



Functional Elements Specification

Working Draft 2.0, 28-October-2005

Document identifier:

fws-fe-2.0-guidelines-spec-wd-01.doc

Location:

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

Editor:

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

Contributor(s):

Andy Tan, Individual (andytan@intrinix.net)

Shawn Cheng Hua-Shan, XMLBoss (shawn@xmlboss.net)

Kenneth Lim, Crimson Logic Pte Ltd (kennethlim@crimsonlogic.com)

Ravi Shankar, Crimson Logic Pte Ltd (ravishankar@crimsonlogic.com)

Jagdip Talla, Crimson Logic Pte Ltd (jagdip@crimsonlogic.com)

Roberto Pascual, Infocomm Development Authority (IDA) of Singapore
(rbpascual@yahoo.com)

Lee Eng Wah, SIMTech (ewlee@simtech.a-star.edu.sg)

V.Ramasamy, SIMTech (rama@simtech.a-star.edu.sg)

Lee Siew Poh, SIMTech (splee@simtech.a-star.edu.sg)

Lee Ah Kim, SIMTech (aklee@simtech.a-star.edu.sg)

Abstract:

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

This document specifies a set of Functional Elements for practitioners to instantiate into a technical architecture, and should be read in conjunction with the Functional Elements

35 Requirements document. It is the purpose of this specification to define the right level of
36 abstraction for these Functional Elements and to specify the purpose and scope of each
37 Functional Element so as to facilitate efficient and effective implementation of Web
38 Services.

39

40 **Status:**

41 This document is updated periodically on no particular schedule.

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

47 For information on whether any patents¹ have been disclosed that may be essential to
48 implementing this specification, and any offers of patent licensing terms, please refer to
49 the Intellectual Property Rights section of the FWSI TC web page ([http://www.oasis-
50 open.org/committees/fwsi/](http://www.oasis-open.org/committees/fwsi/)).

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

51 **Table of Contents**

52 1 Introduction..... 8

53 1.1 Document Outline 8

54 1.2 Motivation..... 9

55 1.3 Terminology 9

56 2 List of Functional Elements 10

57 2.1 Data Integrator (new) 10

58 2.1.1 Motivation 10

59 2.1.2 Terms Used 10

60 2.1.3 Key Features 12

61 2.1.4 Interdependencies 12

62 2.1.5 Related Technologies and Standards 13

63 2.1.6 Model..... 13

64 2.1.7 Usage Scenarios 14

65 2.2 Error Management Functional Element (new) 27

66 2.2.1 Motivation 27

67 2.2.2 Terms Used 28

68 2.2.3 Key Features 29

69 2.2.4 Interdependencies 30

70 2.2.5 Related Technologies and Standards 30

71 2.2.6 Model..... 31

72 2.2.7 Usage Scenarios 31

73 2.3 Event Handler Functional Element 43

74 2.3.1 Motivation 43

75 2.3.2 Terms Used 43

76 2.3.3 Key Features 44

77 2.3.4 Interdependencies 46

78 2.3.5 Related Technologies and Standards 46

79 2.3.6 Model..... 47

80 2.3.7 Usage Scenarios 48

81 2.4 Group Management Functional Element 65

82 2.4.1 Motivation 65

83 2.4.2 Terms Used 65

84 2.4.3 Key Features 66

85 2.4.4 Interdependency..... 66

86 2.4.5 Related Technologies and Standards 67

87 2.4.6 Model..... 67

88 2.4.7 Usage Scenarios 67

89 2.5 Identity Management Functional Element..... 72

90 2.5.1 Motivation 72

91 2.5.2 Terms Used 72

92 2.5.3 Key Features 74

93 2.5.4 Interdependencies 74

94 2.5.5 Related Technologies and Standards 75

95 2.5.6 Model..... 76

96 2.5.7 Usage Scenarios 77

97	2.6	<i>Information Catalogue Functional Element (new)</i>	81
98	2.6.1	<i>Motivation</i>	81
99	2.6.2	<i>Terms Used</i>	81
100	2.6.3	<i>Key Features</i>	81
101	2.6.4	<i>Interdependencies</i>	82
102	2.6.5	<i>Related Technologies and Standards</i>	82
103	2.6.6	<i>Model</i>	83
104	2.6.7	<i>Usage Scenario</i>	83
105	2.7	<i>Information Reporting Functional Element (new)</i>	90
106	2.7.1	<i>Motivation</i>	90
107	2.7.2	<i>Terms Used</i>	91
108	2.7.3	<i>Key Features</i>	91
109	2.7.4	<i>Interdependencies</i>	91
110	2.7.5	<i>Related Technologies and Standards</i>	92
111	2.7.6	<i>Model</i>	92
112	2.7.7	<i>Usage Scenario</i>	92
113	2.8	<i>Key Management Functional Element (new)</i>	104
114	2.8.1	<i>Motivation</i>	104
115	2.8.2	<i>Terms Used</i>	104
116	2.8.3	<i>Key Features</i>	104
117	2.8.4	<i>Interdependencies</i>	105
118	2.8.5	<i>Related Technologies and Standards</i>	105
119	2.8.6	<i>Model</i>	106
120	2.8.7	<i>Usage Scenarios</i>	107
121	2.9	<i>Log Utility Functional Element</i>	113
122	2.9.1	<i>Motivation</i>	113
123	2.9.2	<i>Terms Used</i>	113
124	2.9.3	<i>Key Features</i>	113
125	2.9.4	<i>Interdependencies</i>	114
126	2.9.5	<i>Related Technologies and Standards</i>	114
127	2.9.6	<i>Model</i>	115
128	2.9.7	<i>Usage Scenarios</i>	115
129	2.10	<i>Notification Functional Element</i>	122
130	2.10.1	<i>Motivation</i>	122
131	2.10.2	<i>Terms Used</i>	122
132	2.10.3	<i>Key Features</i>	123
133	2.10.4	<i>Interdependencies</i>	123
134	2.10.5	<i>Related Technologies and Standards</i>	123
135	2.10.6	<i>Model</i>	124
136	2.10.7	<i>Usage Scenarios</i>	124
137	2.11	<i>Phase and Lifecycle Management Functional Element</i>	130
138	2.11.1	<i>Motivation</i>	130
139	2.11.2	<i>Terms Used</i>	130
140	2.11.3	<i>Key Features</i>	131
141	2.11.4	<i>Interdependencies</i>	131
142	2.11.5	<i>Related Technologies and Standards</i>	131
143	2.11.6	<i>Model</i>	131
144	2.11.7	<i>Usage Scenarios</i>	132
145	2.12	<i>Policy Management Functional Element (new)</i>	137
146	2.12.1	<i>Motivation</i>	137

147	2.12.2	Terms Used.....	137
148	2.12.3	Key Features.....	138
149	2.12.4	Interdependency	139
150	2.12.5	Related Technologies and Standards	139
151	2.12.6	Model	140
152	2.12.7	Usage Scenarios.....	140
153	2.13	Policy Enforcement Functional Element (new).....	145
154	2.13.1	Motivation.....	145
155	2.13.2	Terms Used.....	145
156	2.13.3	Key Features.....	145
157	2.13.4	Interdependency	146
158	2.13.5	Related Technologies and Standards	146
159	2.13.6	Model	146
160	2.13.7	Usage Scenarios.....	147
161	2.14	Presentation Transformer Functional Element	149
162	2.15	Quality of Service (QoS) Functional Element (new)	150
163	2.15.1	Motivation.....	150
164	2.15.2	Terms Used.....	150
165	2.15.3	Key Features.....	151
166	2.15.4	Interdependencies.....	151
167	2.15.5	Related Technologies and Standards	152
168	2.15.6	Model	152
169	2.15.7	Usage Scenarios.....	153
170	2.16	Role and Access Management Functional Element	162
171	2.16.1	Motivation.....	162
172	2.16.2	Terms Used.....	162
173	2.16.3	Key Features.....	163
174	2.16.4	Interdependencies.....	164
175	2.16.5	Related Technologies and Standards	164
176	2.16.6	Model	165
177	2.16.7	Usage Scenario.....	165
178	2.17	Search Functional Element.....	175
179	2.17.1	Motivation.....	175
180	2.17.2	Terms Used.....	175
181	2.17.3	Key Features.....	176
182	2.17.4	Interdependencies.....	176
183	2.17.5	Related Technologies and Standards	176
184	2.17.6	Model	177
185	2.17.7	Usage Scenario.....	177
186	2.18	Secure SOAP Management Functional Element.....	181
187	2.18.1	Motivation.....	181
188	2.18.2	Terms Used.....	181
189	2.18.3	Key Features.....	182
190	2.18.4	Interdependencies.....	182
191	2.18.5	Related Technologies and Standards	182
192	2.18.6	Model	183
193	2.18.7	Usage Scenarios.....	183
194	2.19	Sensory Functional Element.....	187
195	2.19.1	Motivation.....	187
196	2.19.2	Terms Used.....	187

197	2.19.3	Key Features.....	187
198	2.19.4	Interdependencies.....	188
199	2.19.5	Related Technologies and Standards.....	188
200	2.19.6	Model.....	188
201	2.19.7	Usage Scenarios.....	188
202	2.20	Service Level Management Functional Element (new).....	191
203	2.20.1	Motivation.....	191
204	2.20.2	Terms Used.....	191
205	2.20.3	Key Features.....	191
206	2.20.4	Interdependencies.....	192
207	2.20.5	Related Technologies and Standards.....	192
208	2.20.6	Model.....	192
209	2.20.7	Usage Scenarios.....	193
210	2.21	Service Level Enforcement Functional Element (new).....	200
211	2.21.1	Motivation.....	200
212	2.21.2	Terms Used.....	200
213	2.21.3	Key Features.....	200
214	2.21.4	Interdependencies.....	200
215	2.21.5	Related Technologies and Standards.....	201
216	2.21.6	Model.....	201
217	2.21.7	Usage Scenarios.....	201
218	2.22	Service Management Functional Element.....	206
219	2.22.1	Motivation.....	206
220	2.22.2	Terms Used.....	206
221	2.22.3	Key Features.....	206
222	2.22.4	Interdependencies.....	207
223	2.22.5	Related Technologies and Standards.....	207
224	2.22.6	Model.....	208
225	2.22.7	Usage Scenarios.....	208
226	2.23	Service Registry Functional Element.....	214
227	2.23.1	Motivation.....	214
228	2.23.2	Terms Used.....	214
229	2.23.3	Key Features.....	215
230	2.23.4	Interdependencies.....	215
231	2.23.5	Related Technologies and Standards.....	215
232	2.23.6	Model.....	216
233	2.23.7	Usage Scenario.....	216
234	2.24	Service Router Functional Element (new).....	225
235	2.24.1	Motivation.....	225
236	2.24.2	Terms Used.....	225
237	2.24.3	Key Features.....	226
238	2.24.4	Interdependencies.....	227
239	2.24.5	Related Technologies and Standards.....	227
240	2.24.6	Model.....	228
241	2.24.7	Usage Scenarios.....	228
242	2.25	Service Tester Functional Element.....	235
243	2.26	Transformer Functional Element (new).....	236
244	2.26.1	Motivation.....	236
245	2.26.2	Terms Used.....	236
246	2.26.3	Key Features.....	237

247	2.26.4	<i>Interdependencies</i>	237
248	2.26.5	<i>Related Technologies and Standards</i>	238
249	2.26.6	<i>Model</i>	238
250	2.26.7	<i>Usage Scenarios</i>	238
251	2.27	<i>User Management Functional Element</i>	244
252	2.27.1	<i>Motivation</i>	244
253	2.27.2	<i>Terms Used</i>	244
254	2.27.3	<i>Key Features</i>	245
255	2.27.4	<i>Interdependencies</i>	246
256	2.27.5	<i>Related Technologies and Standards</i>	246
257	2.27.6	<i>Model</i>	246
258	2.27.7	<i>Usage Scenarios</i>	247
259	2.28	<i>Web Service Aggregator Functional Element</i>	254
260	2.28.1	<i>Motivation</i>	254
261	2.28.2	<i>Terms Used</i>	254
262	2.28.3	<i>Key Features</i>	255
263	2.28.4	<i>Interdependencies</i>	256
264	2.28.5	<i>Related Technologies and Standards</i>	256
265	2.28.6	<i>Model</i>	256
266	2.28.7	<i>Usage Scenarios</i>	257
267	3	<i>Functional Elements Usage Scenarios</i>	262
268	3.1	<i>Service Monitoring</i>	263
269	3.2	<i>Securing SOAP Messages</i>	264
270	3.3	<i>Decoupled User Access Management</i>	265
271	3.4	<i>Single-Sign-On for Distributed Services (Applications)</i>	266
272	4	<i>References</i>	267
273		<i>Appendix A. Acknowledgments</i>	269
274		<i>Appendix B. Revision History</i>	270
275		<i>Appendix C. Notices</i>	271
276			
277			

278

1 Introduction

279

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

288

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

300

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

309

310

1.1 Document Outline

311

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

316

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

320

- Motivation

321

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

- 324 • Terms Used
325 A glossary of the terms used. An explanation or illustration of the runtime capabilities of the
326 Functional Element are also provided where appropriate.
- 327 • Key Features
328 A list of key features to be implemented is provided here and is expressed in the normative
329 form.
- 330 • Interdependencies
331 In this section, the interdependencies between Functional Elements are provided to clarify
332 the linkages between FEs (if any).
- 333 • Related Technologies and Standards
334 Here, the reliance of the Functional Elements on related technologies and specifications (or
335 standards) are provided
336
- 337 Section 3 provides the examples of how the Functional Elements can be assembled to accelerate
338 web service-enabled applications. From these Functional Elements, a variety of solutions can be
339 built.
340

341 **1.2 Motivation**

342
343 In a Service-Oriented Architecture (SOA) environment, new applications/services are created
344 through the assembly of existing services. One of the key advantages of this loosely coupled
345 model is that it allows the new application/service to leverage on 3rd party services. As a typical
346 3rd party's implementation of the services is done via the software component approach, this
347 specification further proliferate new applications/services by defining a framework for Web
348 Services implementation consisting of Functional Elements. Through these Functional Elements,
349 which are implementation neutral, this Specification hopes to influence future software
350 development towards assembly of services rather than 'pure built only'.

351 **1.3 Terminology**

352
353 Within this document the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL
354 NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
355 document are to be interpreted as described in RFC2119 [3].
356

357 Cross-references to the Functional Elements Requirements document [2] are designated
358 throughout this specification to the requirement contained where the requirement number is
359 enclosed in square brackets (e.g. **[MANAGEMENT-005]**).
360
361

2 List of Functional Elements

362

363

2.1 Data Integrator (new)

364

2.1.1 Motivation

365

366 The Data Integrator Functional element is expected to be used for enabling easy and simple
367 mechanisms to access disparate data sources by:

- 368 • Providing unified data view of enterprise across various data sources,
- 369 • Enabling the partitioned view of data for different groups/departments based on defined
370 logical views, and
- 371 • Performing data processing or transformation before presenting the defined logical data
372 view(s).

373

374 This Functional Element fulfills the following requirements from the Functional Elements
375 Requirements Document 02:

- 376 • Primary Requirements
 - 377 ○ PROCESS-220 to PROCESS-236.
- 378 • Secondary Requirements
 - 379 ○ None

380

2.1.2 Terms Used

381

Terms	Description
Batch Retrieval Definition	Batch retrieval definition defines how batch data retrieval is performed. The definition of batch retrieval would include the XML schema for the XML format of retrieved data, the mapping of the data fields in the format to the data fields in the logical data view and the schedule of batch retrieval
Data Repository	Data repository is a form of persistent data storage used by Data Integrator to store information of logical data views information.
Data Source	Data source is physical data storage where data can be retrieved. It may include relational database, XML database, LDAP, text file, XML file, URL that pointing to a set of data in Internet.

