

2962 **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.
--	---

2963

2964 **2.13.6 Model**

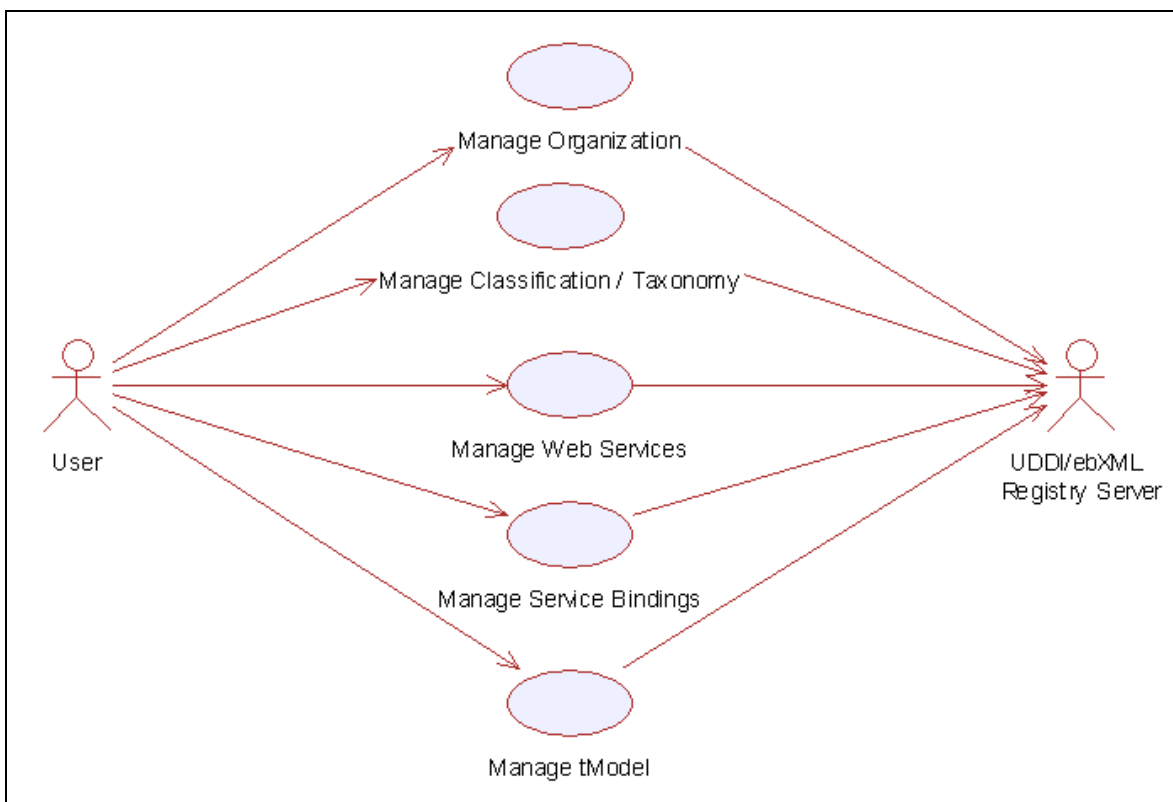


Figure 14: Model Of the Service Registry Functional Element [27]

2965 **2.13.7 Usage Scenario**

2966 **2.13.7.1 Manage Classification / Taxonomy**

2967 **2.13.7.1.1 Description**

2968 This use case allows any users to create, remove and view classification/taxonomy in the  
 2969 registry.

2970 **2.13.7.1.2 Flow of Events**

2971 **2.13.7.1.2.1 Basic Flow**

2972 This use case starts when the users of registry server wishes to create, remove or view the  
 2973 classification/taxonomy in the registry server.

- 2974
- 2975 1: User initiates a request type to the Functional Element stating whether to create, remove or  
2976 view classification/taxonomy.
- 2977 2: The Functional Element checks whether the registry server exists.
- 2978 3: The Functional Element checks the request. Based on the type of request, one of the sub-  
2979 flows is executed.
- 2980 • If the request is to '**Create Classification/Taxonomy**', then sub-flow 3.1 is executed.
  - 2981 • If the request is to '**View Classification/Taxonomy**', then sub-flow 3.2 is executed.
  - 2982 • If the request is to '**Remove Classification/Taxonomy**', then sub-flow 3.3 is executed.
- 2983 3.1: Create Classification/Taxonomy.
- 2984 3.1.1: Other Functional Element provides username, password and registry server URL  
2985 to the Functional Element for authentication.
- 2986 3.1.2: The Functional Element checks for the user validity in the identified registry server.
- 2987 3.1.3: Other Functional Element provides classification/taxonomy information to be  
2988 created in the registry server.
- 2989 3.1.4: The Functional Element checks for the duplicate classification/taxonomy name.
- 2990 3.1.5: The Functional Element creates the classification/taxonomy information in the  
2991 private (default) or the public UDDI registry server according to the URL provided by  
2992 other Functional Element, if it does not exist.
- 2993 3.2: View Classification/Taxonomy.
- 2994 3.2.1: The Functional Element retrieves all the classification/taxonomy from the identified  
2995 registry server, which may be private (default) or public.
- 2996 3.2.2: The Functional Element returns the classification/taxonomy information from the  
2997 identified registry server to other Functional Element.
- 2998 3.3: Remove Classification/Taxonomy.
- 2999 3.3.1: Other Functional Element provides username, password and registry server URL  
3000 to the Functional Element for authentication.
- 3001 3.3.2: The Functional Element checks for the user validity in the identified registry server.

3002 3.3.3: Other Functional Element provides classification/taxonomy key (i.e. UUID) to be  
3003 removed from the identified registry server.

3004 3.3.4: The Functional Element removes the classification/taxonomy information from the  
3005 private (default) or the public UDDI registry server according to the URL provided by the  
3006 user.

3007 4: The Functional Element returns the status of the operation and the use case ends.

3008 **2.13.7.1.2.2 Alternative Flows**

3009 1: Registry Server Down.

3010 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element  
3011 returns an error message and the use case ends.

3012 2: Invalid Username And Password.

3013 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional  
3014 Element returns an error message and the use case ends.

3015 3: Classification/Taxonomy Key Not Found.

3016 3.1: In the basic flow 3.3.3, if the classification/taxonomy key cannot be found in the  
3017 specified registry server, the Functional Element returns an error message and the use  
3018 case ends.

3019 4: Duplicate Classification/Taxonomy.

3020 4.1: In the basic flow 3.1.4, If the same classification/taxonomy name has been defined in  
3021 the registry server, the Functional Element returns an error message and the use case  
3022 ends.

3023 **2.13.7.1.3 Special Requirements**

3024 None

3025 **2.13.7.1.4 Pre-Conditions**

3026 In order to manage the classification/taxonomy in the registry server, users must be registered  
3027 with the registry server. Username and password will be given when a user registers with a  
3028 registry server.

3029 **2.13.7.1.5 Post-Conditions**

3030 None.

3031 **2.13.7.2 Manage Web Services**

3032 **2.13.7.2.1 Description**

3033 This use case allows any users to register, remove and view Web Services in the private (default)  
3034 as well as the public UDDI Registry Server.

3035 **2.13.7.2.2 Flow of Events**

3036 **2.13.7.2.2.1 Basic Flow**

3037 This use case starts when the users of registry server wishes to create, remove and view Web  
3038 Services.

3039 1: User initiates a request type to the Functional Element stating whether to create, remove or  
3040 view Web Services in the identified private or public registry server.

3041 2: The Functional Element checks whether the registry server exists.

3042 3: The Functional Element checks the request. Based on the type of request, one of the sub-  
3043 flows is executed.

3044 • If the request is to '**Create Web Service**', then sub-flow 3.1 is executed.

3045 • If the request is to '**View Web Services**', then sub-flow 3.2 is executed.

3046 • If the request is to '**Remove Web Service**', then sub-flow 3.3 is executed.

3047 3.1: Create Web Service.

3048 3.1.1: User provides username, password and registry server URL to the Functional  
3049 Element for authentication.

3050 3.1.2: The Functional Element checks for the user validity in the identified registry server.

3051 3.1.3: Other Functional Element provides Web Service information to be created in the  
3052 registry server.

3053 3.1.4: The Functional Element creates the Web Service information in the private  
3054 (default) or the public UDDI registry server according to the URL provided by other  
3055 Functional Element.

3056 3.2: View Web Services.

3057 3.2.1: The Functional Element retrieves all the Web Services from the identified registry  
3058 server for specific stated conditions like service name search, business name search,  
3059 etc.

3060 3.2.2: The Functional Element displays the Web Services information search results from  
3061 the identified registry server to other Functional Element.

3062 3.3: Remove Web Service

3063 3.3.1 User provides username, password and registry server URL to the Functional  
3064 Element for authentication.

3065 3.3.2: The Functional Element checks for the user validity in the identified registry server.

3066 3.3.3: Other Functional Element provides Web Service key (i.e. UUID) to be removed  
3067 from the identified registry server.

3068 3.3.4: The Functional Element removes the Web Service information from the private  
3069 (default) or the public UDDI registry server according to the URL provided by other  
3070 Functional Element.

3071 4: The Functional Element returns the results of the operation and the use case ends.

#### 3072 **2.13.7.2.2.2 Alternative Flows**

3073 1: Registry Server Down.