<p>Data Transformation Rule</p>	<p>Data transformation rule defines how raw data is transformed into the data format that is requested by final presentation. Data transformation rule has two types.</p> <p>–The first type is the one that applies at the logical data view level and generates instances of data for the whole data view.</p> <p>» An example of this type rule could be a name of the pre-defined function that gets data instances from various data sources and fills in the data view.</p> <p>–The second type is the one that applies at the data field level of the logical data view and only generates the data for that particular data field.</p> <p>» An examples of this type rule could be a formula like: data field 1 in logical data view = data field 1 in data source 1 X data field 2 in data source 2 .</p>
<p>Logical Data View</p>	<p>Logical data view is a conceptual/semantic data model. It is defined by the name of logical data view, owner, created date, the data fields, the sources of data fields, the constraints of data view, and the transformation rule associated.</p>

382

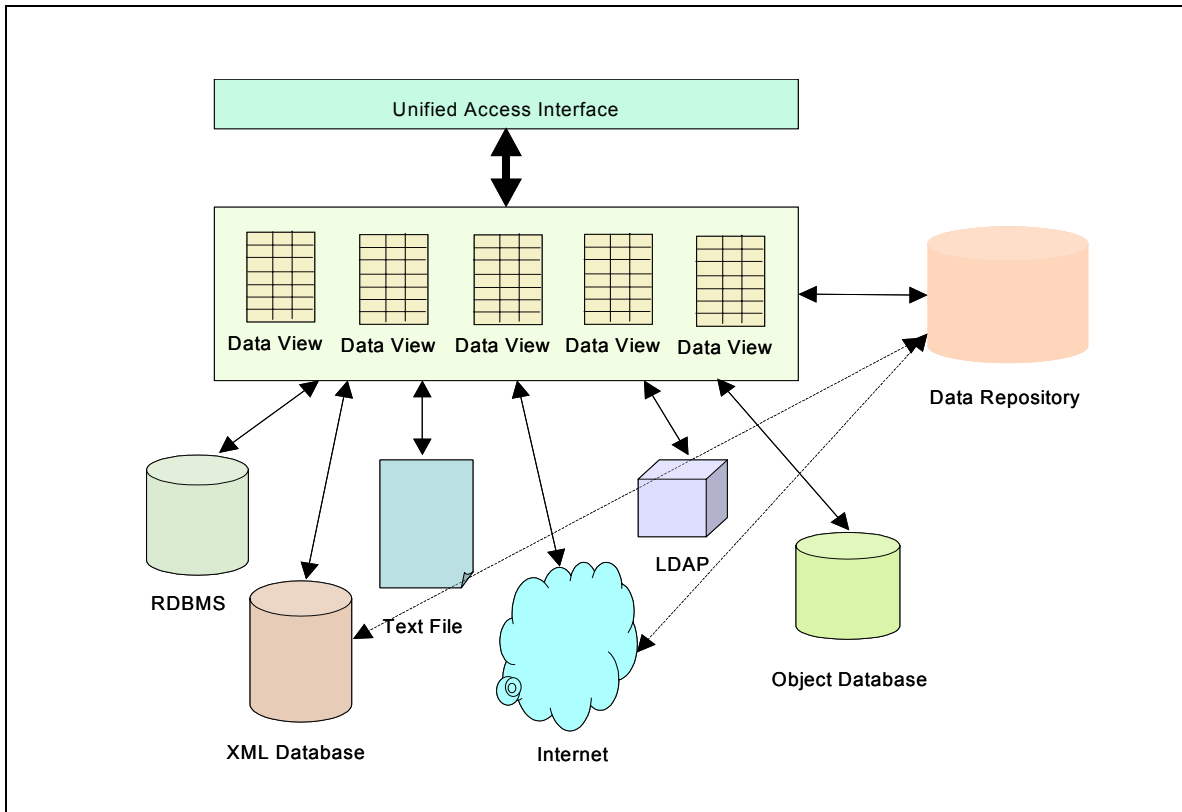


Figure 1: An Overview of the Data Integrator Functional Element

383

384 Figure 1 depicts the basic concepts of how the participating entities collaborate together in the
385 Data Integrator Functional Element. Data can be physically scattered across various data
386 sources, residing on the local area network (LAN) or over Wide Area Network (WAN). Examples
387 include RDBMS, XML database, XML files, URLs that point to a set of data in the Internet, etc.

388 Data Integrator enables the creation of different set of logical data views for various applications
389 or systems. Users of Data Integrator manipulate the data according to the logical data view
390 defined through a unified access interface. Logical data views could be physically stored in Data
391 Repository for easy and fast access.
392

393 **2.1.3 Key Features**

394 Implementations of the Data Integrator Functional Element are expected to provide the following
395 key features:

- 396 1. The Functional Element **MUST** provide the capability to manage the available data sources.
397 This includes capability to:
 - 398 1.1. Add new data source to the pool of available data sources.
 - 399 1.2. Remove data source from the pool of available data sources.
- 400 2. The Functional Element **MUST** provide the capability to define a logical data view based on
401 the pool of available data sources.
- 402 3. The Functional Element **MUST** provide capability to manage the updating and deletion of a
403 logical data view.
- 404 4. The Functional Element **MUST** provide capability to manage the creation, updating and
405 deletion of data transformation rules.
- 406 5. The Functional Element **MUST** provide capability to retrieve data based on the logical data
407 view defined.
- 408 6. The Functional Element **MUST** provide a unified way to query data based on defined logical
409 data views.
- 410 7. The Functional Element **MUST** provide a mechanism to extract data from various data
411 sources and transform the data according to defined transformation rules for a logical data
412 view.

413

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

- 416 1. The Functional Element **MAY** provide capability to insert, update and delete data based on a
417 logical data view (where applicable).
- 418 2. The Functional Element **MAY** provide the capability to retrieve batch data based on logical
419 data view according to a schedule and present the retrieved data in predefined XML formats.
- 420 3. The Functional Element **MAY** provide the capability to manage the definition of batch data
421 retrieval. This includes capability to:
 - 422 3.1 Define a batch data retrieval
 - 423 3.2 Disable the schedule of batch data retrieval
 - 424 3.3 Enable the schedule of batch data retrieval
 - 425 3.4 Remove the definition of batch data retrieval
- 426 4. 5 The Functional Element **MAY** implement data repository to host consolidated data. This
427 data repository hosts the physical entity that stores the content of a logical data view.
- 428 5. 6 The Functional Element **MAY** provide a mechanism to synchronize data between data
429 repository and data sources if data repository is provided.

430

431 **2.1.4 Interdependencies**

432 None

433

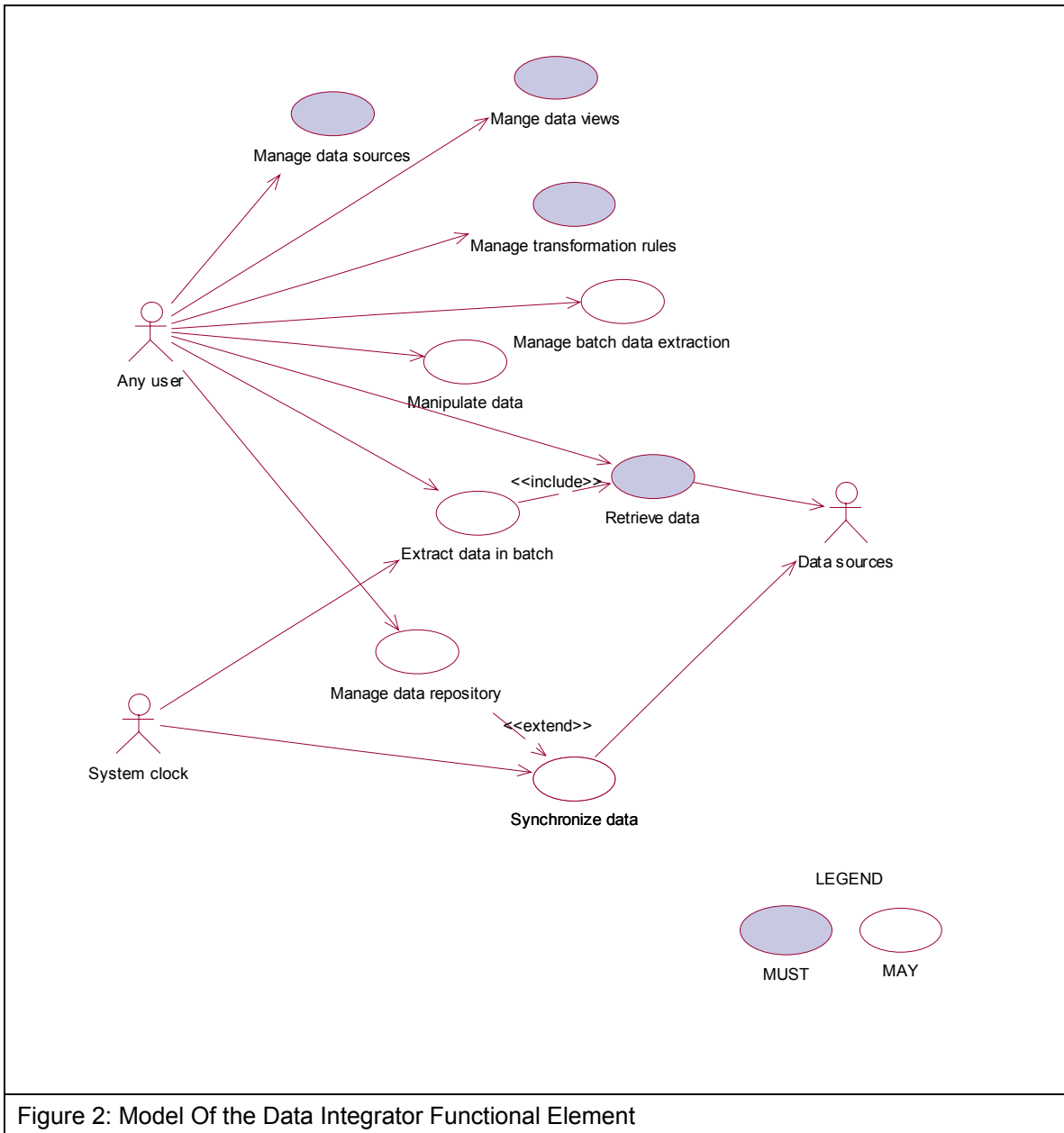
434 2.1.5 Related Technologies and Standards

435 RDBMS, LDAP, XML Database

436

437 2.1.6 Model

438



439

440 **2.1.7 Usage Scenarios**

441 **2.1.7.1 Manage data sources**

442 **2.1.7.1.1 Description**

443 This use case allows the user to manage the available data sources on which logical data views
444 are created.

445 **2.1.7.1.2 Flow of Events**

446 **2.1.7.1.2.1 Basic Flow**

447 The use case begins when the user of the Data Integrator wants to add in new data sources or
448 remove existing data sources.

449 1: The user sends a request to Data Integrator together with data source profile and operation.

450 2: Based on the operation it specified, one of the following sub-flows is executed:

451 If the operation is '**Add in data source**', then sub-flow 2.1 is executed.

452 If the operation is '**Remove data source**', then sub-flow 2.2 is executed.

453 2.1: Add in data source.

454 2.1.1: The Functional Element gets the data source profile data, i.e. name, description,
455 data source location for connection, login Id and password of the user who has privileges
456 to manipulate data sources.

457 2.1.2: The Functional Element registers the data source as available data source.

458 2.2: Remove data source.

459 2.2.1: The Functional Element gets the name of data sources

460 2.2.2: The Functional Element checks whether the data source is valid data source.

461 2.2.3: The Functional Element removes the data source from the pool of available data
462 source.

463 3: The Functional Element returns the results to indicate the success or failure of this operation to
464 the user and the use case ends.

465 **2.1.7.1.2.2 Alternative Flows**

466 1: Data Source Already Registered.

467 1.1: If in the basic flow 2.1.2, the data source is already registered, Functional Element will
468 return an error message to the user and the use case ends.

469 2: Data Source Not Exist.

470 2.1: If in the basic flow 2.2.2, the data source is not registered as available data source,
471 Functional Element will return an error message to the user and the use case ends.

472 3: Persistency Mechanism Error.

473 3.1: If in the basic flow 2.1 and 2.2, the Functional Element cannot perform data persistency,
474 Functional Element will return an error message to the user and the use case ends.

475 **2.1.7.1.3 Special Requirements**

476 None.

477 **2.1.7.1.4 Pre-Conditions**

478 None.

479 **2.1.7.1.5 Post-Conditions**

480 None.

481

482 **2.1.7.2 Manage Data Views**

483 **2.1.7.2.1 Description**

484 This use case allows the user to manage the logical data views.

485 **2.1.7.2.2 Flow of Events**

486 **2.1.7.2.2.1 Basic Flow**

487 The use case begins when the user wants to create/retrieve/update/delete a logical data view.

488 1: The user sends request to manage logical data view together with logical data view definition
489 and operation.

490 2: Based on the operation it specifies, one of the following sub-flows is executed:

491 If the operation is '**Create Data View**', the sub-flow 2.1 is executed.

492 If the operation is '**Retrieve Data View**', the sub-flow 2.2 is executed.

493 If the operation is '**Update Data View**', the sub-flow 2.3 is executed.

494 If the operation is '**Delete Data View**', the sub-flow 2.4 is executed.

495 2.1: Create Data View.

496 2.1.1: The Functional Element gets logical data view definition, i.e. name, description,
497 owner of data view, created date, data fields of data view, the source fields of data fields,
498 and transformation rule.

499 2.1.2: The Functional Element checks whether the logical data view exists.

500 2.1.3: The Functional Element creates the logical data view exists.

501 2.2: Retrieve Data View.

502 2.2.1: The Functional Element gets name of the logical data view and retrieve condition.

503 2.2.2: The Functional Element retrieves the logical data view's information according to
504 the condition.

505 2.3: Update Data View.

506 2.3.1: The Functional Element gets name of the logical data view and its definition
507 2.3.2: The Functional Element checks whether the logical data view exists.
508 2.3.3: The Functional Element updates the logical data view definition
509 2.4: Delete Data View.
510 2.4.1: The Functional Element gets name of the logical data view.
511 2.4.2: The Functional Element checks whether the logical data view exists.
512 2.4.3: The Functional Element removes the logical data view.
513 3: The Functional Element returns the results of the operation to the user and the use case ends.

514 **2.1.7.2.2 Alternative Flows**

515 1: Data View Already Exists.
516 1.1: If in the basic flow 2.1.2, the data view is already defined, Functional Element returns an
517 error message and the use case ends.
518 2: Data View Cannot Be Deleted.
519 2.1: If in the basic flow 2.4.3, the data of the logical data view is stored in Data Repository,
520 Functional Element returns an error message and the use case ends.
521 3: Data View Not Found.
522 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the data view does not exist, Functional
523 Element will return an error message and the use case ends.
524 4: Data View Cannot Be Updated.
525 4.1: If in the basic flow 2.4.3, the data of the logical data view is stored in Data Repository,
526 Functional Element returns an error message and the use case ends.
527 5: Persistency Mechanism Error.
528 5.1: If in the basic flow 2.1.2, 2.1.3, 2.2, 2.3.2, 2.3.3, 2.4.2 and 2.4.3, the Functional Element
529 cannot perform data persistency, Functional Element will return an error message to the user
530 and the use case ends.

531 **2.1.7.2.3 Special Requirements**

532 None.

533 **2.1.7.2.4 Pre-Conditions**

534 None.

535 **2.1.7.2.5 Post-Conditions**

536 None.

537

538 **2.1.7.3 Manage Transformation Rules**

539 **2.1.7.3.1 Description**

540 This use case allows the user to manage the data transformation rules that are used by the Data
541 Integrator to perform the data transformation before passing data back to users.

542 **2.1.7.3.2 Flow of Events**

543 **2.1.7.3.2.1 Basic Flow**

544 The use case begins when the user wants to create/retrieve/update/delete a data transformation
545 rule.

546 1: The user sends request to manage data transformation rule together with the definition of
547 transformation rule and operation.

548 2: Based on the operation it specifies, one of the following sub-flows is executed:

549 If the operation is '**Define Data Transformation Rule**', the sub-flow 2.1 is executed.

550 If the operation is '**Retrieve Data Transformation Rule**', the sub-flow 2.2 is executed.

551 If the operation is '**Update Data Transformation Rule**', the sub-flow 2.3 is executed.

552 If the operation is '**Delete Data Transformation Rule**', the sub-flow 2.4 is executed.

553 2.1: Create Data Transformation Rule.

554 2.1.1: The Functional Element gets the definition of the data transformation rule, i.e.
555 name, description, rule type, function name, data view name, and applied data fields.

556 2.1.2: The Functional Element checks whether the data transformation rule exists.

557 2.1.3: The Functional Element creates the data transformation rule.

558 2.2: Retrieve Data Transformation Rule.

559 2.2.1: The Functional Element gets name of the data transformation rule and retrieve
560 condition.

561 2.2.2: The Functional Element retrieves the data transformation rule's information
562 according to the condition.

563 2.3: Update Data Transformation Rule.

564 2.3.1: The Functional Element gets the name of data transformation rule.

565 2.3.2: The Functional Element checks whether data transformation rule exists.

566 2.3.3: The Functional Element updates the definition of the data transformation rule.

567 2.4: Delete Data Transformation Rule.

568 2.4.1: The Functional Element gets the name of data transformation rule.

569 2.4.2: The Functional Element checks whether the data transformation rule exists.

570 2.4.3: The Functional Element removes the data transformation rule from the Functional
571 Element

572 3: The Functional Element returns the results of the operation to the user and the use case ends.

573 **2.1.7.3.2 Alternative Flows**

574 1: Data Transformation Rule Already Exists.

575 1.1: If in the basic flow 2.1.2, the data transformation rule is already defined, Functional
576 Element returns an error message and the use case ends.

577 2: Data Transformation Rule Cannot Be Deleted.

578 2.1: If in the basic flow 2.4.3, the data of the logical data view, on which the data
579 transformation rule is applied, is stored in Data Repository, Functional Element returns an
580 error message and the use case ends.

581 3: Data Transformation Rule Not Found.

582 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the data transformation rule does not exist,
583 Functional Element will return an error message and the use case ends

584 4: Data Transformation Rule Cannot Be Updated.

585 4.1: If in the basic flow 2.3.3, the data of the logical data view, on which the data
586 transformation rule is applied, is stored in Data Repository, Functional Element returns an
587 error message and the use case ends.

588 5: Logical Data View Not Exist.

589 4.1: If in the basic flow 2.1.3, the data of the logical data view, on which the data
590 transformation rule is applied, dose not exist, Functional Element returns an error message
591 and the use case ends.

592 6: Persistency Mechanism Error.

593 5.1: If in the basic flow 2.1.2, 2.1.3, 2.2, 2.3.2, 2.3.3, 2.4.2 and 2.4.3, the Functional Element
594 cannot perform data persistency, Functional Element will return an error message to the user
595 and the use case ends.

596 **2.1.7.3.3 Special Requirements**

597 None.

598 **2.1.7.3.4 Pre-Conditions**

599 None.

600 **2.1.7.3.5 Post-Conditions**

601 None.

602

603 **2.1.7.4 Manage Batch Data Extraction**

604 **2.1.7.4.1 Description**

605 This use case allows the user to define and disable the batch data extraction.

606 **2.1.7.4.2 Flow of Events**

607 **2.1.7.4.2.1 Basic Flow**

608 The use case begins when the user wants to define, remove, enable and disable a batch data
609 extraction.

610 1: The user sends request to manage batch data extraction together with the definition of batch
611 data extraction and operation.

612 2: Based on the operation it specifies, one of the following sub-flows is executed:

613 If the operation is '**Define Batch Data Extraction**', the sub-flow 2.1 is executed.

614 If the operation is '**Remove Batch Data Extraction Definition**', the sub-flow 2.1 is executed.

615 If the operation is '**Enable Batch Data Extraction**', the sub-flow 2.2 is executed.

616 If the operation is '**Disable Batch Data Extraction**', the sub-flow 2.3 is executed.

617 2.1: Define Batch Data Extraction.

618 2.1.1: The Functional Element gets batch data extraction definition, i.e. name,
619 description, XML schema for the XML data format, the mapping of data fields from logical
620 data view to XML data file, and extraction schedule.

621 2.1.2: The Functional Element checks whether the batch data extraction exists.

622 2.1.3: The Functional Element creates the batch data extraction.

623 2.2: Remove Batch Data Extraction Definition.

624 2.2.1: The Functional Element gets name of the batch data extraction.

625 2.2.2: The Functional Element checks whether the batch data extraction exists.

626 2.2.3: The Functional Element removes the batch data extraction from the Functional
627 Element.

628 2.3: Enable Batch Data Extraction.

629 2.3.1: The Functional Element gets name of the batch data extraction.

630 2.3.2: The Functional Element checks whether the batch data extraction exists.

631 2.3.3: The Functional Element enables the batch data extraction.

632 2.4: Disable Batch Data Extraction.

633 2.4.1: The Functional Element gets name of the batch data extraction.

634 2.4.2: The Functional Element checks whether the batch data extraction exists.

635 2.4.3: The Functional Element disables the batch data extraction.

636 3: The Functional Element returns the results of the operation to the user and the use case ends.

637 **2.1.7.4.2.2 Alternative Flows**

638 1: Batch Data Extraction Exist.

639 1.1: If in the basic flow 2.1.2, the batch data extraction is already defined, Functional Element
640 returns an error message and the use case ends.

641 2: Batch Data Extraction Not Found.

642 2.1: If in the basic flow 2.2.3, 2.3.3 and 2.4.3, the batch data extraction does not exist,
643 Functional Element will return an error message and the use case ends

644 3: Persistency Mechanism Error.

645 3.1: If in the basic flow 2.1.2, 2.1.3, 2.2.2, 2.2.3,2.3.2, 2.3.3, 2.4.2 and 2.4.3, the Functional
646 Element cannot perform data persistency, Functional Element will return an error message to
647 the user and the use case ends.

648 **2.1.7.4.3 Special Requirements**

649 None.

650 **2.1.7.4.4 Pre-Conditions**

651 None.

652 **2.1.7.4.5 Post-Conditions**

653 None.

654

655 **2.1.7.5 Retrieve Data**

656 **2.1.7.5.1 Description**

657 This use case allows the user to perform data retrieval based on the logical data view defined.

658 **2.1.7.5.2 Flow of Events**

659 **2.1.7.5.2.1 Basic Flow**

660 The use case begins when the user wants to perform data retrieval based on a logical data view.

661 1: The user sends request to retrieve data by providing the name of logical data view and SQL
662 query statement.

663 2: The Functional Element checks whether the logical data view exists.

664 3: The Functional Element retrieves the definition of logical data view specified.

665 4: The Functional Element verifies the correctness of the SQL statement by checking the syntax
666 of statement and the data fields used.

667 5: The Functional Element retrieves the definition of data transformation rule related with the data
668 view.

669 6: The Functional Element performs the data retrieval from data sources

670 7: The Functional Element performs the data transformation to the data retrieved and fill up the
671 data according to the definition of the logical data view.

672 8: The Functional Element returns the results of the operation to the user and the use case ends.

673 **2.1.7.5.2 Alternative Flows**

674 1: Query Statement Is Invalid.

675 1.1: If in the basic flow 4, the SQL statement is not valid, Functional Element returns an error
676 message and the use case ends.

677 2: Data View Not Found.

678 2.1: If in the basic flow 3, the specified data view is not found, Functional Element returns an
679 error message and the use case ends.

680 3: Data Source Not Available.

681 3.1: If in the basic flow 6, the data sources are not available for retrieving data, Functional
682 Element returns an error message and the use case ends.

683 4: Data Transformation Rule Not Found.

684 4.1: If in the basic flow 5, the data transformation rule is not available, Functional Element
685 returns an error message and the use case ends.

686 5: Data Repository Are Not Available.

687 5.1: If in the basic flow 6, the data of the logical data view is stored in Data Repository and
688 the Data Repository is not available, Functional Element returns an error message and the
689 use case ends.

690 **2.1.7.5.3 Special Requirements**

691 None.

692 **2.1.7.5.4 Pre-Conditions**

693 None.

694 **2.1.7.5.5 Post-Conditions**

695 None.

696

697 **2.1.7.6 Manipulate Data**

698 **2.1.7.6.1 Description**

699 This use case allows the user to insert, update, and delete data based on a logical data view
700 defined.

701 **2.1.7.6.2 Flow of Events**

702 **2.1.7.6.2.1 Basic Flow**

703 The use case begins when the user wants to insert, update, and delete data based on a logical
704 data view.

705 1: The user sends request to manipulate data by providing the name of the logical data view and
706 SQL statement.

- 707 3: The Functional Element retrieves the definition of logical data view specified.
- 708 4: The Functional Element verifies the correctness of the SQL statement by checking the syntax
709 of statement and the data fields used.
- 710 5: The Functional Element checks the violation of operations based on the definition of logical
711 data view.
- 712 6: The Functional Element performs the operation specified in SQL statement.
- 713 7: The Functional Element returns the results of the operation to the user and the use case ends.

714 **2.1.7.6.2 Alternative Flows**

- 715 1: Manipulation Statement Is Invalid.
- 716 1.1: If in the basic flow 4, the SQL statement is not valid, Functional Element returns an error
717 message and the use case ends.
- 718 2: Data View Not Found.
- 719 2.1: If in the basic flow 3, the specified data view is not found, Functional Element returns an
720 error message and the use case ends.
- 721 3: Data Source Are Not Available.
- 722 3.1: If in the basic flow 6, the data sources are not available for retrieving data, Functional
723 Element returns an error message and the use case ends.
- 724 4: SQL Error.
- 725 4.1: If in the basic flow 6, there is any error of SQL statement execution, Functional Element
726 returns an error message and the use case ends.

727 **2.1.7.6.3 Special Requirements**

728 None.

729 **2.1.7.6.4 Pre-Conditions**

730 None.

731 **2.1.7.6.5 Post-Conditions**

732 None.

733

734 **2.1.7.7 Extract Data in Batch**

735 **2.1.7.7.1 Description**

736 This use case allows the user to perform batch data retrieval in a scheduled approach based on a
737 logical data view defined.

738 **2.1.7.7.2 Flow of Events**

739 **2.1.7.7.2.1 Basic Flow**

740 The use case begins when the user wants to perform batch data retrieval or the time is up for
741 scheduled batch data retrieval.

742 1: The user sends request to retrieve data by providing the name of the batch data retrieval or the
743 Functional Element clock generates a trigger.

744 2: The Functional Element retrieves the definition of batch data retrieval according to the name.

745 3: The Functional Element prepares the parameters for invocation of Retrieve data use case

746 4: The Functional Element invokes the Data Retrieve use case

747 5: The Functional Element formats the data according to the format defined in the batch data
748 retrieval definition

749 6: The Functional Element returns the results of the operation to the user and the use case ends.

750 **2.1.7.7.2.2 Alternative Flows**

751 1: Definition of Batch Data Retrieval Not Found.

752 1.1: If in the basic flow 2, the definition of batch data retrieval is not found, Functional Element
753 returns an error message and the use case ends.

754 2: Error Returned From Data Retrieve Use Case.

755 2.1: If in the basic flow 4, the use case Retrieve data returns an error, Functional Element
756 returns an error message and the use case ends.

757 **2.1.7.7.3 Special Requirements**

758 None.

759 **2.1.7.7.4 Pre-Conditions**

760 None.

761 **2.1.7.7.5 Post-Conditions**

762 None.

763

764 **2.1.7.8 Manage Data Repository**

765 **2.1.7.8.1 Description**

766 This use case allows the user to manage data repository.

767 **2.1.7.8.2 Flow of Events**

768 **2.1.7.8.2.1 Basic Flow**

769 The use case begins when the user wants to persistent a logical data view in the data repository,
770 or the user wants to dispose the persistency of a data view from the data repository.

771 1: The user sends request to manage data repository by providing the name of the logical data
772 view.

773 2: Based on the operation it specifies, one of the following sub-flows is executed:

774 If the operation is '**Persistent Data View**', the sub-flow 2.1 is executed.

775 If the operation is '**Dispose Data View**', the sub-flow 2.1 is executed.

776 2.1: Persistent Data View

777 2.1.1: The Functional Element retrieves the definition of the logical data view.

778 2.1.2: The Functional Element forms the SQL statement according to the definition of the
779 logical data view.

780 2.1.3: The Functional Element performs the data retrieval from data sources.

781 2.1.4: The Functional Element performs the data transformation according to the
782 transformation rule.

783 2.1.5: The Functional Element creates table in Data Repository and fill in the table with
784 data generated in previous step.

785 2.2: Dispose Data View

786 2.1.1: The Functional Element forms the SQL statements of deleting the table

787 2.1.3: The Functional Element deletes the table in Data Repository.

788 3: The Functional Element returns the results of the operation to the user and the use case ends.
789
790

791 **2.1.7.8.2.2 Alternative Flows**

792 1: Data View Not Found

793 1.1: If in the basic flow 2.1.1, the definition of batch data retrieval is not found, Functional
794 Element returns an error message and the use case ends.

795 2: Data Exist

796 2.1: If in the basic flow 2.1.3, there is data in the table, Functional Element returns an error
797 message and the use case ends.

798 3: Data Repository Error

799 3.1: If in the basic flow 2.1.5 and 2.2.3, there is an error in Data Repository, Functional
800 Element returns an error message and the use case ends.

801 4: Data Source Not Available

802 4.1: If in the basic flow 2.1.3, the data sources related is not available, Functional Element
803 returns an error message and the use case ends

804 **2.1.7.8.3 Special Requirements**

805 None.

806 **2.1.7.8.4 Pre-Conditions**

807 None.

808 **2.1.7.8.5 Post-Conditions**

809 None.

810

811 **2.1.7.9 Synchronize Data**

812 **2.1.7.9.1 Description**

813 This use case allows the user to synchronize data in Data Repository with the data from data
814 sources.

815 **2.1.7.9.2 Flow of Events**

816 **2.1.7.9.2.1 Basic Flow**

817 The use case begins when the user wants to synchronize data of a logical data view in data
818 repository with the data in data sources, or the time is up for synchronization of data.

819 1: The user sends request to synchronize data repository or the Functional Element clock
820 generates a trigger.

821 2: The Functional Element gets or finds those data views that are required to be synchronized
822 with data sources.

823 3: The Functional Element retrieves data view definitions.

824 4: The Functional Element retrieves data from data sources according th definition of logical data
825 view.

826 5: The Functional Element performs the data transformation on the data retrieved.

827 6: The Functional Element update the table in Data Repository with the data generated in
828 previous step.

829 7: The Functional Element returns the result of the operation and the use case ends.

830 **2.1.7.9.2.2 Alternative Flows**

831 1: Data View Definition Not Found

832 1.1: If in the basic flow 3, the definition of batch data retrieval is not found, Functional Element
833 returns an error message and the use case ends.

834 2: Data Repository Error

835 2.1: If in the basic flow 6, there is an error in updating the Data repository, Functional Element
836 returns an error message and the use case ends.

837 3: Data Source Not Available

838 3.1: If in the basic flow 4, the data sources related is not available, Functional Element returns
839 an error message and the use case ends

840 **2.1.7.9.3 Special Requirements**

841 None.

842 **2.1.7.9.4 Pre-Conditions**

843 None.

844 **2.1.7.9.5 Post-Conditions**

845 None.

846

847 **2.2 Error Management Functional Element (new)**

848 **2.2.1 Motivation**

849 Error management is an important aspect in any software application development. In particular,
850 it is important to know the cause of error in the Service Oriented Architecture (SOA) environment
851 as an application can consume any service provided from any domain space spans across the
852 Internet space. When an error occurs, it can be from within the same application domain or from
853 different domain space. Hence, it is important to know the system state when the error occurred
854 in the SOA environment. For example, when an error occurred, what services were used; which
855 services' interfaces were used; the passed in parameters and its associated values used for the
856 interfaces, the time when the error occurred, API or SOAP invocation, etc are the important
857 information for managing the application in the SOA environment.

858

859 The Error Management Functional Element is a framework designed to capture the system state
860 at which the error occurred. The variables that governed the system state when an error
861 occurred are defined as follows:

- 862 • The time at which the error occurred.
- 863 • The class/object name that the error occurred.
- 864 • The method name of the said class/object at which the error occurred.
- 865 • The input parameters, parameters types and its associated values of the said method at
866 which the error occurred.
- 867 • The expected output type of the mentioned method name.
- 868 • The error category, error code and error severity assigned by the application.
- 869 • The name of the consumed service/component.
- 870 • The name of the interface used for the said service/component.
- 871 • The input parameters and types defined for the said interface.
- 872 • The values used for the mentioned input parameters.
- 873 • The Universal Resource Location (URL) of the consumed service endpoint.
- 874 • The SOAP Fault message <Fault> element returned from the consumed service.
- 875 • The type of invocation whether it is a Web Service call or Application Programming
876 Interfaces (APIs) call.
- 877 • The domain controller information includes :
 - 878 ○ Name of the domain controller
 - 879 ○ Contact Information, .e. Email Id, Short Message Services (SMS), Telephone,
880 Mobile phone, etc.
 - 881 ○ Means of Notification

882 The main motivation of the Functional Element is to provide a snapshot and capture all the
883 system state information for an application when an error occurred. It assists system
884 administrator to manage the system fault better for the necessary actions required for tracking the
885 fault.