3074 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element  
3075 returns an error message and the use case ends.

3076 2: Invalid Username And Password.

3077 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional  
3078 Element returns an error message and the use case ends.

3079 3: Web Service Key Not Found.

3080 3.1: In the basic flow 3.3.3, if the Web Service key cannot be found in the specified registry  
3081 server, the Functional Element returns an error message and the use case ends.

#### 3082 **2.13.7.2.3 Special Requirements**

#### 3083 **2.13.7.2.4 Pre-Conditions**

3084 In order to manage Web Services in the registry server, the users must be registered with the  
3085 registry server. Username and password will be given when a user registers with a registry  
3086 server.

#### 3087 **2.13.7.2.5 Post-Conditions**

3088 None.

## 3089 **2.13.7.3 Manage Organization**

### 3090 **2.13.7.3.1 Description**

3091 This use case allows any users to create, remove and view organization in the registry.

### 3092 **2.13.7.3.2 Flow of Events**

#### 3093 **2.13.7.3.2.1 Basic Flow**

3094 This use case starts when the users of registry server wishes to create, remove or view  
3095 Organization.

3096 1: User initiates a request type to the Functional Element stating whether to create, remove or  
3097 view Organization.

3098 2: The Functional Element checks whether the registry server exists.

3099 3: The Functional Element checks the request. Based on the type of request, one of the sub-  
3100 flows is executed.

3101 • If the request is to '**Create Organization**', then sub-flow 3.1 is executed.

3102 • If the request is to '**View Organizations**', then sub-flow 3.2 is executed.

3103 • If the request is to '**Remove Organization**', then sub-flow 3.3 is executed.

3104 3.1: Create Organization.

3105 3.1.1: Other Functional Element provides username, password and registry server URL  
3106 to the Functional Element for authentication.

3107 3.1.2: The Functional Element checks for the user validity in the identified registry server.

3108 3.1.3: Other Functional Element provides organization information to be created in the  
3109 registry server.

3110 3.1.4: The Functional Element checks for the duplicate organization name.

3111 3.1.5: The Functional Element creates the organization information in the private (default)  
3112 or the public UDDI registry server according to the URL provided by other Functional  
3113 Element, if it does not exist.

3114 3.2: View Organizations.

3115 3.2.1: The Functional Element retrieves all the organizations from the identified registry  
3116 server for specific stated conditions like organization name, key, etc.

- 3117 3.2.2: The Functional Element returns the organization information from the identified  
3118 registry server to other Functional Element.
- 3119 3.3: Remove Organization.
- 3120 3.3.1: Other Functional Element provides username, password and registry server URL  
3121 to the Functional Element for authentication.
- 3122 3.3.2: The Functional Element checks for the user validity in the identified registry server.
- 3123 3.3.3: Other Functional Element provides Organization key (i.e. UUID) to be removed  
3124 from the identified registry server.
- 3125 3.3.4: The Functional Element removes the Organization information from the private  
3126 (default) or the public UDDI registry server according to the URL provided by the user.
- 3127 4: The Functional Element returns the status of the operation and the use case ends.

3128 **2.13.7.3.2.2 Alternative Flows**

- 3129 1: Registry Server Down.
- 3130 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element  
3131 returns an error message and the use case ends.
- 3132 2: Invalid Username And Password.
- 3133 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional  
3134 Element returns an error message and the use case ends.
- 3135 3: Organization Key Not Found.
- 3136 3.1: In the basic flow 3.3.3, if the Organization key cannot be found in the specified registry  
3137 server, the Functional Element returns an error message and the use case ends.
- 3138 4: Duplicate Organization.
- 3139 4.1: In the basic flow 3.1.4, If the same Organization name has been defined in the registry  
3140 server the Functional Element returns an error message and the use case ends.

3141 **2.13.7.3.3 Special Requirements**

3142 None

3143 **2.13.7.3.4 Pre-Conditions**

3144 In order to manage Organization in the registry server, users must be registered with the registry  
3145 server. Username and password will be given when a user registers with a registry server.

3146 **2.13.7.3.5 Post-Conditions**

3147 None.

## 3148 **2.13.7.4 Manage Service Binding**

### 3149 **2.13.7.4.1 Description**

3150 This use case allows any users to register, remove and view Service Binding in the private  
3151 (default) as well as the public UDDI Registry Server.

### 3152 **2.13.7.4.2 Flow of Events**

#### 3153 **2.13.7.4.2.1 Basic Flow**

3154 This use case starts when the users of registry server wishes to create, remove and view Service  
3155 Binding.

3156 1: User initiates a request type to the Functional Element stating whether to create, remove or  
3157 view Service Binding in the identified private or public registry server.

3158 2: The Functional Element checks whether the registry server exists.

3159 3: The Functional Element checks the request. Based on the type of request, one of the sub-  
3160 flows is executed.

3161 • If the request is to '**Create Service Binding**', then sub-flow 3.1 is executed.

3162 • If the request is to '**View Service Bindings**', then sub-flow 3.2 is executed.

3163 • If the request is to '**Remove Service Binding**', then sub-flow 3.3 is executed.

3164 3.1: Create Service Binding.

3165 3.1.1: User provides username, password and registry server URL to the Functional  
3166 Element for authentication.

3167 3.1.2: The Functional Element checks for the user validity in the identified registry server.

3168 3.1.3: Other Functional Element provides Service Binding information to be created in the  
3169 registry server.

3170 3.1.4: The Functional Element creates the Service Binding information in the private  
3171 (default) or the public UDDI registry server according to the URL provided by other  
3172 Functional Element.

3173 3.2: View Service Bindings.

3174 3.2.1: The Functional Element retrieves all the Service Bindings from the identified  
3175 registry server for specific stated conditions like service binding key search, etc.

- 3176 3.2.2: The Functional Element displays the Service Bindings information search results  
3177 from the identified registry server to other Functional Element.
- 3178 3.3: Remove Service Binding
- 3179 3.3.1 User provides username, password and registry server URL to the Functional  
3180 Element for authentication.
- 3181 3.3.2: The Functional Element checks for the user validity in the identified registry server.
- 3182 3.3.3: Other Functional Element provides Service Binding key (i.e. UUID) to be removed  
3183 from the identified registry server.
- 3184 3.3.4: The Functional Element removes the Service Binding information from the private  
3185 (default) or the public UDDI registry server according to the URL provided by other  
3186 Functional Element.
- 3187 4: The Functional Element returns the results of the operation and the use case ends.
- 3188 **2.13.7.4.2.2 Alternative Flows**
- 3189 1: Registry Server Down.
- 3190 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element returns  
3191 an error message and the use case ends.
- 3192 2: Invalid Username And Password.
- 3193 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional  
3194 Element returns an error message and the use case ends.
- 3195 3: Service Binding Key Not Found.
- 3196 3.1: In the basic flow 3.3.3, if the Service Binding key cannot be found in the specified registry  
3197 server, the Functional Element returns an error message and the use case ends.
- 3198 **2.13.7.4.3 Special Requirements**
- 3199 **2.13.7.4.4 Pre-Conditions**
- 3200 In order to manage Service Binding in the registry server, the users must be registered with the  
3201 registry server. Username and password will be given when a user registers with a registry  
3202 server.

3203 **2.13.7.4.5 Post-Conditions**

3204 None.

3205 **2.13.7.5 Manage tModel**

3206 **2.13.7.5.1 Description**

3207 This use case allows any users to register, remove and view tModel in the private (default) as  
3208 well as the public UDDI Registry Server.

3209 **2.13.7.5.2 Flow of Events**

3210 **2.13.7.5.2.1 Basic Flow**

3211 This use case starts when the users of registry server wishes to create, remove and view tModel.

3212 1: User initiates a request type to the Functional Element stating whether to create, remove or  
3213 view tModel in the identified private or public registry server.

3214 2: The Functional Element checks whether the registry server exists.

3215 3: The Functional Element checks the request. Based on the type of request, one of the sub-  
3216 flows is executed.

3217 • If the request is to '**Create tModel**', then sub-flow 3.1 is executed.

3218 • If the request is to '**View tModels**', then sub-flow 3.2 is executed.

3219 • If the request is to '**Remove tModel**', then sub-flow 3.3 is executed.

3220 3.1: Create tModel.

3221 3.1.1: User provides username, password and registry server URL to the Functional  
3222 Element for authentication.

3223 3.1.2: The Functional Element checks for the user validity in the identified registry server.

3224 3.1.3: Other Functional Element provides tModel information to be created in the registry  
3225 server.

3226 3.1.4: The Functional Element creates the tModel information in the private (default) or  
3227 the public UDDI registry server according to the URL provided by other Functional  
3228 Element.

3229 3.2: View tModels.

- 3230 3.2.1: The Functional Element retrieves all the tModels from the identified registry server  
3231 for specific stated conditions like tModel name search, tModel key search, etc.
- 3232 3.2.2: The Functional Element displays the tModel information search results from the  
3233 identified registry server to other Functional Element.
- 3234 3.3: Remove tModel.
- 3235 3.3.1 User provides username, password and registry server URL to the Functional  
3236 Element for authentication.
- 3237 3.3.2: The Functional Element checks for the user validity in the identified registry server.
- 3238 3.3.3: Other Functional Element provides tModel key (i.e. UUID) to be removed from the  
3239 identified registry server.
- 3240 3.3.4: The Functional Element removes the tModel information from the private (default)  
3241 or the public UDDI registry server according to the URL provided by other Functional  
3242 Element.
- 3243 4: The Functional Element returns the results of the operation and the use case ends.