886

887 Figure 3 illustrates the perspective usage of Error Management Functional Element. When an
888 error occurred in an application, the Functional Element will be used to capture the system state
889 into a data store which can either be a database or a flat file.

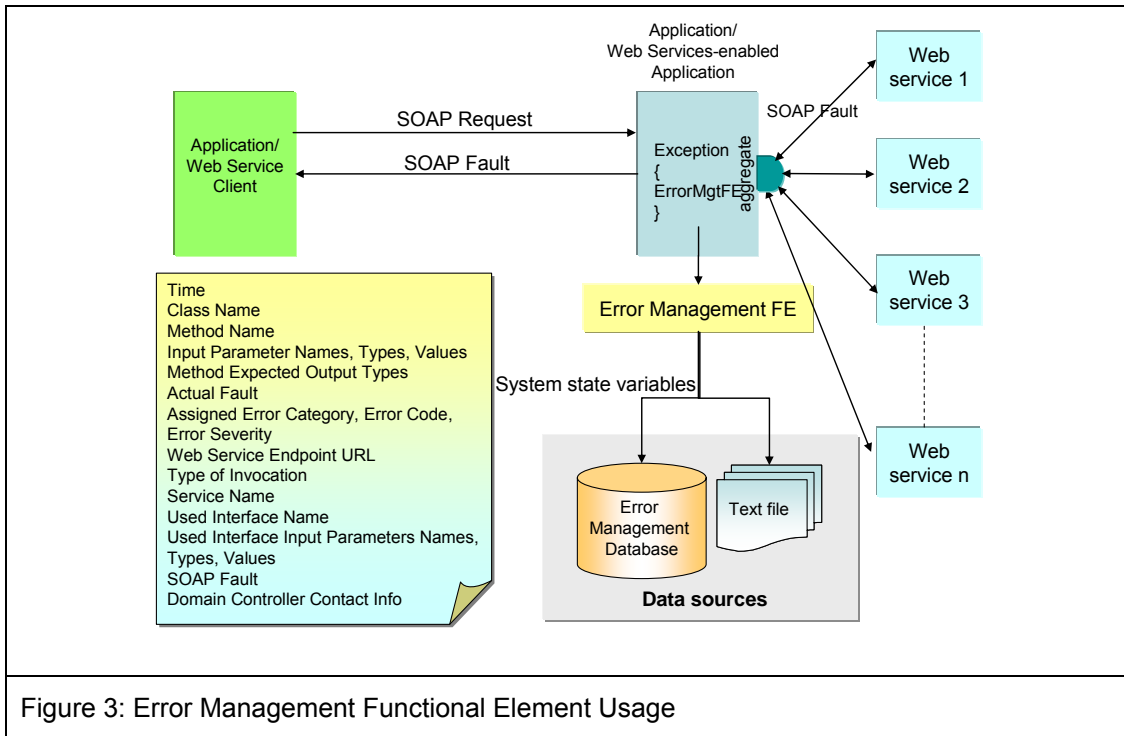


Figure 3: Error Management Functional Element Usage

890

891 This Functional Element fulfills the following requirements from the Functional Elements
892 Requirements Document 02:

- 893
- Primary Requirements
 - MANAGEMENT-340 to MANAGEMENT-346
 - Secondary Requirements
 - None

896

897

898 2.2.2 Terms Used

Terms	Description
Error Category	The Category or classification of error. For example, the category of error can be classified as: Database error → DATABX, Transaction error → TRANSX, Authentication error → AUTHEX, System error → SYSTEMX, Application-specific error → APPLSX, Third-party service error → THIRDPX, etc.
Error Code	The Error Code defined for each Error Category. For example, 001, 002, 003, etc

Error Severity	The Error Severity defined for each Error Code. For example, the severity could be in the order of <i>Critical, Major, Minor, Warning, For Information Only</i> .
External Application Error	The External Application Error is defined as an error / fault / exception occurred by consuming external Web Services / Components providers. For example, customized exception, SOAPException and SOAP Fault resulted from APIs or SOAP invocation to external components or Web Services.
Internal Application Error	The Internal Application Error is defined as error / fault / exception raised resulted from an internal processing or run time error. For example, exceptions such as Null Pointer Exception, Class Type Casting, Array Out of Bound, etc. that occurred due to processing or run time error.

899

900 Figure 4 is an example illustrating the error hierarchy in terms of Error Category, Error Code and
 901 Error Severity.

902

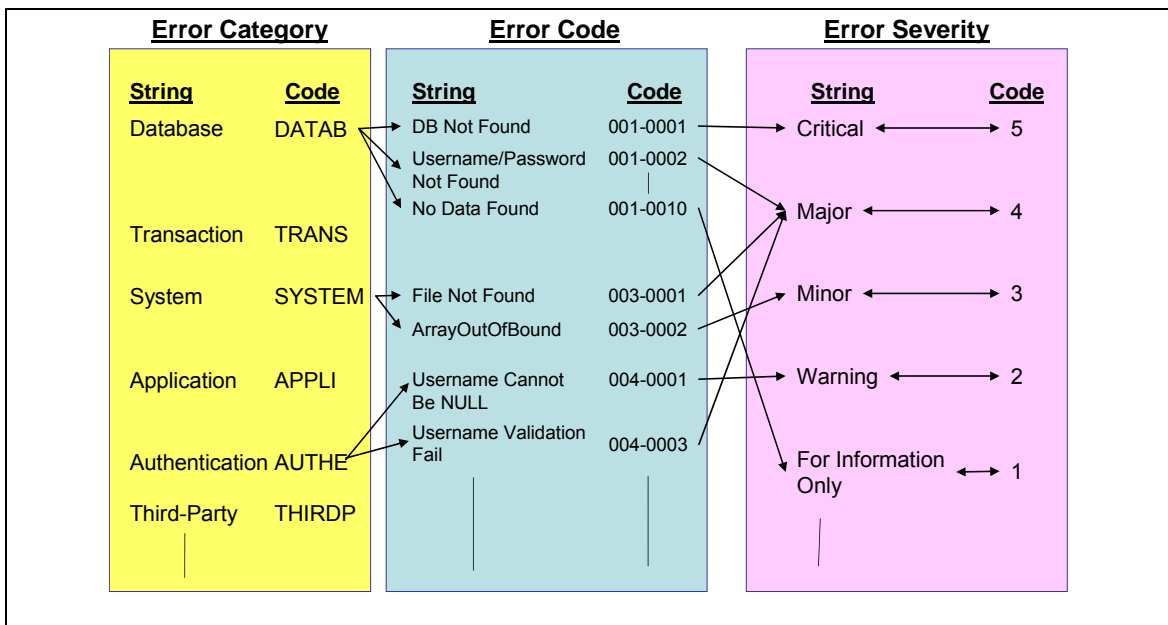


Figure 4: An Example of Error Hierarchy

903

904 For example, a database could give rise to a number of errors. For example, database not found,
 905 invalid username and password, no data found, null field, duplicate key etc are the common
 906 database errors. Each database error could have different severity. For example, database not
 907 found or invalid username and password are critical to business logic. An illustration of the
 908 resultant error code is defined as DATABX0001-CRITICAL.

909

910 2.2.3 Key Features

911 Implementations of the Error Management Functional Element are expected to provide the
 912 following key features:

- 913 1. The Functional Element MUST provide the ability to create new Error Category

- 914 2. The Functional Element MUST provide the ability to modify and delete defined Error
 915 Category.
- 916 3. The Functional Element MUST provide the ability to all the information stored in the Error
 917 Category. This includes the capability to:
- 918 3.1 Add new Error Code(s) and descriptions into a Error Category
- 919 3.2 Retrieve, modify and delete error code and descriptions
- 920 3.3 Support Error Code(s) in numeric, alpha-numeric or string format
- 921 4. The Functional Element MUST provide a mechanism to capture the defined system state at
 922 which an error occurred.

923

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

- 926 1. The Functional Element MAY provide the ability to manage Error Severity by enabling the
 927 capability to:
- 928 3.1 Tag/Add to Error Code defined
- 929 3.2 Retrieve, modify and delete Severity tag to Error Code
- 930 3.3 Retrieve information based on either Error Code or Severity

931

932 **2.2.4 Interdependencies**

Direct Dependency	
Log Utility Functional Element	The Log Utility Functional Element helps to log the audit trail.
Notification Functional Element	The Notification Element helps to notify the target user via email, or short messaging service.

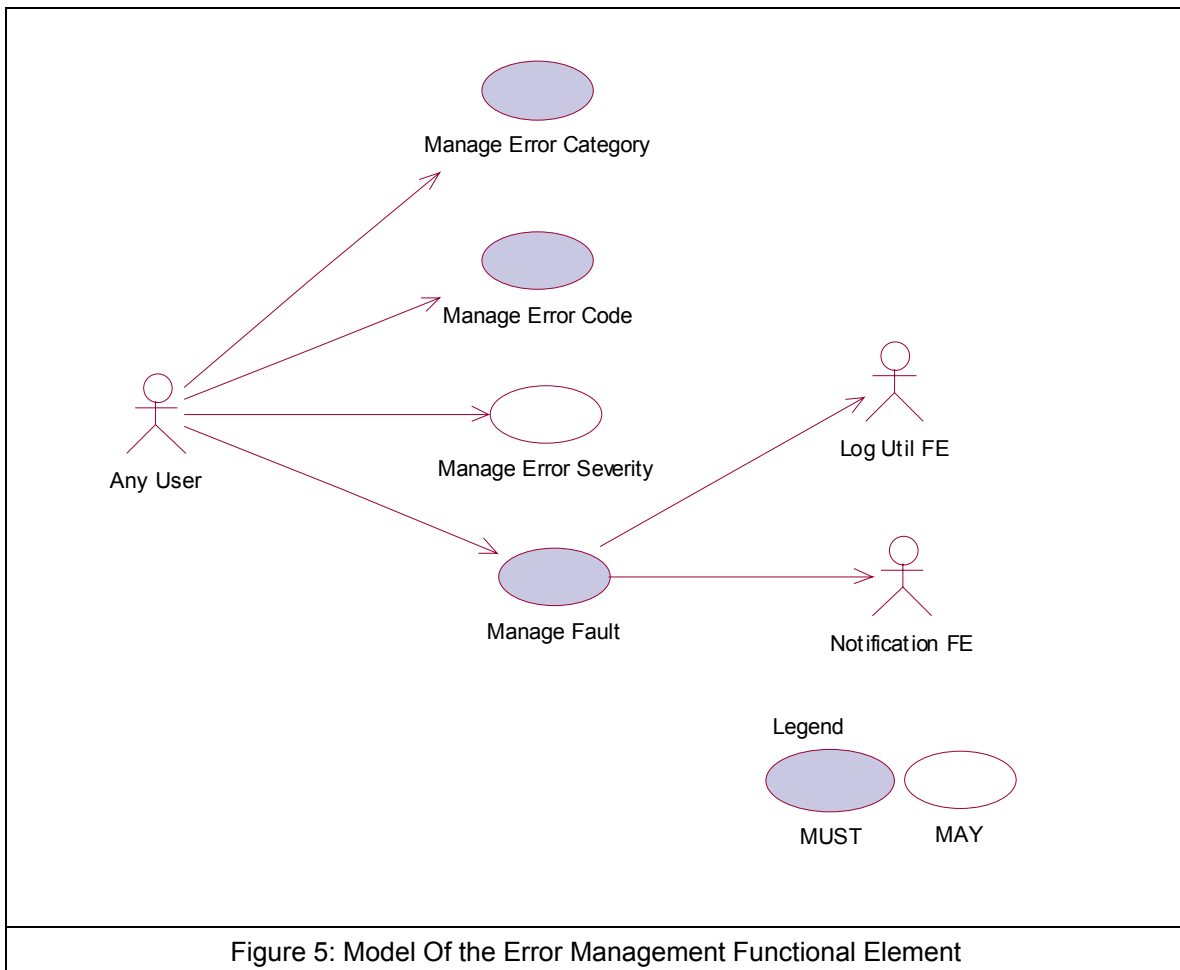
933

934 **2.2.5 Related Technologies and Standards**

Specifications	Specific References
XML Version 1.0	Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation 04 February 2004.
XML Schema	XML Schema Part 0: Primer Second Edition W3C Recommendation 28 October 2004 XML Schema Part 1: Structures Second Edition W3C Recommendation 28 October 2004 XML Schema Part 2: Datatypes Second Edition W3C Recommendation 28 October 2004
WSDL Version 1.1	Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001
SOAP Version 1.1	Simple Object Access Protocol (SOAP) 1.1 W3C Note 08 May 2000
Functional Elements Specification	OASIS Functional Elements Specification Committee Specifications 1.0, 16-Dec-2004

935

2.2.6 Model



937

938 2.2.7 Usage Scenarios

939 2.2.7.1 Manage Error Category

940 2.2.7.1.1 Description

941 This use case allows the error management administrator to manage Error Category.

942 2.2.7.1.2 Flow of Events

943 2.2.7.1.2.1 Basic Flow

944 The use case begins when the user wants to create/retrieve/update/delete an Error Category.

945 1: The user sends a request to manipulate an Error Category.

946 2: Based on the operation it specifies, one of the following sub-flows is executed:

947 If the operation is '**Create Error Category**', the sub-flow 2.1 is executed.

948 If the operation is **'Retrieve Error Category'**, the sub-flow 2.2 is executed.

949 If the operation is **'Update Error Category'**, the sub-flow 2.3 is executed.

950 If the operation is **'Delete Error Category'**, the sub-flow 2.4 is executed.

951 2.1: Create Error Category.

952 2.1.1: The Functional Element gets category definition.

953 2.1.2: The Functional Element checks whether the category exists.

954 2.1.3: The Functional Element creates the category and save it in the error database.

955 2.2: Retrieve Error Category.

956 2.2.1: The Functional Element gets the Error Category name.

957 2.2.2: The Functional Element checks whether the category exists.

958 2.2.3: The Functional Element retrieves the Error Category's information from the Error
959 Management Data sources.

960 2.3: Update Error Category.

961 2.3.1: The Functional Element gets the Error Category name.

962 2.3.2: The Functional Element checks whether the Error Category exists.

963 2.3.3: The Functional Element updates the category definition and save it in the Error
964 Management Data sources.

965 2.4: Delete Error Category.

966 2.4.1: The Functional Element gets the Error Category name.

967 2.4.2: The Functional Element checks whether the Error Category name exists.

968 2.4.3: The Functional Element checks whether the Error Code associated to the Error
969 Category name exists.

970 ○ If Error Codes associated to the Error Category name exists, then basic sub-flow
971 2.4.4 is executed.

972 ○ If Error Codes associated to the Error Category name does not exist, then the
973 basic sub-flow 2.4.7 is executed.

974 2.4.4: Error Codes associated to the Error Category name exists.

975 ○ If the Error Severity associated to the respective Error Codes exists, the basic
976 sub-flow 2.4.5 is executed.

977 ○ If the Error Severity associated to the respective Error Code does not exist, then
978 the basic sub-slow 2.4.6 is executed.

979 2.4.5: The Error Severity Exist.

980 2.4.5.1: The Functional Element removes the error severities associated to the
981 respective Error Code from sub-flow 2.4.4.

982 2.4.6: The Error Severity Does Not Exist.

983 2.4.6.1 The Functional Element removes the respective Error Codes from sub-flow
984 2.4.3 from the Error Management Data sources.

985 2.4.7: The Error Codes Associated to the Error Category Name Does Not Exist.

986 2.4.7.1: The Functional Element removes the respective Error Codes associated to
987 the Error Category name (from sub-flow 2.4.3) from the Error Management Data
988 sources.

989 2.4.8: The Functional Element removes the Error Category name from the Error
990 Management Data sources.

991 3: The Functional Element returns the results of the operation to the end user and the use case
992 ends.

993 **2.2.7.1.2.2 Alternative Flows**

994 1: Error Category Already Exists.