#### 3244 **2.13.7.5.2.2 Alternative Flows**

- 3245 1: Registry Server Down.
- 3246 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element returns  
3247 an error message and the use case ends.
- 3248 2: Invalid Username And Password.
- 3249 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional  
3250 Element returns an error message and the use case ends.
- 3251 3: tModel Key Not Found.
- 3252 3.1: In the basic flow 3.3.3, if the tModel key cannot be found in the specified registry server,  
3253 the Functional Element returns an error message and the use case ends.

#### 3254 **2.13.7.5.3 Special Requirements**

#### 3255 **2.13.7.5.4 Pre-Conditions**

- 3256 In order to manage tModel in the registry server, the users must be registered with the registry  
3257 server. Username and password will be given when a user registers with a registry server.

3258 **2.13.7.5.5 Post-Conditions**

3259 None.

## 3260 2.14 Service Tester Functional Element

### 3261 2.14.1 Motivation

3262 In a Web Service environment where the lifecycle of services may be rather dynamic, there exist  
3263 a need for a client to dynamically discover the capabilities of the hosted service and bind to these  
3264 services dynamically. The later is the main motivation of this Functional Element.

3265

3266 This Functional Element fulfills the following requirements from the Functional Elements  
3267 Requirements, Working Draft 01a:

- 3268 • Primary Requirements
  - 3269 • MANAGEMENT-091,
  - 3270 • MANAGEMENT-094, and
  - 3271 • PROCESS-130 to PROCESS-132.
- 3272 • Secondary Requirements
  - 3273 • PROCESS-133.

### 3274 2.14.2 Terms Used

Terms	Description
WSDL	Web Services Description Language

### 3275 2.14.3 Key Features

3276 Implementations of the Service Tester Functional Element are expected to provide the following  
3277 key features:

- 3278 1. The Functional Element MUST provide the capability to generate a Web Service client from  
3279 a WSDL file.
- 3280 2. The Functional Element MUST provide the capability to test the availability of Web Services  
3281 based on the generated Web Service client.

3282 *Example: To retrieve the response time of a particular user-specified Web Service*  
3283 *operation to test the availability of a Web Service.*

### 3284 2.14.4 Interdependencies

3285 None

### 3286 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.
---------------	---

3287 **2.14.6 Model**

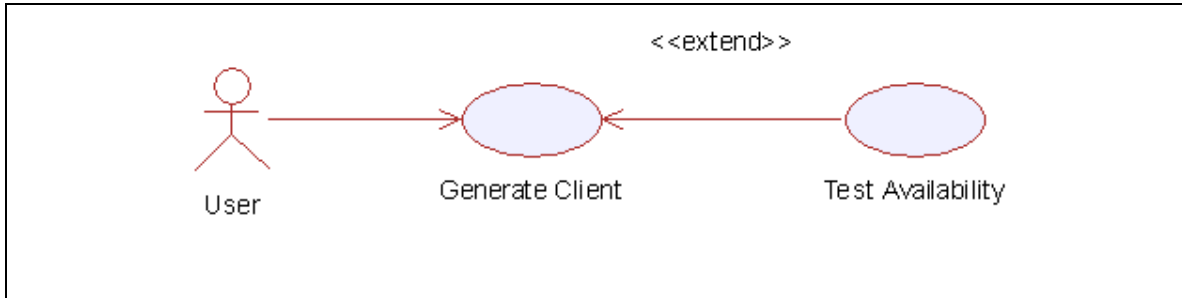


Figure 15: Model Of the Service Tester Functional Element [29]

3288

3289 **2.14.7 Usage Scenarios**

3290 **2.14.7.1 Generate Client**

3291 **2.14.7.1.1 Description**

3292 This use case describes the steps to generate a dynamic client

3293 **2.14.7.1.2 Flow of Events**

3294 **2.14.7.1.2.1 Basic Flow**

3295 This use case starts when the user wants to create a dynamic client

3296 1: User submits the WSDL of the Web Service.

3297 2: Functional Element parses the WSDL document and extracts the necessary information.

3298 3: Functional Element generates the client base on the available parameters.

3299 **2.14.7.1.2.2 Alternative Flows**

3300 1: Invalid WSDL

3301 11: In basic flow 2, if the structure of the WSDL does not comply with the standard, the  
 3302 Functional Element returns an error message and the use case ends.

3303 **2.14.7.1.3 Special Requirements**

3304 None.

3305 **2.14.7.1.4 Pre-Conditions**

3306 None.

3307 **2.14.7.1.5 Post-Conditions**

3308 None.

3309 **2.14.7.2 Test Availability**

3310 **2.14.7.2.1 Description**

3311 This use case allows the user to test the availability of a Web Service.

3312 **2.14.7.2.2 Flow of Events**

3313 **2.14.7.2.2.1 Basic Flow**

3314 This use case starts when a user wants to test the availability of a Web Service.

3315 1: User forms the dynamic client as describe in the use case 'Generate Client'.

3316 2: User inputs the acceptable response time for the purpose of testing the service.

3317 3: Functional Element invokes the web service and waits for the response. The response time is  
3318 then compared with the stipulated time and the result is subsequently returned to the user.

3319 **2.14.7.2.2.2 Alternative Flows**

3320 1: Failure to Generate the Client

3321 1.1: In basic flow 1, if the Functional Element fails to generate the client, the Functional  
3322 Element returns an error message and the use case ends.

3323 2: Time Out

3324 2.1: In basic flow 3, if the response if not returned within the stipulated time, the Functional  
3325 Element returns an error message and the use case ends.

3326

3327 **2.14.7.2.3 Special Requirements**

3328 None.

3329 **2.14.7.2.4 Pre-Conditions**

3330 None.

3331 **2.14.7.2.5 Post-Conditions**

3332 None.

## 3333 2.15 User Management Functional Element

### 3334 2.15.1 Motivation

3335 The User Management Functional Element is expected to be an integral part of the user access  
3336 management (UAM) functionalities that is expected to be needed by a Web Service-enabled  
3337 implementation. This FE is expected to fulfill the needs arising out of managing resources within  
3338 an application, with a user-centric viewpoint. As such it will cover aspects that include:

- 3339 • Basic user accounts management facilities,
- 3340 • Ability to extend dynamically from the basic set of account information,
- 3341 • Capability for configurable policies governing account management,
- 3342 • Providing log trails for user activities, and
- 3343 • Management of user authentication means, either directly or indirectly.

3344

3345 This Functional Element fulfills the following requirements from the Functional Elements  
3346 Requirements, Working Draft 01a:

- 3347 • Primary Requirements
  - 3348 • MANAGEMENT-001 to MANAGEMENT-003,
  - 3349 • MANAGEMENT-005,
  - 3350 • MANAGEMENT-008,
  - 3351 • MANAGEMENT-012, and
  - 3352 • SECURITY-002 (all).
- 3353 • Secondary Requirements
  - 3354 • SECURITY-001.

3355

### 3356 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	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"> <li>• Defining a set of basic user information that should be stored in any enterprise application.</li> <li>• Providing a means to extend this basic set of user information when needed.</li> <li>• Simplifying management by grouping related users together through certain criteria.</li> <li>• Having the flexibility of adopting both coarse/fine grain access controls.</li> </ul>
User Repository	User Repository is where the user information is stored. It can be a database or a flat file.

3357

### 3358 **2.15.3 Key Features**

3359 Implementations of the User Management Functional Element are expected to provide the  
3360 following key features:

- 3361 1. The Functional Element MUST provide a User Repository.
- 3362 2. The Functional Element MUST be able to control access to such a User Repository.
- 3363 3. The Functional Element MUST provide a basic User structure with a set of pre-defined  
3364 attributes.
- 3365 4. The Functional Element MUST provide the capability to extend on the basic User structure  
3366 dynamically.
- 3367 5. As part of Key Feature (4), this dynamic extension MUST be definable and configurable at  
3368 runtime implementation of the Functional Element.
- 3369 6. The Functional Element MUST provide the capability to manage the creation and deletion of  
3370 instances of Users based on defined structure.
- 3371 7. The Functional Element MUST provide the capability to manage all the information (attribute  
3372 values) stored in such Users. This includes the capability to:
  - 3373 7.1. Retrieve and update attribute's values belonging to a User,
  - 3374 7.2. Generate a random password,
  - 3375 7.3. Encrypt sensitive user information, and
  - 3376 7.4. Authenticate a user.
- 3377 8. As part of Key Feature (7.4), the authentication of a User MUST be achieved at least  
3378 through the use of a password.
- 3379 9. The Functional Element MUST provide a mechanism for managing Users across different  
3380 application domains.

3381 *Example: Namespace control mechanism*

3382

3383 In addition, the following key features could be provided to enhance the Functional Element  
3384 further:

3385 1. The Functional Element MAY provide a mechanism to control the username format.

3386 *Example: Usernames must be at least 8 characters long.*

3387 2. The Functional Element MAY provide additional security mechanisms to enhance the  
3388 security of sensitive information like user passwords.

3389 *Example: Passwords are stored in security tokens, or a more secure encryption algorithms  
3390 for passwords.*

3391 3. If Key Feature (2) is provided, the Functional Element MAY also provide a selection of  
3392 selectable encryption algorithms.

3393 4. The Functional Element MAY provide additional security policies to ensure that systems are  
3394 not compromised.

3395 *Example: Passwords must be changed every 30 days.*

3396 5. If Key Feature (4) is provided, the Functional Element MAY also provide a facility to notify  
3397 users before the password expires.

3398

## 3399 **2.15.4 Interdependencies**

<b>Interaction Dependencies</b>	
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.

3400

## 3401 **2.15.5 Related Technologies and Standards**

3402 None

## 3403 **2.15.6 Model**

3404

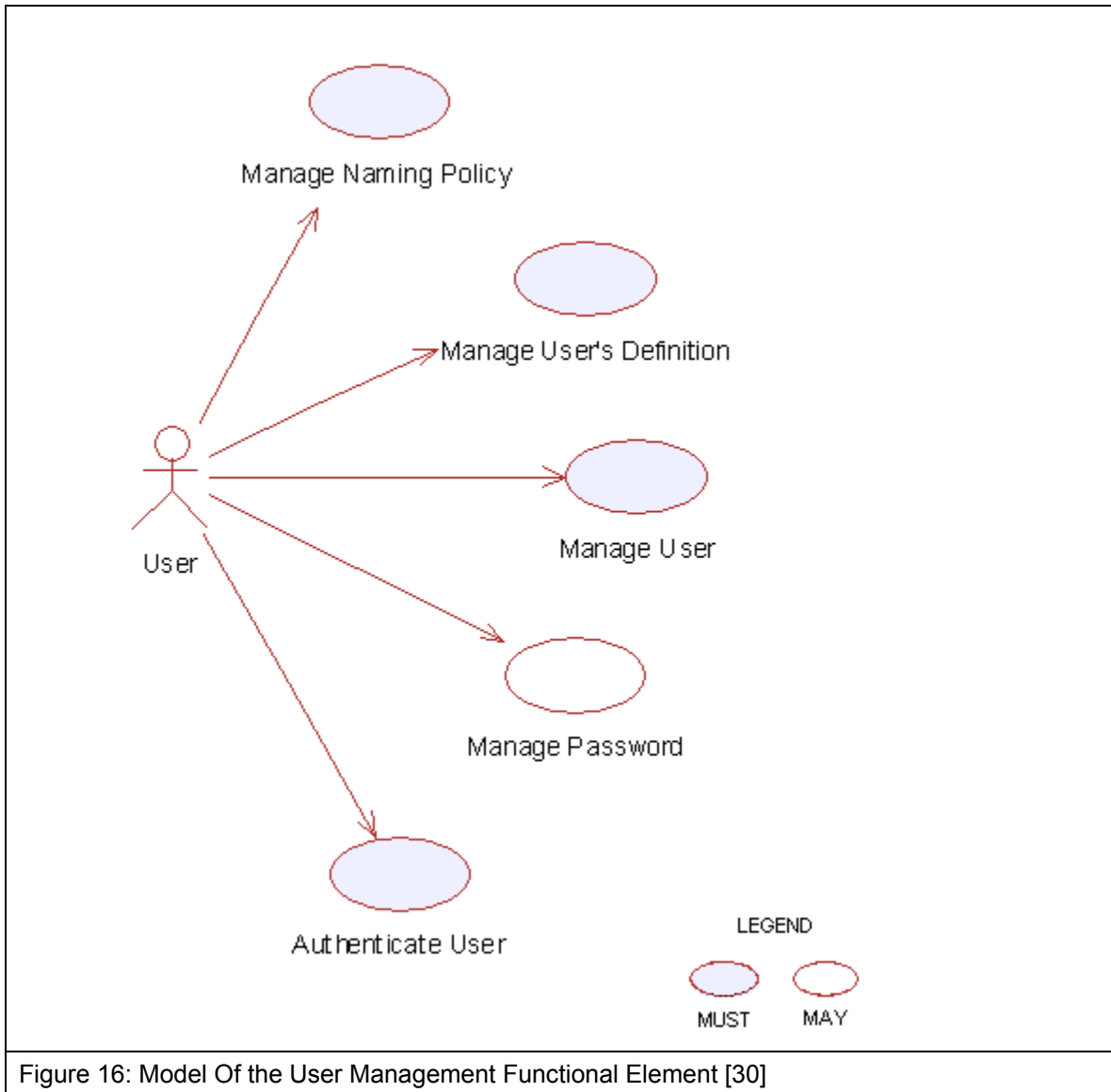


Figure 16: Model Of the User Management Functional Element [30]

## 3405 2.15.7 Usage Scenarios

### 3406 2.15.7.1 Manage Naming Policy

#### 3407 2.15.7.1.1 Description

3408 This use case allows any user to manage naming policy when creating/updating user accounts.  
 3409 The service user may create, update, retrieve and delete a naming policy.

3410 **2.15.7.1.2 Flow of Events**

3411 **2.15.7.1.2.1 Basic Flow**

3412 This use case starts when any user wants to manage naming policy for creating/updating user  
3413 account.

3414 1: The user sends Manage Naming Policy request to the Functional Element together with the  
3415 specified operation.

3416 2: Functional Element gets the operation. Based on the operation, one of the sub-flows is  
3417 executed.

- 3418 • If the service user provides '**Create Naming Policy**', then sub-flow 2.1 is executed.
- 3419 • If the service user provides '**Update Naming Policy**', then sub-flow 2.2 is executed.
- 3420 • If the service user provides '**Delete Naming Policy**', then sub-flow 2.3 is executed.

3421 2.1: Create Naming Policy.

3422 2.1.1: The service user specifies namespace, name and description of the policy to  
3423 create, for example, the policy name may be name length, the policy description may be  
3424 "=7".

3425 2.1.2: The Functional Element checks the existing naming policy.

3426 2.1.3: The Functional Element generates naming policy information and adds to the  
3427 Functional Element and the use case ends.

3428 2.2: Update Naming Policy.

3429 2.2.1: The service user specifies the policy to update.

3430 2.2.2: The Functional Element retrieves the existing naming policy information.

3431 2.2.3: The service user provides the update naming policy information according to the  
3432 policy name used in creating a naming policy.

3433 2.2.4: The Functional Element updates the naming policy with the updated information  
3434 and ends use case.

3435 2.3: Retrieve Naming Policy.

3436 2.3.1: The service user specifies the policy to retrieve.

3437 2.3.2: The Functional Element retrieves the existing naming policy information and ends  
3438 the use case.

3439 2.4: Delete Naming Policy.

3440 2.4.1: The service user specifies the policy to delete.

3441 2.4.2: The Functional Element retrieves the existing naming policy information.

3442 2.4.3: The Functional Element deletes the naming policy from the Functional Element  
3443 and the use case ends.

#### 3444 **2.15.7.1.2.2 Alternative Flows**

3445 1: Invalid Policy.

3446 1.1: If in the basic flow 2.1.1, Functional Element detects any invalid description, Functional  
3447 Element returns general error message and ends the use case.

3448 2: Naming Policy already exists.

3449 2.1: If in the basic flow 2.1.2, the Functional Element checks the existing naming policy and  
3450 finds the naming policy already exists. The Functional Element returns an error and ends the  
3451 use case.

#### 3452 **2.15.7.1.3 Special Requirements**

#### 3453 **2.15.7.1.4 Pre-Conditions**

3454 None.

#### 3455 **2.15.7.1.5 Post-Conditions**

3456 If the use case was successful, the naming policy information is added to the Functional Element.  
3457 To do any creating and updating of User information after the naming policy is added must satisfy  
3458 the naming policies defined. If unsuccessful, the Functional Element's state is unchanged.

### 3459 **2.15.7.2 Manage User Definition**

#### 3460 **2.15.7.2.1 Description**

3461 The use case allows any user to manage user definition when more basic user definition can not  
3462 satisfied in creating/updating user accounts. The service user may create, update, retrieve and  
3463 delete a user definition.

3464 **2.15.7.2.2 Flow of Events**

3465 **2.15.7.2.2.1 Basic Flow**

3466 This use case starts when any user wants to manage user definition for creating/updating user  
3467 account.

3468 1: The user sends Manage User Definition request to the Functional Element together with the  
3469 specified operation.

3470 2: Functional Element gets the operation. Based on the operation, one of the sub-flows is  
3471 executed.

3472 • If the service user provides '**Create User Definition**', then sub-flow 2.1 is executed.

3473 • If the service user provides '**Update User Definition**', then sub-flow 2.2 is executed.

3474 • If the service user provides '**Delete User Definition**', then sub-flow 2.3 is executed.

3475 2.1: Create User Definition.

3476 2.1.1: The service user specifies namespace, name and description of the user definition  
3477 fields to create.

3478 2.1.2: The Functional Element checks the existing user definition fields (including basic  
3479 ones).

3480 2.1.3: The Functional Element generates user definition information and adds to the  
3481 Functional Element and the use case ends.

3482 2.2: Update User Definition.

3483 2.2.1: The service user specifies the user definition field to update.

3484 2.2.2: The Functional Element retrieves the existing user definition information.