995 1.1: If in the basic flow 2.1.2, the error category is already defined, the Functional Element
996 writes the system state variables into the Error Management Data Sources using Log Utility
997 Functional Element and notifies the system domain controller using the Notification
998 Functional Element and the use case ends.

999 1.1: If in the basic flow 2.1.2, the error category is already defined, the Functional Element
1000 writes the system state variables into the Error Management Data Sources using Log Utility
1001 Functional Element and notifies the system domain controller using the Notification
1002 Functional Element and the use case ends.

1003

1004 2: Error Category Not Found.

1005 2.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the error category does not exist, the Functional
1006 Element writes the system state variables into the Error Management Data Sources using
1007 Log Utility Functional Element and notifies the system domain controller using the Notification
1008 Functional Element and the use case ends.

1009 **2.2.7.1.3 Special Requirements**

1010 None.

1011 **2.2.7.1.4 Pre-Conditions**

1012 None.

1013 **2.2.7.1.5 Post-Conditions**

1014 Once the Error Category is deleted, all the associated Error Code and its Error Severity will be
1015 removed.

1016

1017 **2.2.7.2 Manage Error Code**

1018 **2.2.7.2.1 Description**

1019 This use case allows the user to manage Error Code.

1020 **2.2.7.2.2 Flow of Events**

1021 **2.2.7.2.2.1 Basic Flow**

1022 The use case begins when the user wants to create/retrieve/update/delete an error code
1023 associated to an error category.

1024 1: The user sends a request to manipulate an error code.

1025 2: Based on the operation it specifies, one of the following sub-flows is executed:

1026 If the operation is '**Create Error Code**', the sub-flow 2.1 is executed.

1027 If the operation is '**Retrieve Error Code**', the sub-flow 2.2 is executed.

1028 If the operation is '**Update Error Code**', the sub-flow 2.3 is executed.

1029 If the operation is '**Delete Error Code**', the sub-flow 2.4 is executed.

1030 2.1: Create Error Code.

1031 2.1.1: The Functional Element gets the Error Category name

1032 2.1.2. The Functional Element gets Error Code definition for the Error Category.

1033 2.1.3: The Functional Element checks whether the Error Code exists.

1034 2.1.4: The Functional Element creates the Error Code for the Error Category name and
1035 saves it into the Fault Management database.

1036 2.2: Retrieve Error Code.

1037 2.2.1: The Functional Element gets the Error Category name

1038 2.2.2. The Functional Element gets the Error Code name.

1039 2.2.3: The Functional Element checks whether the Error Code exists.

1040 2.2.4. The Functional Element retrieves the Error Code's information from the error
1041 database.

1042 2.3: Update Error Code.

1043 2.3.1: The Functional Element gets the Error Category name.

1044 2.3.2. The Functional Element gets the Error Code name.

1045 2.3.3: The Functional Element checks whether the Error Code exists.

1046 2.3.4: The Functional Element updates the error code definition associated to the Error
1047 Category and save it in the Error Management Data sources.

1048 2.4: Delete Error Code.

1049 2.4.1: The Functional Element gets the Error Category name.

1050 2.4.2. The Functional Element gets the Error Code name.

1051 2.4.3: The Functional Element checks whether the Error Code exists.

1052 2.4.4: The Functional Element checks whether the Error Severity associated to the Error
1053 Category and Error Code exists. Depending on whether the Error Severity exists, one of
1054 the following sub-flows will be executed.

- 1055 ○ If the Error Severity exists, then basic sub-flow 2.4.5 is executed.
- 1056 ○ If the Error Severity does not exist, the basic sub-flow 2.4.6 is executed.

1057 2.4.5: Error Severity Exists.

1058 2.4.5.1: The Functional Element removes the Error Severity associated to the Error
1059 Category and Error Code from the Error Management Data sources.

1060 2.4.5.2: The Functional Element removes the Error Code associated to the Error
1061 Category name from the Error Management Data sources.

1062 2.4.6: Error Severity Does Not Exist

1063 2.4.6.1: The Functional Element removes the Error Code associated to the Error
1064 Category name from the Error Management Data sources.

1065 3: The Functional Element returns the results of the operation to the end user and the use case
1066 ends.

1067 **2.2.7.2.2.2 Alternative Flows**

1068 1: Error Code Already Exists.

1069 1.1: If in the basic flow 2.1.3, the Error Code associated to the Error Category name is
1070 already defined, the Functional Element writes the system state variables into the Error
1071 Management Data sources using the Log Utility Functional Element and notifies the system
1072 domain controller using the Notification Functional Element and the use case ends.

1073 2: Error Code Not Found.

1074 2.1: If in the basic flows 2.2.3, 2.3.3 and 2.4.3 the Error Code associated to the Error
1075 Category name does not exist, the Functional Element writes the system state variables into
1076 the Error Management Data sources using the Log Utility Functional Element and notifies the
1077 system domain controller using the Notification Functional Element and the use case ends.

1078 **2.2.7.2.3 Special Requirements**

1079 None.

1080 **2.2.7.2.4 Pre-Conditions**

1081 None.

1082 **2.2.7.2.5 Post-Conditions**

1083 None.

1084

1085 **2.2.7.3 Manage Error Severity**

1086 **2.2.7.3.1 Description**

1087 This use case allows the user to manage error severity.

1088 **2.2.7.3.2 Flow of Events**

1089 **2.2.7.3.2.1 Basic Flow**

1090 The use case begins when the user wants to create/retrieve/update/delete an Error Severity
1091 associated to an Error Category and Error Code.

1092 1: The user sends a request to manipulate an error severity.

1093 2: Based on the operation it specifies, one of the following sub-flows is executed:

1094 If the operation is '**Create Error Severity**', the sub-flow 2.1 is executed.

1095 If the operation is '**Retrieve Error Severity**', the sub-flow 2.2 is executed.

1096 If the operation is '**Update Error Severity**', the sub-flow 2.3 is executed.

1097 • If the operation is '**Delete Error Severity**', the sub-flow 2.4 is executed

1098 2.1: Create Error Severity.

1099 2.1.1: The Functional Element gets Error Category name.

1100 2.1.2: The Functional Element gets Error Code name.

1101 2.1.3: The Functional Element gets Error Severity definition.

1102 2.1.4: The Functional Element checks whether the Error Severity associated to the Error
1103 Category and error Code name exists.

1104 2.1.5: The Functional Element creates the Error Severity associated to the Error
1105 Category name and Error Code name and saves it into the Error Management Data
1106 sources.

1107 2.2 Retrieve Error Severity.

1108 2.2.1: The Functional Element gets the Error Category name.

1109 2.2.2: The Functional Element gets the Error Code name.

1110 2.2.3. The Functional Element gets the Error Severity name.

1111 2.2.4: The Functional Element checks whether the Error Severity exists associated to the
1112 Error Category and Error Code names.

1113 2.2.5. The Functional Element retrieves the Error Severity's information associated to the
1114 Error Category and Error Code names from the Error Management Data sources.

1115 2.3: Update Error Severity.

1116 2.3.1: The Functional Element gets the Error Category name.

1117 2.3.2: The Functional Element gets the Error Code name.

1118 2.3.3. The Functional Element gets the Error Severity name.
1119 2.3.4: The Functional Element checks whether the Error Severity exists associated to the
1120 Error Category and Error Code names.
1121 2.3.5: The Functional Element updates the Error Severity definition associated to the
1122 Error Category and Error Code names and saves it into the Error Management Data
1123 sources.
1124 2.4: Delete Error Severity.
1125 2.4.1: The Functional Element gets the Error Category name.
1126 2.4.2: The Functional Element gets the Error Code name.
1127 2.4.3. The Functional Element gets the Error Severity name.
1128 2.4.4: The Functional Element checks whether the Error Severity exists associated to the
1129 Error Category and Error Code names.
1130 2.4.5: The Functional Element removes the Error Severity associated to the Error
1131 Category and Error Code names from the Error Management Data sources.
1132 3: The Functional Element returns the results of the operation to the end user and the use case
1133 ends.

1134 **2.2.7.3.2 Alternative Flows**

1135 1: Error Severity Already Exists.
1136 1.1: If in the basic flow 2.1.4, the Error Severity associated to the Error Category and Error
1137 Code names is already defined, the Functional Element writes the system state variables into
1138 the Error Management Data sources using the Log Utility Functional Element and notifies the
1139 system domain controller using the Notification Functional Element and the use case ends.
1140 2: Error Severity Not Found.
1141 2.1: If in the basic flows 2.2.4, 2.3.4 and 2.4.4, the Error Severity associated to the Error
1142 Category and Error Code names does not exist, the Functional Element writes the system
1143 state variables into the Error Management Data sources using the Log Utility Functional
1144 Element and notifies the system domain controller using the Notification Functional Element
1145 and the use case ends.

1146 **2.2.7.3.3 Special Requirements**

1147 None

1148 **2.2.7.3.4 Pre-Conditions**

1149 None

1150 **2.2.7.3.5 Post-Conditions**

1151 None

1152 **2.2.7.4 Manage Fault**

1153 **2.2.7.4.1 Description**

1154 This use case allows an application to manage error/fault depicted from a consumed service.

1155 **2.2.7.4.2 Flow of Events**

1156 **2.2.7.4.2.1 Basic Flow**

1157 The use case begins when the user wants to manage an application' fault arises.

1158 If it is the '**Internal Application Error**, then basic flow 1 is executed.

1159 If it is the '**External Application Error**, the basic flow 2 is executed.

1160 1. Internal Application Error.

1161 1.1. User sends the internal error detail information that needs to be tracked, together with
1162 Error Category, Error Code and Error Severity, which is an optional parameter, to the
1163 Functional Element. The internal error detailed information is described by Table 1.

1164 1.2 The Functional Element logs the System State Information as defined in Table 1 using
1165 the Log Utility Functional Element into the Error Management Data sources.

1166

S/N	Attributes of System State	Description	Mandatory / Optional
1	Time	The time where the fault occurred.	Mandatory
2	Class Name	The name of class where the fault occurred	Mandatory
3	Method Name	The name of the method where the fault occurred.	Mandatory
4	Input Parameters Names	The list of input parameters names for the said method name.	Mandatory
5	Input Parameters Types	The list of input parameter types associated to each of the input parameters names of the said method name.	Mandatory
6	Input Parameters Values	The list of input parameters values associated to each of the input parameters names of the said method name.	Mandatory
7	Expected Output Type	The expected output type of the said method name.	Optional

8	Fault	The fault that causes the exception.	Mandatory
9	Error Category	The Error Category assigned to the said Fault.	Mandatory
10	Error Code	The Error Code assigned to the said Fault.	Mandatory
11	Error Severity	The Error Severity assigned to the said Fault, if any.	Optional
12	Domain Controller Contact	The contact information of the domain controller. The contact information entails: Name of domain controller Email Id / Short Messaging Services (SMS) / Telephone / Mobile Phone Means of Notification	Mandatory

1167 Table 1 System State Information for Internal Application Error

1168

1169 2. External Application Error

1170 2.1 User sends error information that needs to be tracked, as well as Error Category, Error
1171 Code and optional Error Severity to the Functional Element. The external error information
1172 includes System State Information for Internal Application Error defined in Table 1.

1173 2.2 The Functional Element logs the System State Information as defined in Table 2 using
1174 the Log Utility Functional Element into the Error Management Data sources.

1175

S/No.	Attributes of System State	Description	Mandatory / Optional
1	Time	The time where the fault occurred.	Mandatory
2	Class Name	The name of class where the fault occurred	Mandatory
3	Method Name	The name of the method where the fault occurred.	Mandatory
4	Input Parameters Names	The list of input parameters names for the said method name.	Mandatory
5	Input Parameters Types	The list of input parameter types associated to each of	Mandatory

		the input parameters names of the said method name.	
6	Input Parameters Values	The list of input parameters values associated to each of the input parameters names of the said method name.	Mandatory
7	Expected Output Type	The expected output type of the said method name.	Optional
8	Fault	The fault that causes the exception.	Mandatory
9	Error Category	The Error Category assigned to the said Fault.	Mandatory
10	Error Code	The Error Code assigned to the said Fault.	Mandatory
11	Error Severity	The Error Severity assigned to the said Fault, if any.	Optional
12	Domain Controller Contact	The contact information of the domain controller. The contact information entails: Name of domain controller Email Id / Short Messaging Services (SMS) / Telephone / Mobile Phone Means of Notification	Mandatory
13*	Web Services Endpoint URL	The URL for the consumed web service.	Mandatory
14*	Invocation Type	The invocation type used for interface invocation, i.e. API or SOAP invocation.	Mandatory
15*	Consumed Web Service Name	The name of the consumed web service from within the application.	Mandatory
16*	Used Interface Name	The name of the interface used	Mandatory
17*	Used Interface Input Parameters Name	The list of input parameters names required for the said interface.	Mandatory

18*	Used Interface Input Parameters Types	The list of input parameters names types defined for the said interface.	Mandatory
19*	Used Interface Input Parameters Values	The list of input parameters values passed in for the said interface.	Mandatory
20*	SOAP Fault <Fault> Element	The content of the received SOAP Fault message <Fault> element.	Mandatory

1176 Table 2. System State Information for External Application

1177 Items indicated by the symbol “*” are the additional System State Information attributes
1178 which are applicable to External Application Error only.

1179 3. The Functional Element returns the result of the operation to the user and the use case ends.

1180 **2.2.7.4.2.2 Alternative Flow**

1181 1: Error Category Does Not Exist

1182 1.1: If in the basic flows 1.1 and 2.1, the Error Category Name is not defined, the Functional
1183 Element writes the system state variables into the Error Management Data sources using
1184 the Log Utility Functional Element and notifies the system domain controller and the use
1185 case ends.

1186

1187 2. Error Code Does Not Exist

1188 2.1. If in the basic flows 1.1 and 2.1, the Error Code associated to the Error Category is not
1189 defined, the Element writes the system state variables into the Error Management Data
1190 sources using the Log Utility Functional Element and notifies the system domain controller
1191 using the Notification Functional Element and the use case ends.

1192

1193 3. Error Severity Does Not Exist

1194 3.1. If in the basic flows 1.1 and 2.1, the Error Severity associated to the Error Category,
1195 and Error Code is not defined, the Functional Element writes the system state variables into
1196 the Error Management Data sources using the Log Utility Functional Element and notifies
1197 the system domain controller using the Notification Functional Element and the use case
1198 ends.

1199

1200 4. Log Utility Functional Element Not Available.

1201 4.1. If in the basic flows 1.2 and 2.2, the Log Utility Functional Element writes the system
1202 state variables into the Error Management Data sources using the Log Utility Functional
1203 Element and notifies the system domain controller using the Notification Functional
1204 Element and the use case ends.

1205

1206 **2.2.7.4.3 Special Requirements**

1207 None

1208 **2.2.7.4.4 Pre-Conditions**

1209 None

1210 **2.2.7.4.5 Post-Conditions**

1211 None

1212

1213

1214

1215 **2.3 Event Handler Functional Element**

1216 **2.3.1 Motivation**

1217 Information is in abundance in a service-oriented environment. However, not all information is
1218 applicable to a particular enterprise and there lies the need to control information flow in an
1219 organization. In a Web Service-enabled implementation, the Event Handler Functional Element
1220 can help to fulfill this need by:

- 1221 • Managing the information flow through a subscription based mechanism,
- 1222 • Streamlining information into meaningful categories so as to improve relevancy to a
1223 potential consumer of the information, and
- 1224 • Refining information flow via a filtering mechanism

1225 This Functional Element fulfills the following requirements from the Functional Elements
1226 Requirements Document 02:

- 1227 • Primary Requirements
 - 1228 ○ MANAGEMENT-111,
 - 1229 ○ PROCESS-005, and
 - 1230 ○ PROCESS-100 to PROCESS-117.
- 1231 • Secondary Requirements
 - 1232 ○ None

1233

1234 **2.3.2 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.