3485 2.2.3: The service user provides the update user definition information.

3486 2.2.4: The Functional Element updates the user definition with the updated information  
3487 and ends use case.

3488 2.3: Retrieve User Definition.

3489 2.3.1: The service user specifies the user definition to retrieve.

3490 2.3.2: The Functional Element retrieves the existing user definition information and ends  
3491 the use case.

3492 2.4: Delete User Definition.

3493 2.4.1: The service user specifies the user definition to delete.

3494 2.4.2: The Functional Element retrieves the existing user definition information.

3495 2.4.3: The Functional Element deletes the user definition from the Functional Element  
3496 and the use case ends.

### 3497 **2.15.7.2.3 Alternative Flows**

3498 1: Invalid User Definition.

3499 1.1: If in basic flow 2.1.1, Functional Element detects any invalid description, Functional  
3500 Element returns general error message and ends the use case.

3501 2: User Definition already exists.

3502 2.1: If in basic flow 2.1.2, the Functional Element checks the existing user definition and finds  
3503 the user definition already exists. The Functional Element returns an error and ends the use  
3504 case.

3505 3: User Definition not exists.

3506 3.1: If in basic flow 2.2.2, 2.3.2 and 2.4.2, the Functional Element checks the existing user  
3507 definition and finds the user definition does not exist. The Functional Element returns an  
3508 error and ends the use case.

### 3509 **2.15.7.2.4 Special Requirements**

3510 None

### 3511 **2.15.7.2.5 Pre-Conditions**

3512 None.

### 3513 **2.15.7.2.6 Post-Conditions**

3514 If the use case was successful, the user definition information is added to the Functional Element.  
3515 Thereafter, when creating and updating User, the User information must satisfy the user definition  
3516 defined earlier. If the use case fails, the Functional Element's state is unchanged.

3517 **2.15.7.3 Manage User**

3518 This use case describes the management of a user, namely the creation, deletion, retrieval and  
3519 update of the user.

3520 **2.15.7.3.1 Flow of Events**

3521 **2.15.7.3.1.1 Basic Flow**

3522 This use case starts when the user wants to manage a user.

- 3523 • If user wants to **Create User**, then basic flow 1 is executed.
- 3524 • If user wants to **Retrieve User**, then basic flow 2 is executed.
- 3525 • If user wants to **Update User**, then basic flow 3 is executed.
- 3526 • If user wants to **Delete User**, then basic flow 4 is executed.

3527 1: Create User.

3528 1.1: User provides the information that is necessary for creating a user.

3529 1.2: The Functional Element validates the user information provided against the naming  
3530 policy.

3531 1.3: The Functional Element validates the user information provided against the user's  
3532 definition.

3533 1.4: Functional Element creates the user and the use case ends.

3534 2: Retrieve User.

3535 2.1: User provides the necessary information for retrieving the complete user's attributes.

3536 2.2: The Functional Element returns the user's information and the use case ends.

3537 3: Update User.

3538 3.1: User provides the necessary information for updating the group's attributes.

3539 3.2: The Functional Element validates the user's information provided against the naming  
3540 policy.

3541 3.3: The Functional Element validates the user information provided against the user's  
3542 definition.

3543 3.4: The Functional Element updates the user and the use case ends.

3544 4: Delete User.

3545 4.1: User provides the necessary information for deleting a user group.

3546 4.2: Functional Element deletes the user and the use case ends.

3547 **2.15.7.3.1.2 Alternative Flows**

3548 1: User Exist.

3549 1.1: In basic flow 1.4, if the Functional Element detects an identical user, the Functional  
3550 Element returns an error message and the use case ends.

3551 2: User Does Not Exist.

3552 1.1: In basic flow 2.2, 3.4 and 4.2, if the Functional Element cannot find a user that matches  
3553 the user's criteria, the Functional Element returns an error message and the use case ends.

3554 **2.15.7.3.2 Special Requirements**

3555 None.

3556 **2.15.7.3.3 Pre-Conditions**

3557 None.

3558 **2.15.7.3.4 Post-Conditions**

3559 None.

3560 **2.15.7.4 Authenticate User**

3561 **2.15.7.4.1 Description**

3562 This use case allows users to authenticate a user.

3563 **2.15.7.4.2 Flow of Events**

3564 **2.15.7.4.2.1 Basic Flow**

3565 This use case starts when users wish to authenticate a user.

3566 1: Users provide user name and password to Functional Element.

3567 2: The Functional Element checks the user name and password.

3568 3: The Functional Element returns the result to users and the use case ends.

3569 **2.15.7.4.2.2 Alternative Flows**

3570 None.

3571 **2.15.7.4.3 Special Requirements**

3572 None.

3573 **2.15.7.4.4 Pre-Conditions**

3574 None.

3575 **2.15.7.4.5 Post-Conditions**

3576 None.

3577 **2.15.7.5 Manage Password**

3578 This use case describes the management of password in this Functional Element.

3579 **2.15.7.5.1 Flow of Events**

3580 **2.15.7.5.1.1 Basic Flow**

3581 This use case starts when the user wants to obtain an encrypted password. This can be  
3582 achieved via one of the following basic flow.

3583 • If user wants to '**Generate Password**', then basic flow 1 is executed.

3584 • If user wants to '**Encrypt Password**', then basic flow 2 is executed.

3585 1: Generate Password

3586 1.1: The user specifies the option of format of password among available options in the  
3587 Functional Element.

3588 1.2: The Functional Element generates clear text password based on the format specified by  
3589 the service user.

3590 1.3: The Functional Element includes "Encrypt Password" use case to encrypt the clear text  
3591 password.

3592 1.4: The Functional Element returns the clear text password and encrypted password to user  
3593 and the use case ends.

- 3594 2: Encrypt Password
- 3595 1.1: The user provides clear text password to Functional Element.
- 3596 1.2: The user specifies the encryption algorithm to be used.
- 3597 1.3: The Functional Element encrypts the clear text password.
- 3598 1.4: The Functional Element returns the encrypted password to user and the use case ends.

3599 **2.15.7.5.1.2 Alternative Flows**

3600 None.

3601 **2.15.7.5.2 Special Requirements**

3602 None.

3603 **2.15.7.5.3 Pre-Conditions**

3604 None.

3605 **2.15.7.5.4 Post-Conditions**

3606 None.

## 3607 **2.16 Web Service Aggregator Functional Element**

### 3608 **2.16.1 Motivation**

3609 In any Web Service-enabled application, it is expected that complex business functions have to  
3610 be realized via aggregation of multiple Web Services. This Functional Element is expected to  
3611 fulfill the needs arising out of Web Services composition. As such it will cover aspects that  
3612 include:

- 3613 • Facilitating the composition of Web Services, and
- 3614 • Testing of aggregated Web Services.

3615

3616 This Functional Element fulfills the following requirements from the Functional Elements  
3617 Requirements, Working Draft 01a:

- 3618 • Primary Requirements
  - 3619 • PROCESS-010 to PROCESS-014.
- 3620 • Secondary Requirements
  - 3621 • PROCESS-131

3622

### 3623 **2.16.2 Terms Used**

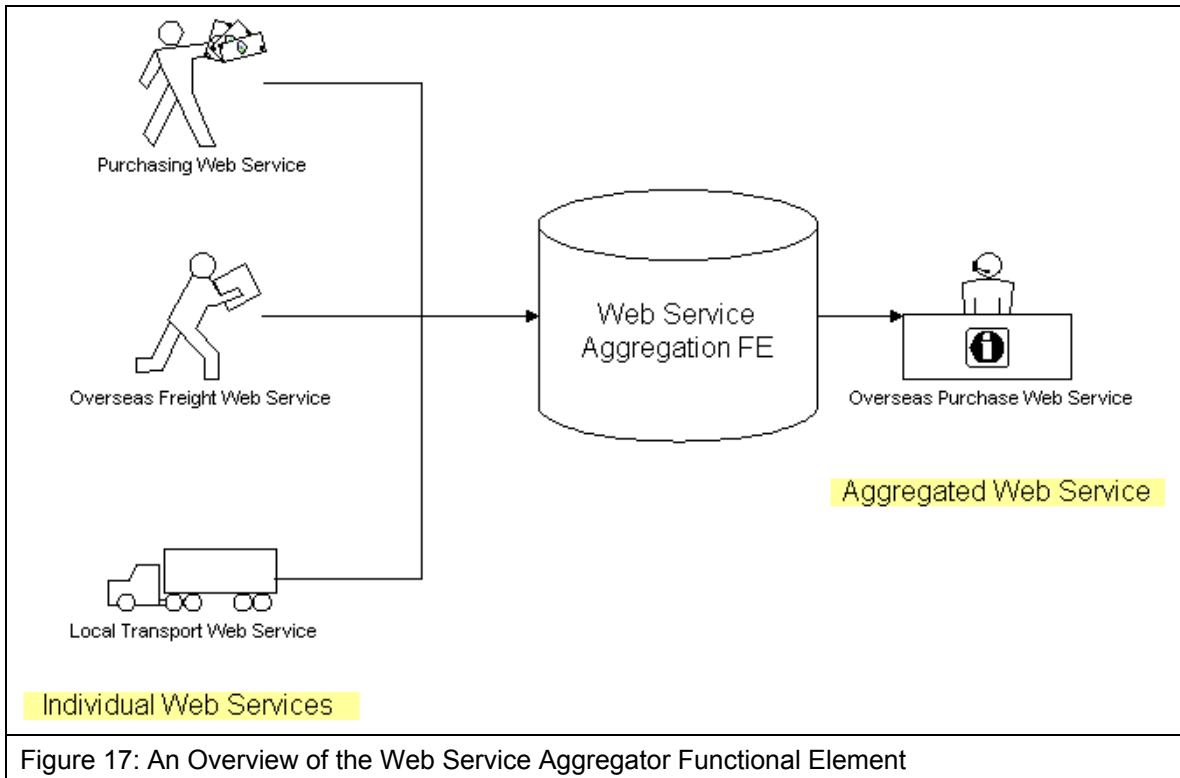
Terms	Description
Aggregated Web Service	Aggregated Web Service is single Web Services that invoke multiple Web Services to realize its functionality.
Composition Rule	A Composition Rule is an expression specifying how individual Web Services are invoked to form aggregated Web Services. It includes the name of Web Services that are included in aggregation, specification of aggregation sequence, data dependency among the individual Web Services.

3624

3625 The following diagram shows the meaning of the terms in the context of Web Services  
3626 aggregation.

3627

3628



3629

### 3630 2.16.3 Key Features

3631 Implementations of the Web Service Aggregator Functional Element are expected to provide the  
3632 following key features:

- 3633 1. The Functional Element MUST provide a mechanism for composing any number of Web  
3634 Services into single Web Service according to specified Composition Rule(s).
- 3635 2. Individual web services can reside at any location, but it is expected to be accessible.
- 3636 3. As part of Key Feature (1), the WSDL of a web service used for composition MUST be  
3637 available.
- 3638 4. The Functional Element MUST support the definition, modification and removal of  
3639 Composition Rules.
- 3640 5. The Functional Element MUST encapsulate the composition logic used into an interpretable  
3641 XML-based script based on a particular standard\*.

3642 *Example: BPEL or WSCI. The TC will have to decide on which standard to use*

3643

3644 In addition, the following key features could be provided to enhance the Functional Element  
3645 further:

- 3646 1. The Functional Element MAY provide the capability to transform the interpretable XML-based  
3647 script into an executable program.
- 3648 2. If Key Feature (1) is provided, then the Functional Element MAY also have the following  
3649 capabilities:

- 3650 2.1 The ability to test the functionality of the aggregated Web Service,
- 3651 2.2 A WSDL to describe the aggregated Web Service, and
- 3652 2.3 The capability to publish the aggregated Web Service into an UDDI-compliant registry

3653 **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

3654

3655 **2.16.5 Related Technologies and Standards**

3656 *\* The interpretable XML based script should follow the standard identified by TC.*

3657 **2.16.6 Model**

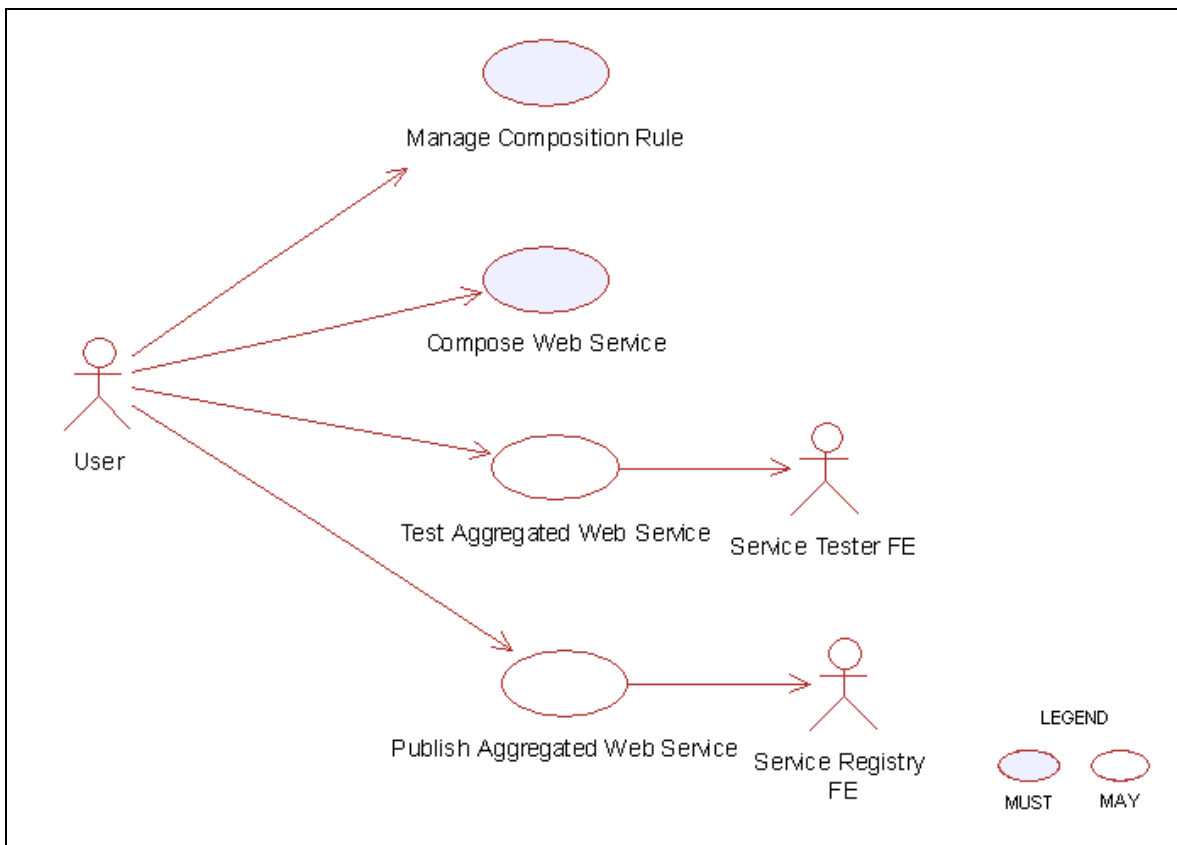


Figure 18: Model Of the Web Service Aggregation Functional Element [31]

3658

3659 **2.16.7 Usage Scenarios**

3660 **2.16.7.1 Manage composition rule**

3661 **2.16.7.1.1 Description**

3662 This use case allows the user to manage the composition rule used for Web Services  
3663 aggregation.

3664 **2.16.7.1.2 Flow of Events**

3665 **2.16.7.1.2.1 Basic Flow**

3666 The use case begins when the user wants to manage a composition rule.

3667 1: The user sends a request to the Functional Element together with the composition rule and  
3668 operation.

3669 2: Based on the operation it specified, one of the following sub-flows is executed:

- 3670
- If the operation is '**Define a rule**', then sub-flow 2.1 is executed.
  - 3671 • If the operation is '**Update a rule**', then sub-flow 2.2 is executed.
  - 3672 • If the operation is '**Retrieve a rule**', then sub-flow 2.3 is executed.
  - 3673 • If the operation is '**Remove a rule**', then sub-flow 2.4 is executed.

3674 2.1: Define Rule.

3675 2.1.1: The Functional Element gets the composition rule, i.e. names of all Web Service,  
3676 the sequence specification, parameters mapping between Web Services.

3677 2.1.2: The Functional Element verifies the correctness of composition rule.

3678 2.1.3: The Functional Element saves the composition rule to persistent mechanism.

3679 2.2: Update Rule.

3680 2.2.1: The Functional Element gets the name of composition rule.

3681 2.2.2: The Functional Element retrieves the composition rule definition from persistent  
3682 mechanism.

3683 2.2.3: The Functional Element verifies the correctness of composition rule.

- 3684 2.2.4: The Functional Element updates the composition rule.
- 3685 2.3: Retrieve Rule.
- 3686 2.3.1: The Functional Element gets the name of composition rule.
- 3687 2.3.2: The Functional Element retrieves the definition of composition rule.
- 3688 2.3.3: The Functional Element returns the definition of rule.
- 3689 2.4: Remove Rule.
- 3690 2.4.1: The Functional Element gets the name of composition rule.
- 3691 2.4.2: The Functional Element checks whether the rule exists.
- 3692 2.4.3: The Functional Element removes the rule.
- 3693 3: The Functional Element returns the results to indicate the success or failure of this operation to  
3694 the user and the use case ends.
- 3695 **2.16.7.1.2.2 Alternative Flows**
- 3696 1: Composition Rule Already Created.
- 3697 1.1: If in the basic flow 2.1.2, the same rule already created, Functional Element will return an  
3698 error message to the user and the use case ends.
- 3699 2: Composition Rule Not Exist.
- 3700 2.1: If in the basic flow 2.2, 2.3, and 2.4 the specified rule does not exist, Functional Element  
3701 will return an error message to the user and the use case ends.
- 3702 3: Persistency Mechanism Error.
- 3703 3.1: If in the basic flow 2.1, 2.2, 2.3, and 2.4, the Functional Element cannot perform data  
3704 persistency, Functional Element will return an error message to the user and the use case  
3705 ends.
- 3706 **2.16.7.1.3 Special Requirements**
- 3707 None.