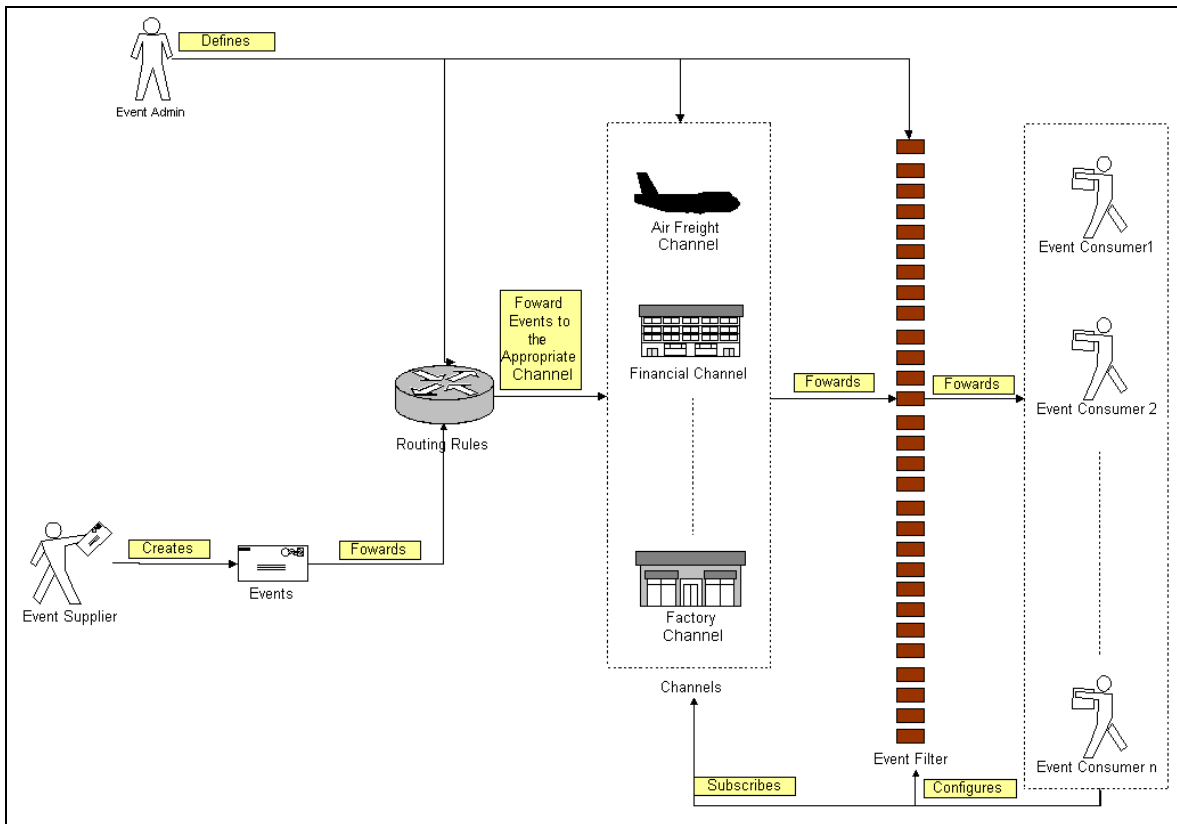


Figure 6: An Overview of the Event Handler Functional Element

1236

1237 Figure 3 depicts the basic concepts of how the participating entities collaborate together in the
 1238 Event Handler Functional Element. Beginning with the event supplier who generates an event,
 1239 the event is subsequently routed to the routing rules engine. Depending on the rules specified by
 1240 the event administrator on the engine, the event could be routed to an appropriate channel, for
 1241 example, the airfreight channel. In this case, a notification message will be sent to the subscribing
 1242 event consumers. In between that, there is a filtering engine to determine if a particular event is
 1243 meaningful to the intended recipients and this is configurable by the recipients themselves.

1244 **2.3.3 Key Features**

1245 Implementations of the Event Handler Functional Element are expected to provide the following
 1246 key features:

- 1247 1. The Functional Element MUST provide the capability to manage the creation (or registration)
- 1248 and deletion of instances of the following concepts based on a pre-defined structure:
 - 1249 1.1. Event Supplier,
 - 1250 1.2. Event Consumer,
 - 1251 1.3. Event,
 - 1252 1.4. Filter,
 - 1253 1.5. Channel, and
 - 1254 1.6. Routing Rule.

- 1255 2. The Functional Element MUST provide the capability to manage all the information (attribute
1256 values) stored in such concepts. This includes the capability to retrieve and update
1257 attribute's values belonging to the concepts mentioned in Key Feature (1).
- 1258 3. The Functional Element MUST provide the capability to enable Event Suppliers to trigger
1259 relevant Events.
- 1260 4. The Functional Element MUST provide a mechanism to associate/unassociate Routing
1261 Rules to an Event.
- 1262 *Example: As shown in Figure 1, where an event can be routed to Air Freight or Financial*
1263 *Channel or even to all channels based on the Routing Rules that are associated*
1264 *with the Event.*
- 1265 5. As part of Key Feature (3), the Routing Rules must be able to route an event to all, specified
1266 Channels or individual Event Consumers.
- 1267 6. The Functional Element MUST enable Event Consumers to execute the following tasks to
1268 improve the relevancy of the incoming events”
- 1269 6.1. Subscribe/Unsubscribe to relevant Channel(s), and
1270 6.2. Apply a filter to the appropriate channel or event, which helps to refine the criteria of a
1271 useful event further.
- 1272 7. The Functional Element MUST provide the capability to notify relevant Event Consumers
1273 when an event occurs.
- 1274 Examples of notification types include SMS, email and Web Services invocations.
- 1275 8. As part of Key Feature (6), the notification must be able to handle differing requirements
1276 arising from different notification formats.
- 1277 *Example: If the incoming event contains 2 important attributes, the order or position of*
1278 *these 2 attributes must be configurable to suit the convenience of the Event*
1279 *Consumer. This is extremely important in the case of Web Service Invocations.*
- 1280 10. The Functional Element MUST provide a mechanism for managing the concepts specified
1281 across different application domains.
- 1282 *Example: Namespace control mechanism*
1283

1284 In addition, the following key features could be provided to enhance the Functional Element
1285 further:

- 1286 1. The Functional Element MAY provide a mechanism to enable active event detection.
- 1287 2. If Key Feature (1) is implemented, then the Functional Element MUST provide the following
1288 capabilities also:
- 1289 2.1. Non-intrusive detection
1290 *Example: The detection of a new event through periodic inspection of the audit log.*
- 1291 2.2. Configurable event detection schedule
1292 *Example: To inspect the audit log every 2 hours, where the duration between*
1293 *inspections is configurable.*
- 1294 2.3. Ability to retrieve relevant data from external source(s) for further event processing by
1295 Event Handler
1296 *Example: To retrieve Error Type and Message from audit log.*
- 1297 3. The Functional Element MAY provide the capability to record event processing within the
1298 Event Handler. The logging of event processing includes the occurrences of event, sending
1299 of notifications, warning and error messages generated in the processing of events.
- 1300 4. The Functional Element MAY provide the capability scheduled-based event notification.
1301

1302 **2.3.4 Interdependencies**

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

1303

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

1304

1305 **2.3.5 Related Technologies and Standards**

1306 None

2.3.6 Model

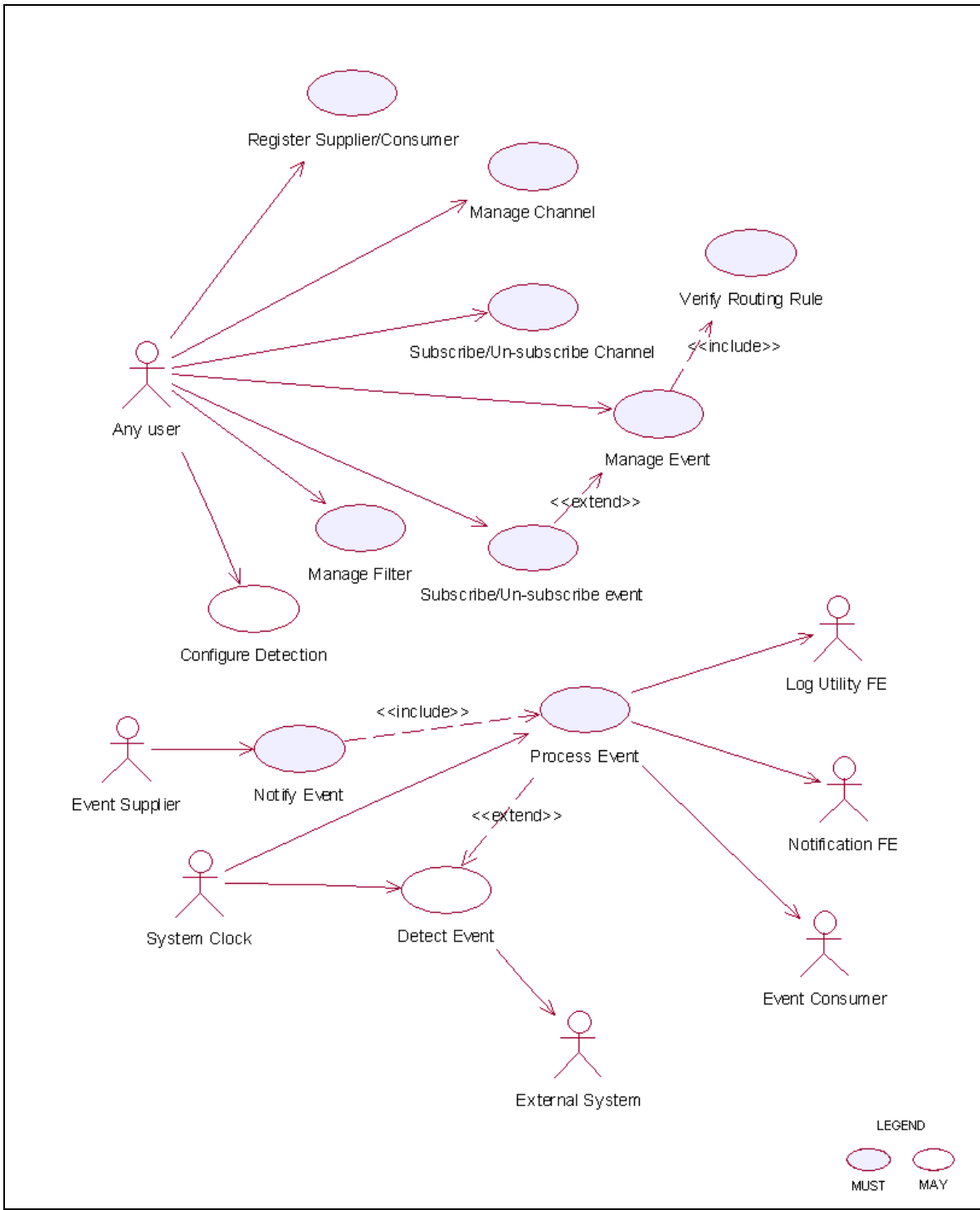


Figure 7: Model Of the Event Handler Functional Element [4]

1308 **2.3.7 Usage Scenarios**

1309 **2.3.7.1 Register Supplier/Consumer**

1310 **2.3.7.1.1 Description**

1311 This use case allows the user to register itself to the Event Handler Functional Element as an
1312 event supplier or an event consumer.

1313 **2.3.7.1.2 Flow of Events**

1314 **2.3.7.1.2.1 Basic Flow**

1315 The use case begins when the user of the Event Handler wants to register an event supplier or
1316 event consumer with the Event Handler.

1317 1: The user sends a request to Event Handler together with its profile data and operation.

1318 2: Based on the operation it specified, one of the following sub-flows is executed:

1319 If the operation is '**Register as supplier**', then sub-flow 2.1 is executed.

1320 If the operation is '**Register as consumer**', then sub-flow 2.2 is executed.

1321 If the operation is '**Un-register as supplier**', then sub-flow 2.3 is executed.

1322 If the operation is '**Un-register as consumer**', then sub-flow 2.4 is executed.

1323 If the operation is '**Update supplier**', then sub-flow 2.5 is executed.

1324 If the operation is '**Update consumer**', then sub-flow 2.6 is executed.

1325 If the operation is '**Retrieve supplier**', then sub-flow 2.7 is executed.

1326 If the operation is '**Retrieve consumer**', then sub-flow 2.8 is executed.

1327 2.1: Register as Supplier.

1328 2.1.1: The Functional Element gets the user profile data, i.e. namespace, name,
1329 description and type.

1330 2.1.2: The Functional Element registers the user as event supplier.

1331 2.1.3: The Functional Element returns the Supplier Id to the user.

1332 2.2: Register as Consumer.

1333 2.2.1: The Functional Element gets the user profile data, i.e. namespace, name,
1334 description and type.

1335 2.2.2: The Functional Element registers the user as event consumer.

1336 2.2.3: The Functional Element returns the Consumer Id to the user.

1337 2.3: Un-register as Supplier.

1338 2.3.1: The Functional Element gets the user namespace and name or User Id.

1339 2.3.2: The Functional Element checks whether the user is a supplier.

1340 2.3.3: The Functional Element removes the user as supplier.

- 1341 2.4: Un-register as Consumer.
- 1342 2.4.1: The Functional Element gets the user namespace and name or User Id.
- 1343 2.4.2: The Functional Element checks whether the user is a consumer.
- 1344 2.4.3: The Functional Element removes the user as consumer.
- 1345 2.5: Update Supplier.
- 1346 2.5.1: The Functional Element gets the user namespace and name or User Id together
1347 with the user profile.
- 1348 2.5.2: The Functional Element checks whether the user is a supplier.
- 1349 2.5.2: The Functional Element updates the user profile.
- 1350 2.6: Update Consumer.
- 1351 2.6.1: The Functional Element gets the user namespace and name or User Id together
1352 with the user profile.
- 1353 2.6.2: The Functional Element checks whether the user is a consumer.
- 1354 2.6.3: The Functional Element updates the user profile.
- 1355 2.7: Retrieve Supplier.
- 1356 2.7.1: The Functional Element gets the user namespace and name or User Id.
- 1357 2.7.2: The Functional Element checks whether the user is a supplier.
- 1358 2.7.3: The Functional Element returns the user profile.
- 1359 2.8: Retrieve Consumer.
- 1360 2.8.1: The Functional Element gets the user namespace and name or User Id.
- 1361 2.8.2: The Functional Element checks whether the user is a consumer.
- 1362 2.8.3: The Functional Element returns the user profile.
- 1363 3: The Functional Element returns the results to indicate the success or failure of this operation to
1364 the user and the use case ends.
- 1365 **2.3.7.1.2.2 Alternative Flows**
- 1366 1: Supplier Already Registered.
- 1367 1.1: If in the basic flow 2.1.2, the user already registered as supplier, Functional Element will
1368 return an error message to the user and the use case ends.
- 1369 2: Consumer Already Registered.
- 1370 2.1: If in the basic flow 2.2.2, the user already registered as consumer, Functional Element
1371 will return an error message to the user and the use case ends.
- 1372 3: Supplier or Consumer Not Registered.

1373 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
1374 registered, Functional Element will return an error message to the user and the use case
1375 ends.

1376 4: Persistency Mechanism Error.

1377 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
1378 perform data persistency, Functional Element will return an error message to the user and the
1379 use case ends.

1380

1381 **2.3.7.1.3 Special Requirements**

1382 None.

1383 **2.3.7.1.4 Pre-Conditions**

1384 None.

1385 **2.3.7.1.5 Post-Conditions**

1386 None.

1387 **2.3.7.2 Manage Channel**

1388 **2.3.7.2.1 Description**

1389 This use case allows the user to manage channels.

1390 **2.3.7.2.2 Flow of Events**

1391 **2.3.7.2.2.1 Basic Flow**

1392 The use case begins when the user wants to create/retrieve/update/delete a channel

1393 1: The user sends request to manipulate a channel.

1394 2: Based on the operation it specifies, one of the following sub-flows is executed:

1395 If the operation is '**Create Channel**', the sub-flow 2.1 is executed.

1396 If the operation is '**Retrieve Channel**', the sub-flow 2.2 is executed.

1397 If the operation is '**Update Channel**', the sub-flow 2.3 is executed.

1398 If the operation is '**Delete Channel**', the sub-flow 2.4 is executed.

1399 2.1: Create Channel.

1400 2.1.1: The Functional Element gets channel definition, i.e. namespace, channel name
1401 and description.

1402 2.1.2: The Functional Element checks whether the channel exists.

1403 2.1.3: The Functional Element creates the channel.

1404 2.2: Retrieve Channel.

1405 2.2.1: The Functional Element gets namespace, channel name and retrieve condition.

1406 2.2.2: The Functional Element retrieves the channel's information according to the
1407 condition.

1408 2.3: Update Channel.

1409 2.3.1: The Functional Element gets namespace, channel name and description.

1410 2.3.2: The Functional Element checks whether the channel exists.

1411 2.3.3: The Functional Element updates the channel definition.

1412 2.4: Delete Channel.

1413 2.4.1: The Functional Element gets namespace and channel name.

1414 2.4.2: The Functional Element checks whether the channel exists.

1415 2.4.3: The Functional Element removes the channel from the Functional Element.

1416 3: The Functional Element returns the results of the operation to the user and the use case ends.

1417 **2.3.7.2.2.2 Alternative Flows**

1418 1: Channel Already Exists.

1419 1.1: If in the basic flow 2.1.2, the channel is already defined, Functional Element returns an
1420 error message and the use case ends.

1421 2: Conditional Retrieving.

1422 2.1: In the basic flow 2.2.2:

1423 2.1 1: If the condition is the retrieval by channel name and the channel does not exist,
1424 then it will go to Alternative Flow 3.

1425 2.1.2: If the condition is the retrieval of one channel definition, it returns the definition of
1426 that channel and the use case ends.

1427 2.1.3: If the condition is the retrieval of all channels' information, it returns all channels
1428 definition and the use case ends.

1429 2.1.4: If the condition is the retrieval of channel through channel description, it will return
1430 all matched channels and the use case ends.

1431 2.1.5: If the condition is the retrieval of registered consumers, it returns the list of
1432 consumer registered on the channel and the use case ends.

1433 3: Channel Not Found.

1434 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the channel does not exist, Functional
1435 Element will return an error message and the use case ends.

1436 4: Consumer Not Found.

1437 4.1: If in the basic flow 2.1.3, 2.5.3 and 2.6.3, the event consumer does not exist,
1438 Functional Element will return an error message and the use case ends.

1439 5: Extension Point.

1440 5.1: If in the basic flow 2.1.3, and 2.3.3, the event consumers that subscribed to the
1441 channel are provided, the use case Subscribe/un-subscribe channel will be extended.

1442 **2.3.7.2.3 Special Requirements**

1443 None.

1444 **2.3.7.2.4 Pre-Conditions**

1445 None.

1446 **2.3.7.2.5 Post-Conditions**

1447 None.

1448 **2.3.7.3 Subscribe/Un-subscribe To Channel**

1449 **2.3.7.3.1 Description**

1450 This use case performs the subscription or un-subscription on a channel for an event consumer.

1451 **2.3.7.3.2 Flow of Events**

1452 **2.3.7.3.2.1 Basic Flow**

1453 The use case begins when the user wants to subscribe or un-subscribe to a channel.

1454 1: The user sends the request.

1455 2: Based on the operation it specifies, one of the following sub-flows is executed:

1456 If the operation is '**Subscribe to Channel**', then sub-flow 2.1 is executed.

1457 If the operation is '**Un-Subscribe to Channel**', then sub-flow 2.2 is executed.

1458 2.1: Subscribe To Channel.

1459 2.1.1: The Functional Element gets event consumer Id, or consumer namespace and
1460 consumer name, together with channel namespace and channel name.

1461 2.1.2: The Functional Element checks whether the channel exists.

1462 2.1.3: The Functional Element adds the subscription of the consumer to the channel.

1463 2.2: Un-Subscribe To Channel.

1464 2.2.1: The Functional Element gets event consumer Id, or consumer namespace and
1465 consumer name, together with channel namespace and channel name.

1466 2.2.2: The Functional Element checks whether the channel exists.

1467 2.2.3: The Functional Element removes the subscription of the consumer to the channel.

1468 3: The Functional Element returns the results of the operation to the user and the use case ends.

1469 **2.3.7.3.2.2 Alternative Flows**

1470 1: Channel Not Found.

1471 1.1: If in the basic flow 2.1.2 and 2.2.2, the channel specified does not exist, Functional
1472 Element will return an error message to the user and the use case ends.

1473 2: Event Consumer Not Found.

1474 2.1: If in the basic flow 2.1.2 and 2.2.2, the event consumer related does not exist, Functional
1475 Element will return an error message to the user and the use case ends.

1476 **2.3.7.3.3 Special Requirements**

1477 None.

1478 **2.3.7.3.4 Pre-Conditions**

1479 None.

1480 **2.3.7.3.5 Post-Conditions**

1481 None.

1482 **2.3.7.4 Manage Event**

1483 **2.3.7.4.1 Description**

1484 This use case describes the scenarios of managing events.

1485 **2.3.7.4.2 Flow of Events**

1486 **2.3.7.4.2.1 Basic Flow**

1487 The use case begins when the user wants to manage events.

1488 1: The user sends a request to the Functional Element.

1489 2: Based on the operation it specifies, one of the following sub-flows is executed:

1490 If the operation is '**Create Event**', then sub-flow 2.1 is executed.

1491 If the operation is '**Retrieve Event Information**', then sub-flow 2.2 is executed.

1492 If the operation is '**Update Event Definition**', then sub-flow 2.3 is executed.

1493 If the operation is '**Delete Event**', then sub-flow 2.4 is executed.

1494 If the operation is '**Assign Flow**', then sub-flow 2.5 is executed.

1495 If the operation is '**Un-Assign Flow**', then sub-flow 2.6 is executed.

1496 2.1: Create Event

1497 2.1.1: The Functional Element gets event definition including namespace, event name,
1498 event description, event routing rule, and event attributes definition.

1499 2.1.2: The Functional Element verifies the parameters.

1500 2.1.3: The Functional Element verifies the routing rule through use case verify routing
1501 rule.

1502 2.1.4: The Functional Element creates event definition by recording the definition of
1503 event.

- 1504 2.2: Retrieve Event.
- 1505 2.2.1: The Functional Element gets namespace, event name, and condition.
- 1506 2.2.2: The Functional Element retrieves the event definition according to the condition.
- 1507 2.3: Update Event Definition
- 1508 2.3.1: The Functional Element gets event definition including namespace, event name,
1509 event description, event routing rule, and event attributes definition.
- 1510 2.3.2: The Functional Element verifies the parameters.
- 1511 2.3.3: The Functional Element verifies the routing rule through use case verify routing
1512 rule.
- 1513 2.3.4: The Functional Element updates the event definition.
- 1514 2.4: Delete Event.
- 1515 2.4.1: The Functional Element gets namespace and event name.
- 1516 2.4.2: The Functional Element checks whether the event exists.
- 1517 2.4.3: The Functional Element deletes the event definition.
- 1518 2.5: Assign Flow.
- 1519 2.5.1: The Functional Element gets namespace, event name and flow name.
- 1520 2.5.2: The Functional Element checks whether the event exists and flow defined.
- 1521 2.5.3: The Functional Element assigns the flow to the event.
- 1522 2.6: Un-assign Flow.
- 1523 2.6.1: The Functional Element gets namespace, event name and flow name.
- 1524 2.6.2: The Functional Element checks whether the event exists and flow defined.
- 1525 2.6.3: The Functional Element un-assigns the flow to the event.
- 1526 3: The Functional Element returns the results of the operation to the user and the use case ends.
- 1527 **2.3.7.4.2.2 Alternative Flows**
- 1528 1: Event Already Exist.
- 1529 1.1: If in the basic flow 2.1.2, the event already exists, Functional Element will return an error
1530 message to the user and the use case ends.
- 1531 2: Parameters Are Invalid.
- 1532 2.1: If in the basic flow 2.1.2 and 2.3.2, the parameters provided are invalid, Functional
1533 Element will return an error message to the user and the use case ends.
- 1534 3: Event Not Found.
- 1535 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the event does not exist, Functional Element
1536 will return an error message to the user and the use case ends.

- 1537 4: Flow Not Defined.
- 1538 4.1: If in the basic flow 2.1.2 and 2.3.2, the flow does not exist, Functional Element will return
1539 an error message to the user and the use case ends.
- 1540 5: Condition Retrieve.
- 1541 5.1: In the basic flow 2.2.2:
- 1542 5.1.1: If the retrieving condition is the retrieval of event definition based on event name, it
1543 returns event definition and the use case ends.
- 1544 5.1.2: If the retrieving condition is the retrieval of all event definition, it returns all event
1545 definition and the use case ends.
- 1546 5.1.3: If the retrieving condition is the retrieval of events assigned to specified channel, it
1547 returns the list of event definitions.
- 1548 5.1.4: If the retrieving condition is the retrieval of channels associated with specified
1549 event, it returns the list of channel definition.
- 1550 6: Extension Point.
- 1551 6.1: If in the basic flow 2.1.4, and 2.3.4, the event consumers that subscribed to the event are
1552 provided, the use case Subscribe/Un-subscribe event will be extended.

1553 **2.3.7.4.3 Special Requirements**

1554 None.

1555 **2.3.7.4.4 Pre-Conditions**

1556 None.

1557 **2.3.7.4.5 Post-Conditions**

1558 None.

1559 **2.3.7.5 Subscribe/Un-subscribe To Event**

1560 **2.3.7.5.1 Description**

1561 This use case performs the subscription or un-subscription on an event for an event consumer.

1562 **2.3.7.5.2 Flow of Events**

1563 **2.3.7.5.2.1 Basic Flow**

1564 The use case begins when the user wants to subscribe or un-subscribe an event.

1565 1: The user sends a request.

1566 2: Based on the operation it specifies, one of the following sub-flows is executed:

1567 If the operation is '**Subscribe to Event**', then sub-flow 2.1 is executed.

1568 If the operation is '**Un-Subscribe to Event**', then sub-flow 2.2 is executed.

1569 2.1: Subscribe To Event.

1570 2.1.1: The Functional Element gets event consumer Id, or consumer namespace and
1571 consumer name, together with event namespace and event name.

1572 2.1.2: The Functional Element checks whether the event exists.

1573 2.1.3: The Functional Element adds the subscription of the consumer to the event.

1574 2.2: Un-Subscribe To Event.

1575 2.2.1: The Functional Element gets event consumer Id, or consumer namespace and
1576 consumer name, together with event namespace and event name.

1577 2.2.2: The Functional Element checks whether the event exists.

1578 2.2.3: The Functional Element removes the subscription of the consumer to the event.

1579 3: The Functional Element returns the results of the operation to the user and the use case ends.

1580 **2.3.7.5.2 Alternative Flows**

1581 1: Event Not Found.

1582 1.1: If in the basic flow 2.1.2 and 2.2.2, the event specified does not exist, Functional Element
1583 will return an error message to the user and the use case ends.

1584 2: Event Consumer Not Found.

1585 2.1: If in the basic flow 2.1.2 and 2.2.2, the event consumer related does not exist, Functional
1586 Element will return an error message to the user and the use case ends.

1587 **2.3.7.5.3 Special Requirements**

1588 None.

1589 **2.3.7.5.4 Pre-Conditions**

1590 None.

1591 **2.3.7.5.5 Post-Conditions**

1592 None.

1593 **2.3.7.6 Verify Routing Rule**

1594 **2.3.7.6.1 Description**

1595 This use case verifies the syntax of routing rule.

1596 **2.3.7.6.2 Flow of Events**

1597 **2.3.7.6.2.1 Basic Flow**

1598 The use case begins when the user wants to verify the correctness of a routing expression.

1599 1: The user sends a request.

1600 2: The Functional Element gets the routing expression.

- 1601 3: The Functional Element checks the syntax of routing expression.
1602 4: The Functional Element verifies the parameters.
1603 5: The Functional Element returns the status of the operation to the user and the use case ends.

1604 **2.3.7.6.2 Alternative Flows**

- 1605 1: Routing Rule Expression Syntax Error.
1606 1.1: If in the basic flow 3, there is a syntax error, Functional Element will return an error
1607 message to the user and the use case ends.
1608 2: Event Consumer Not Found.
1609 2.1: If in the basic flow 4, the event consumer related does not exist, Functional Element will
1610 return an error message to the user and the use case ends.

1611 **2.3.7.6.3 Special Requirements**

1612 None.

1613 **2.3.7.6.4 Pre-Conditions**

1614 None.

1615 **2.3.7.6.5 Post-Conditions**

1616 None.

1617 **2.3.7.7 Manage Filter**

1618 **2.3.7.7.1 Description**

1619 A filter is used to filter out certain events to those event consumers even though they are the
1620 intended receivers according to the routing rules.

1621 **2.3.7.7.2 Flow of Events**

1622 **2.3.7.7.2.1 Basic Flow**

1623 The use case begins when the user wants to create/retrieve/update/delete a filter.

- 1624 1: The user sends a request to manage a filter.
1625 2: Based on the operation it specifies, one of the following sub-flows is executed:
1626 If the operation is '**Create Filter**', then sub-flow 2.1 is executed.
1627 If the operation is '**Retrieve Filter**', then sub-flow 2.2 s executed.
1628 If the operation is '**Update Filter**', then sub-flow 2.3 is executed.
1629 If the operation is '**Delete Filter**', then sub-flow 2.4 is executed.

1630 2.1: Create Filter.

1631 2.1.1: The Functional Element gets filter definition, i.e. consumer namespace, consumer
1632 name, filter name, description, event name or channel name.

1633 2.1.2: The Functional Element checks whether the event or channel exists.
1634 2.1.3: The Functional Element saves the filter definition.
1635 2.2: Retrieve Filter.
1636 2.2.1: The Functional Element gets the filter name.
1637 2.2.2: The Functional Element retrieves the filter information according to the name.
1638 2.3: Update Filter.
1639 2.3.1: The Functional Element gets filter definition, i.e. consumer namespace, name, filter
1640 name, description, event name or channel name.
1641 2.3.2: The Functional Element checks the parameters.
1642 2.3.3: The Functional Element updates the filter definition.
1643 2.4: Delete Filter.
1644 2.4.1: The Functional Element gets namespace and filter name.
1645 2.4.2: The Functional Element checks whether the filter exists.
1646 2.4.3: The Functional Element removes the filter from the Functional Element.
1647 3: The Functional Element returns the results of the operation to the user and the use case ends.

1648 **2.3.7.7.2 Alternative Flows**

1649 1: Filter Already Exists.
1650 1.1: If in the basic flow 2.1.2, the filter is already defined, Functional Element will return an
1651 error message and the use case ends.
1652 2: Event Not Found.
1653 2.1: If in the basic flow 2.1.2 and 2.3.2, the event used does not exist, Functional Element will
1654 return an error message and the use case ends.
1655 3: Channel Not Found.
1656 3.1: If in the basic flow 2.1.2 and 2.3.2, the channel used does not exist, Functional Element
1657 will return an error message and the use case ends.
1658 4: Consumer Not Found.
1659 4.1: If in the basic flow 2.1.3, 2.5.3, and 2.6.3, the event consumer does not exist, Functional
1660 Element will return an error message and the use case ends.

1661 **2.3.7.7.3 Special Requirements**

1662 None.

1663 **2.3.7.7.4 Pre-Conditions**

1664 None.

1665 **2.3.7.7.5 Post-Conditions**

1666 None.

1667 **2.3.7.8 Notify Event**

1668 **2.3.7.8.1 Description**

1669 This use case allows the event supplier to notify an event to the Event Handler Functional
1670 Element. Once the Event Handler Functional Element receives the notification, it will process the
1671 event based on the processing logic defined.

1672 **2.3.7.8.2 Flow of Events**

1673 **2.3.7.8.2.1 Basic Flow**

1674 The use case begins when the user wants to notify an event.

1675 1: The user sends a notification.

1676 2: The Functional Element receives the notification with parameters, i.e. event supplier id or event
1677 supplier namespace and name.

1678 3: The Functional Element checks whether the event is defined and event supplier is registered.

1679 4: Include use case Process Event to process the notification of event.

1680 5: The Functional Element returns the status of the operation to the user and the use case ends.

1681 **2.3.7.8.2.2 Alternative Flows**

1682 1: User Is Not Registered.

1683 1.1: If in the basic flow 3, the user is not registered, Functional Element will return an error
1684 message to the user and the use case ends.

1685 2: Event Not Defined.

1686 2.1: If in the basic flow 3, the event is not defined, Functional Element will return an error
1687 message to the user and the use case ends.

1688 3: Error Returned.

1689 3.1: If in the basic flow 4, an error is returned by use case Process event, Functional Element
1690 will return an error message to the user and the use case ends.

1691 **2.3.7.8.3 Special Requirements**

1692 None.

1693 **2.3.7.8.4 Pre-Conditions**

1694 None.

1695 **2.3.7.8.5 Post-Conditions**

1696 None.

1697 **2.3.7.9 Configure Monitoring**

1698 **2.3.7.9.1 Description**

1699 This use case describes the capability of configuration on event monitoring. Based on the
1700 configuration, Event Handler will pro-actively check whether an event has happened.