3708 **2.16.7.1.4 Pre-Conditions**

3709 None.

3710 **2.16.7.1.5 Post-Conditions**

3711 None.

3712 **2.16.7.2 Compose Web Services**

3713 **2.16.7.2.1 Description**

3714 This use case will allow users to aggregate several simpler services into a higher-level service.

3715 **2.16.7.2.2 Flow of Events**

3716 **2.16.7.2.2.1 Basic Flow**

3717 This use case begins when any user wants to compose a Web Service.

3718 1: The user passes in a list of parameters for composition, including URLs of the WSDL,  
3719 composition rules.

3720 2: Functional Element checks the signature of the Web Services to be composed via accessing  
3721 WSDL.

3722 3: Functional Element generates interpretable XML-based script to encapsulate the composition  
3723 logic.

3724 4: Functional Element returns the generated script and the use case ends.

3725 **2.16.7.2.2.2 Alternative Flows**

3726 1: Functional Element generates executable program and WSDL.

3727 1.1: At basic flow 3, Functional Element may transform the interpretable XML-based script  
3728 into an executable program, if the user requested.

3729 1.2: At basic flow 3, Functional Element may generate WSDL for the executable program, if  
3730 the user requested.

3731 1.3: Functional Element returns the code of executable program and WSDL file

3732 2: Functional Element detects ambiguity in Web Services signature.

3733 2.1: At basic flow 2, Functional Element encounters an ambiguity in the Web Services  
3734 signature which it cannot resolve.

3735 2.2: Functional Element returns an error message that there is a composition error.

3736 3: Functional Element detects error in Web Services composition.

3737 3.1: At basic flow 3, Functional Element encounters an error in the Web Services  
3738 composition.

3739 3.2: Functional Element returns an error message that there is a composition error.

### 3740 **2.16.7.2.3 Special Requirements**

3741 None.

### 3742 **2.16.7.2.4 Pre-Conditions**

3743 The composition rule for this Web Services aggregation must be pre-defined.

### 3744 **2.16.7.2.5 Post-Conditions**

3745 The generated program is ready for deployment in any Web Services container.

3746

## 3747 **2.16.7.3 Test Aggregated Web Services**

### 3748 **2.16.7.3.1 Description**

3749 This use case will allow users to test the functionality of aggregate web service.

### 3750 **2.16.7.3.2 Flow of Events**

#### 3751 **2.16.7.3.2.1 Basic Flow**

3752 This use case begins when any user wants to test aggregated web service.

3753 1: The user passes in a list of parameters for testing, including URLs of the WSDL, values of  
3754 parameters for invocation.

3755 2: Functional Element invokes the aggregated web service with parameters.

3756 3: Functional Element compares the returned parameter with the expected values.

3757 4: Functional Element returns the result of comparison and the use case ends.

3758 **2.16.7.3.2.2 Alternative Flows**

3759 1: Functional Element cannot invoke the aggregated web service.

3760 1.1: At basic flow 2, Functional Element encounters problems of invoking the aggregated web  
3761 services.

3762 1.2: Functional Element returns an error message that indicates the invocation error.

3763 **2.16.7.3.3 Special Requirements**

3764 None.

3765 **2.16.7.3.4 Pre-Conditions**

3766 The executable program must be generated and deployed in web services hosting environment  
3767 and ready for invocation.

3768 **2.16.7.3.5 Post-Conditions**

3769 None.

3770 **2.16.7.4 Publish Aggregated Web Services**

3771 **2.16.7.4.1 Description**

3772 This use case will allow users to publish the aggregated web services into UDDI registry.

3773 **2.16.7.4.2 Flow of Events**

3774 **2.16.7.4.2.1 Basic Flow**

3775 This use case begins when any user wants to publish the aggregated web services into UDDI  
3776 registry.

3777 1: The user passes in a list of parameters for publishing, including URLs of the WSDL of  
3778 aggregated web services, URL of UDDI and parameters of business and services description.

3779 2: Functional Element checks the availability of UDDI.

3780 3: Functional Element publishes services description of aggregated web services into UUDI.

3781 4: Functional Element returns the publish result and the use case ends.

3782 **2.16.7.4.2.2 Alternative Flows**

3783 1: UDDI registry server is not available

3784 1.1: At basic flow 2, Functional Element cannot connect to UDDI registry if UDDI registry  
3785 server is not available.

3786 1.2: Functional Element returns the error message that UDDI connection cannot be built.

3787 2: Functional Element detects error in Web Services publishing.

3788 2.1: At basic flow 3, Functional Element encounters an error in the publishing Web Services.

3789 2.2: Functional Element returns an error message that there is a publishing error.

3790 **2.16.7.4.3 Special Requirements**

3791 None.

3792 **2.16.7.4.4 Pre-Conditions**

3793 The WSDL of the aggregated web services must exist.

3794 **2.16.7.4.5 Post-Conditions**

3795 None

3796

---

## 3 Functional Elements Usage Scenario

3797

3798

3799

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

3800

- A service monitoring solution for the management of services in a SOA model

3801

- Enabling security through the Secure SOAP Functional Element

3802

3803

- Decoupled User Access Management with support for multi-domain capabilities in a web service environment

3804

3805

3806 **3.1 Service Monitoring**

3807 In a SOA environment, management of services includes the capability to monitor services within  
3808 the management domain. These includes:

- 3809 • Monitoring the performance of services invoked
- 3810 • Generating audit trails of services invoked
- 3811 • Monitoring and testing the availability of services on the remote machine (server)

3812 A basic solution can be realised through the aggregation of two Functional Element, namely  
3813 Service Management and Service Tester, as shown in Figure 19. This solution can be improved  
3814 with notification capabilities, using the Notification Engine, be it to a remote client, a system  
3815 administrator or an end user of a particular service. Further enhancement can be added with a  
3816 Rule Engine that will have the cognitive ability to make decisions. An example of this  
3817 enhancement would be the ability to decide when should notifications or alerts be sent and in  
3818 what form.

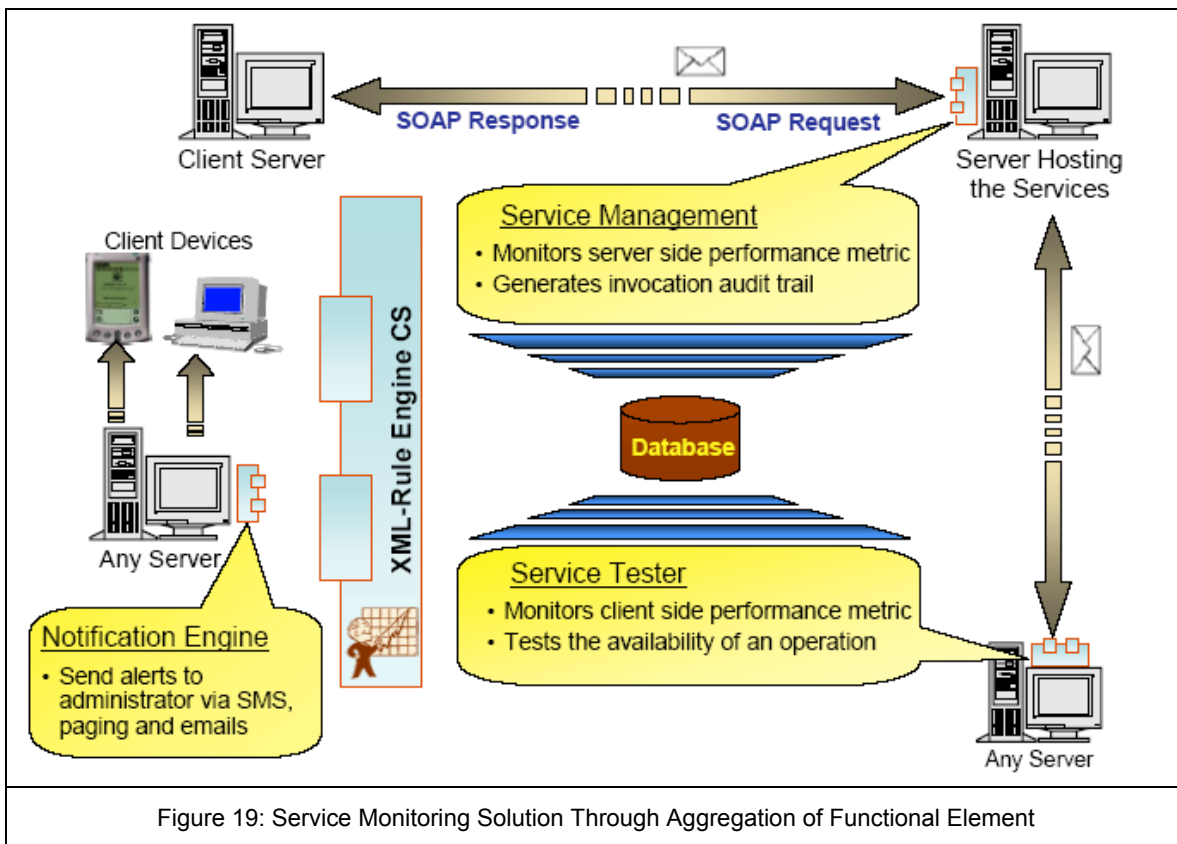
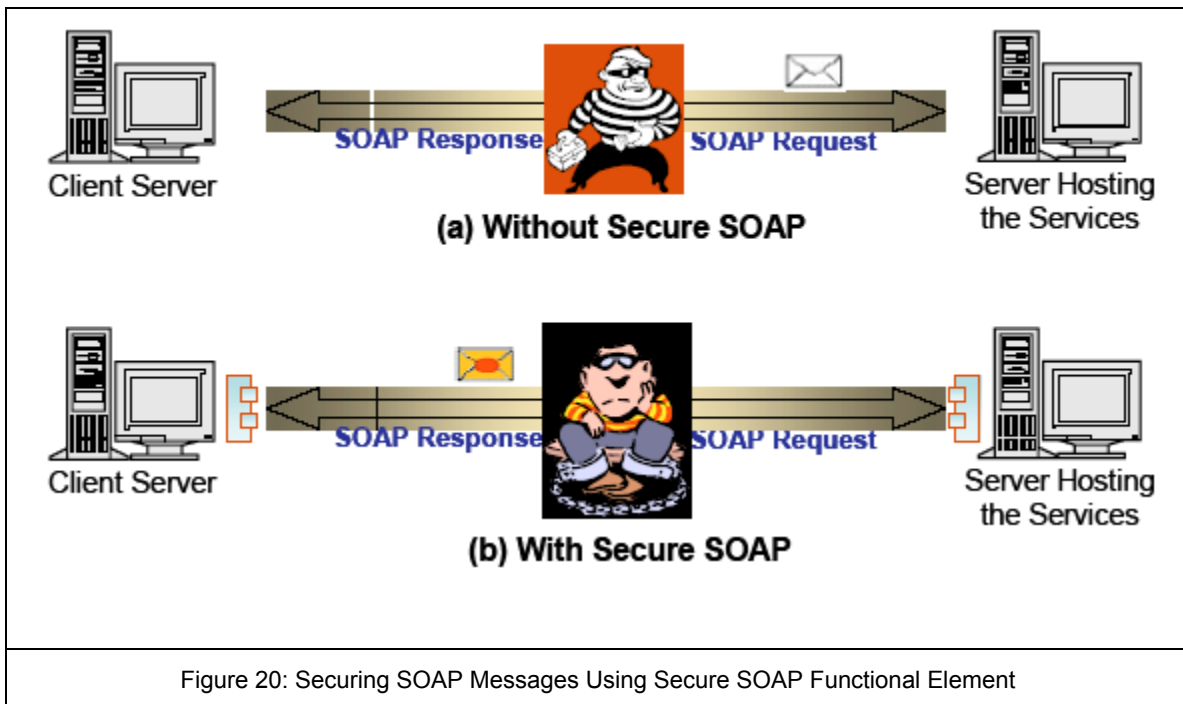


Figure 19: Service Monitoring Solution Through Aggregation of Functional Element

3819

3820 **3.2 Securing SOAP Messages**

3821 SOAP in its pure form does not have any built in security as it is meant to be a simple and  
3822 lightweight protocol. As such, where security is needed, additional capabilities must be provided.  
3823 Presently, standards like XML Encryption and XML Signature are available. Making use of these  
3824 standards, the Secure Soap Functional Element, when deployed on both the sending and  
3825 receiving parties, will be able to provide encryption and signing of messages as illustrated in  
3826 Figure 20.  
3827



3828

### 3829 **3.3 Decouple d User Access Management**

3830 User Access Management (UAM) has been implemented in many forms and in a wide variety of  
3831 ways, from the most basic to the most complex. At the most simple form, the functionality would  
3832 include username and password support. On the end of the scale, it would include functionalities  
3833 like distributed access management, replication capabilities and fine-grain controls just to name a  
3834 few.

3835 In this specification, the goal is to provide a set of Functional Element that can be used as  
3836 building blocks for UAM, and can be extended when the need arises. It is provided as a  
3837 decoupled building blocks consisting of four Functional Elements, namely User Management,  
3838 Group Management, Role & Access Management and Phase & Lifecycle Management, as  
3839 illustrated in Figure 21. These Functional Elements can be used in a variety of combinatorial  
3840 forms, and some of these examples include:

- 3841 • User Management only, or
- 3842 • User Management and Group Management, or
- 3843 • User Management and Role & Access Management, or
- 3844 • User Management, Group Management and Role & Access Management, or
- 3845 • All the four Functional Elements in tandem

3846 On the same token, any of the Functional Elements can be replaced with similar functionality third  
3847 party web services. As these services are designed to be in a web service environment, each of  
3848 them also supports the concept of namespace. This namespace provision enables each of the  
3849 Functional Elements to be used as web services that can be accessed by multiple organisations  
3850 or to cater for users from different domains. With this, access control for example, can be defined  
3851 for multiple domains without corruption or interferences problems.

3852

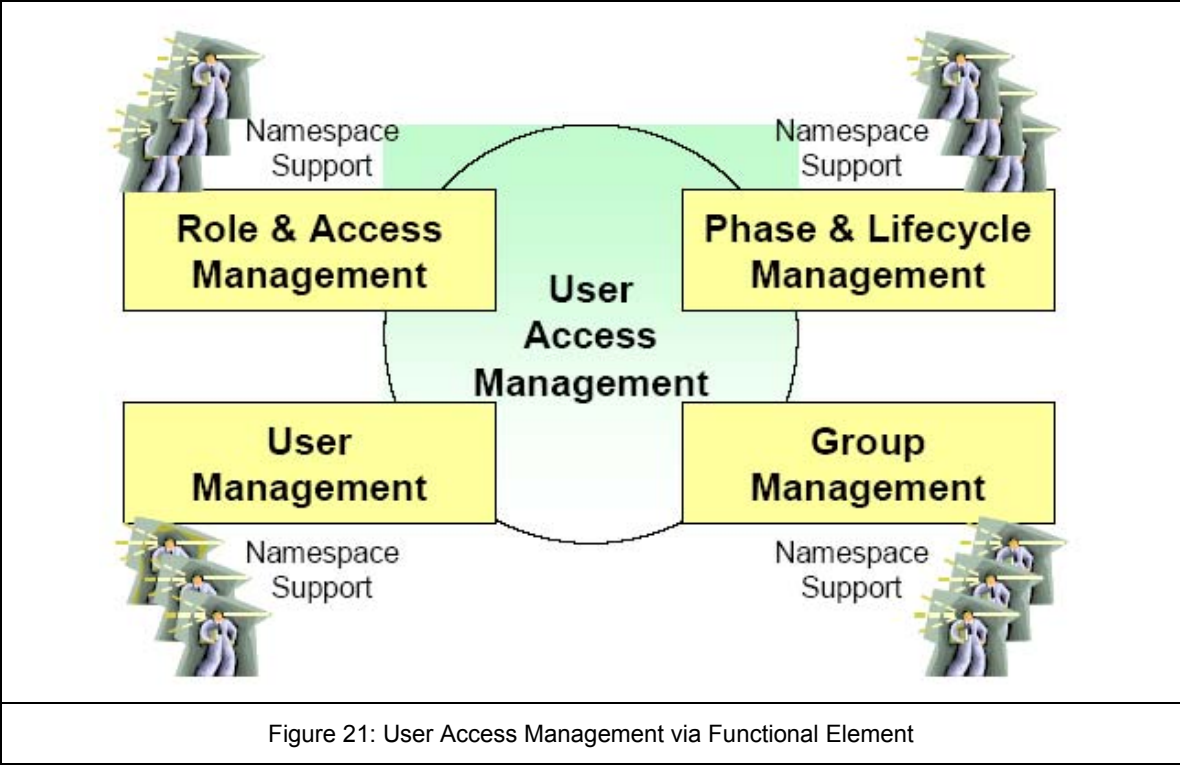


Figure 21: User Access Management via Functional Element

---

## 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](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](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.

3854

---

## Appendix A. Acknowledgments

3855

Special thanks to the following individuals who contributed significantly towards to the initial draft of this work during the development of this specification:

3856

3857

- Chan Lai Peng, Singapore Institute of Manufacturing Technology

3858

- Cheng Yushi, Singapore Institute of Manufacturing Technology

3859

- Dilip Kumar Limbu, Singapore Institute of Manufacturing Technology

3860

- V. Ramasamy, Singapore Institute of Manufacturing Technology

3861

- Wu Yingzi, Singapore Institute of Manufacturing Technology

3862

- Xu Xingjian, Singapore Institute of Manufacturing Technology

3863

- Yin Zunliang, Singapore Institute of Manufacturing Technology

3864

3865

The committee would also like to express its appreciation for the encouragement and guidance provided by Jamie Clark throughout the course of the TC work.

3866

3867

3868

The committee would also like to record its heartfelt appreciation to IBM Rational (Singapore) Pte. Ltd. for kindly agreeing to allow the use of the Rational Tools towards the creation of the Use Case Model used in this document.

3869

3870

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
fws-fe-1.0-guidelines-spec-wd-03.doc	25-November-2004	Huang Kheng Cheng	Second Draft (Voted version)
fws-fe-1.0-guidelines-spec-cs-01.doc	04-March-2005	Puay Siew Tan	Update the document to reflect its change of status to a Committee Specs (as of 16 Dec 2004)
fws-fe-1.0-guidelines-spec-cs-02.doc	27-May-2005	Puay Siew Tan	Update the document on syntactical errors. Features are not changed.

3878

3879

---

## 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.