1701 **2.3.7.9.2 Flow of Events**

1702 **2.3.7.9.2.1 Basic Flow**

1703 The use case begins when the user wants to configure the event monitoring.

1704 1: The user sends a request to manage a filter.

1705 2: Based on the operation it specifies, one of the following sub-flows is executed:

1706 If the operation is '**Add Configuration**', then sub-flow 2.1 is executed.

1707 If the operation is '**Remove Configuration**', then sub-flow 2.2 is executed.

1708 2.1: Add Configuration.

1709 2.1.1: The Functional Element gets configuration definition, i.e. configuration name,
1710 namespace, event name, connection parameters, condition that signifies the events and
1711 schedule.

1712 2.1.2: The Functional Element saves filter definition.

1713 2.2: Remove Configuration.

1714 2.2.1: The Functional Element gets configuration name.

1715 2.2.2: The Functional Element removes the configuration.

1716 3: The Functional Element returns the results of the operation to the user and the use case ends.

1717 **2.3.7.9.2.2 Alternative Flows**

1718 1: Configuration Exist.

1719 1.1: If in the basic flow 2.1.2, the configuration already exists, Functional Element will return
1720 an error message and the use case ends.

1721 **2.3.7.9.3 Special Requirements**

1722 None.

1723 **2.3.7.9.4 Pre-Conditions**

1724 None.

1725 **2.3.7.9.5 Post-Conditions**

1726 None.

1727 **2.3.7.10 Detect Event**

1728 **2.3.7.10.1 Description**

1729 This use case describes the event monitoring capability that Event Handler provides. Once Event
1730 Handler detects an event, it will trigger the pre-defined process for the event.

1731 **2.3.7.10.2 Flow of Events**

1732 **2.3.7.10.2.1 Basic Flow**

1733 The use case begins when the Functional Element clock generates the trigger.

1734 1: The Functional Element clock generates a trigger.

1735 2: The Functional Element receives the trigger and checks the condition for pre-defined
1736 monitoring sources.

1737 3: The Functional Element checks whether the event happens.

1738 4: The Functional Element returns the results of the operation and the use case ends.

1739 **2.3.7.10.2.2 Alternative Flows**

1740 1: External Functional Element Not Available.

1741 1.1: If in the basic flow 3, the external Functional Element is not available and the Event
1742 Handler cannot make a connection, Functional Element will return an error message and the
1743 use case ends.

1744 2: Data Not Available.

1745 2.1: If in the basic flow 3, the data that signifies the event cannot be accessed, Functional
1746 Element will return an error message and the use case ends.

1747 3: Extension Point.

1748 3.1: If in the basic flow 3, the event happens, Functional Element will extend to use case
1749 Process event.

1750 **2.3.7.10.3 Special Requirements**

1751 None.

1752 **2.3.7.10.4 Pre-Conditions**

1753 None.

1754 **2.3.7.10.5 Post-Conditions**

1755 None.

1756 **2.3.7.11 Process Event**

1757 **2.3.7.11.1 Description**

1758 This use case describes the core functionality of Event Handler. It is the engine that processes
1759 the events. Actor can be the Functional Element clock that triggers the scheduled event
1760 notification, or any user who wants to notify the event.

1761 **2.3.7.11.2 Flow of Events**

1762 **2.3.7.11.2.1 Basic Flow**

1763 The use case begins when there is a request to process the event.

1764 1: The user sends a request to process an event.

1765 2: Based on the actor of this use case, one of the sub-flows is executed.

1766 If the initiator is the Functional Element clock, then sub-flow '**Initiated By Functional Element**
1767 **Clock**' is executed.

1768 If the initiator is other than Functional Element clock, then sub-flow '**Initiated By Any User**' is
1769 executed.

1770 2.1: Initiated By Functional Element Clock.

1771 2.1.1: The Functional Element looks up scheduled events defined to find out time-due
1772 notification.

1773 2.1.2: The Functional Element retrieves the routing rule for the event.

1774 2.1.3: The Functional Element looks up the corresponding consumers based on the
1775 routing rule.

1776 2.1.4: The Functional Element retrieves filters defined and find out the event receivers.

1777 2.1.5: The Functional Element notifies or invokes the event consumers based on the
1778 routing rule defined.

1779 2.2: Initiated By Any User.

1780 2.2.1: The Functional Element retrieves the routing rule for the event.

1781 2.2.2: The Functional Element looks up the corresponding consumers.

1782 2.2.3: The Functional Element retrieves filters defined and find out the event receivers.

1783 2.2.4: The Functional Element notifies or invokes the event consumers based on the
1784 routing rule defined.

1785 3: The Functional Element logs the notification of event and the use case ends.

1786 **2.3.7.11.2.2 Alternative Flows**

1787 1: Notify Event.

1788 In basic flow 2.1.4 and 2.2.4, based on the type of consumer, one of the sub-flows is execute.

1789 If the consumer type is '**SMTP**', then sub-flow Notify via SMTP is executed.

1790 If the consumer type is '**SMS Gateway**', then sub-flow Notify via SMS Gateway is executed.

1791 If the consumer type is '**Notify RPC-Web Service**', then sub-flow Notify RPC-Web Service is
1792 executed.

1793 If the consumer type is '**Notify Document Style Web Service**' then sub-flow Notify Document
1794 style Web Service is executed.

1795 1.1: Notify via SMTP.

1796 1.1.1: The Functional Element gets the pre-defined message for event and forms the
1797 parameters.

1798 1.1.2: The Functional Element gets the parameters for SMTP server.

1799 1.1.3: The Functional Element sends out the pre-defined message and the use case
1800 ends.

1801 1.2: Notify via SMS Gateway.

1802 1.2.1: The Functional Element gets the pre-defined message for event and forms the
1803 parameters.

1804 1.2.2: The Functional Element gets the parameters for the SMS gateway.

1805 1.2.3: The Functional Element sends out the pre-defined message and the use case
1806 ends.

1807 1.3: Notify RPC-Web Service.

1808 1.3.1: The Functional Element gets the operation parameter.

1809 1.3.2: The Functional Element gets Web Services endpoint parameters.

1810 1.3.3: The Functional Element dynamically invokes the Web Service and the use case
1811 ends.

1812 1.4: Notify Document Style Web Service.

1813 1.4.1: The Functional Element gets the operation parameter.

1814 1.4.2: The Functional Element gets Web Services endpoint parameters.

1815 1.4.3: The Functional Element dynamically generates the SOAP message and sends to
1816 the Web Services and the use case ends.

1817 2: Flow Is Defined.

1818 If in the basic flow 2.1.2 and 2.2.1, a flow is defined for the event, Functional Element will perform
1819 the following steps:

1820 2.1: The Functional Element retrieves all the intended event consumers defined in the flow.

1821 2.2: The Functional Element will go to basic flow 2.2.

1822 2.3: The Functional Element will resume the execution from basic flow 2.1.2 or 2.2.1.

1823 3: Log Utility Not Available.

1824 3.1: If in the basic flow 3, the Log Utility Functional Element is not available, Functional
1825 Element will return an error message to the user and the use case ends.

- 1826 4: SMS Gateway Not Available.
- 1827 4.1: If in the Alternative Flow 1.2.3, the SMS Gateway is not available, Functional Element will
1828 return an error message to the user and the use case ends.
- 1829 5: SMTP Server Not Available.
- 1830 5.1: If in the Alternative Flow 1.1.3, the SMTP server is not available, Functional Element will
1831 return an error message to the user and the use case ends.
- 1832 6: RPC Web Service Not Available.
- 1833 6.1: If in the Alternative Flow 1.3.3, the Web Service is not available, Functional Element will
1834 return an error message to the user and the use case ends.
- 1835 7: Document Style Web Service Not Available.
- 1836 7.1: If in the Alternative Flow 1.4.3, document style Web Service is not available, Functional
1837 Element will return an error message to the user and the use case ends.
- 1838 **2.3.7.11.3 Special Requirements**
- 1839 **2.3.7.11.3.1 Supportability**
- 1840 The application server used must have a JMS service provided.
- 1841 **2.3.7.11.4 Pre-Conditions**
- 1842 None.
- 1843 **2.3.7.11.5 Post-Conditions**
- 1844 None.
- 1845

1846 **2.4 Group Management Functional Element**

1847 **2.4.1 Motivation**

1848 The Group Management Functional Element is expected to be an integral part of the User Access
1849 Management (UAM) functionalities. In a Web Service-enabled implementation, this Functional
1850 Element helps to provide the mechanism to manage users in a collective manner. This is
1851 important as it provides the flexibility of adopting either coarse or fine-grain access controls, or
1852 both.

1853

1854 This Functional Element fulfills the following requirements from the Functional Elements
1855 Requirements Document 02:

- 1856 • Primary Requirements
 - 1857 ○ MANAGEMENT-050 to MANAGEMENT-053, and
 - 1858 ○ MANAGEMENT-078
- 1859 • Secondary Requirements
 - 1860 ○ None

1861 **2.4.2 Terms Used**

Terms	Description
Group	A Group is a collection of individual users, and are typically grouped together as they have certain commonalities
Namespace	Namespace is use to segregate the instantiation of the application across different application domains. If a company has two separate standalone application, for example, an email application and an equipment booking application, then these two are considered as separate application domains.
User	A user is loosely defined to include both human and virtual users. Virtual users could include service users and application (or machine) users that are utilising other services in a SOA environment.
User Access Management / UAM	User Access Management or UAM refer to the concept of managing users in a holistic manner, considering all aspect which includes: Defining a set of basic user information that should be stored in any enterprise application. Providing a means to extend this basic set of user information when needed. Simplifying management by grouping related users together through certain criteria. Having the flexibility of adopting both coarse and fine grain access controls.

1862

1863

1864

2.4.3 Key Features

1865

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

1866

1867

1. The Functional Element MUST provide a basic Group structure with a set of pre-defined attributes.

1868

1869

2. The Functional Element MUST provide the capability to extend on the basic Group structure dynamically.

1870

1871

3. As part of Key Feature (2), this dynamic extension MUST be definable and configurable at runtime implementation of the Functional Element.

1872

1873

4. The Functional Element MUST provide the capability to manage the creation and deletion of instances of Groups based on defined structure.

1874

1875

5. The Functional Element MUST provide the capability to manage all the information (attribute values) stored in such Groups. This includes the capability to retrieve and update attribute's values belonging to a Group.

1876

1877

1878

5. The Functional Element MUST provide a mechanism to manage the collection of users in a Group. This includes the capability to create, retrieve, update and delete users belonging to a Group.

1879

1880

1881

6. The Functional Element MUST provide a mechanism for managing Groups across different application domains.

1882

1883

Example: Namespace control mechanism

1884

1885

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

1887

1. The Functional Element MAY provide a mechanism to enable different Groups to be related to one another.

1888

1889

2. The Functional Element MAY also provide a mechanism to enable hierarchical relationships between Groups.

1890

1891

Example: Parent and Child Relationship.

1892

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

1893

1894

Example: Adding of Group "Dept-A" to "Company-XYZ" – "Dept-A" is a Group, and also part of the collection of Group "Company-XYZ".

1895

1896

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

1897

1898

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

1899

1900

1901

2.4.4 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.

1902

1903 **2.4.5 Related Technologies and Standards**

1904 None.

1905

1906 **2.4.6 Model**

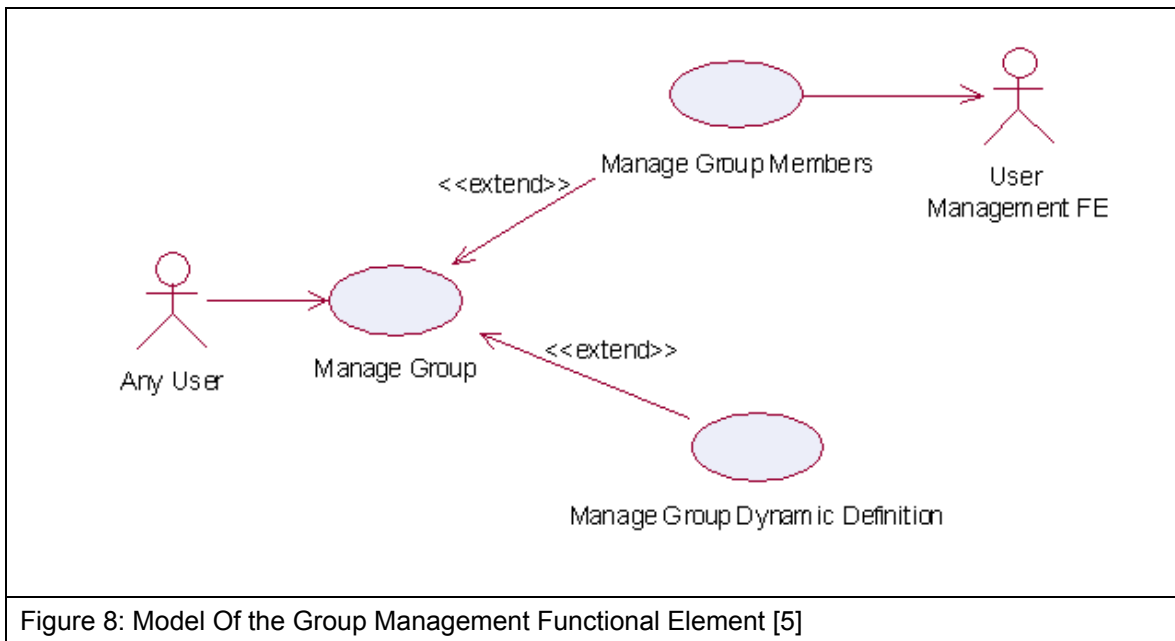


Figure 8: Model Of the Group Management Functional Element [5]

1907

1908 **2.4.7 Usage Scenarios**

1909 **2.4.7.1 Manage Group**

1910 This use case describes the management of a group, namely the creation, deletion, retrieval and
1911 update of the group.

1912 **2.4.7.1.1 Flow of Events**

1913 **2.4.7.1.1.1 Basic Flow**

1914 This use case starts when the user wants to manage group.

1915 If user wants to '**Create Group**', then basic flow 1 is executed.

1916 If user wants to '**Retrieve Group**', then basic flow 2 is executed.

1917 If user wants to '**Update Group**', then basic flow 3 is executed.

1918 If user wants to '**Delete Group**', then basic flow 4 is executed.

1919 1: Create Group.

1920 1.1: User provides the basic information that is necessary for creating a group.

1921 1.2: Functional Element creates the group and the use case ends.

- 1922 2: Retrieve Group.
- 1923 2.1: User provides the necessary information for retrieving the complete group's attributes.
- 1924 2.2: Functional Element returns the group's information and the use case ends.
- 1925 3: Update Group.
- 1926 3.1: User provides the necessary information for updating the group's attributes.
- 1927 3.2: Functional Element updates the group and the use case ends.
- 1928 4: Delete Group.
- 1929 4.1: User provides the necessary information for deleting a particular group.
- 1930 4.2: Functional Element deletes the group and the use case ends.

1931 **2.4.7.1.1.2 Alternative Flows**

- 1932 1: Group Exist.
- 1933 1.1: In basic flow 1.2, Functional Element detects an identical group. Functional Element
1934 returns an error message and the use case ends.
- 1935 2: Group Does Not Exist.
- 1936 2.1: In basic flow 2.2, 3.2 and 4.2, Functional Element cannot find a group that matches the
1937 user's criteria. Functional Element returns an error message and the use case ends.
- 1938 3: Save Updated Information.
- 1939 3.1: In basic flow 1.2, 2.2, 3.2 and 4.2, Functional Element fails to save the updated
1940 information. Functional Element returns an error message and the use case ends.

1941 **2.4.7.1.2 Special Requirements**

1942 None.

1943 **2.4.7.1.3 Pre-Conditions**

1944 None.

1945 **2.4.7.1.4 Post-Conditions**

1946 None.

1947 **2.4.7.2 Manage Group Members**

1948 **2.4.7.2.1 Description**

1949 This use case is an extension of the manage group use case. Specifically, it describes the
1950 scenarios to manage members in the group.

1951 **2.4.7.2.2 Flow of Events**

1952 **2.4.7.2.2.1 Basic Flow**

1953 This use case starts when the user wants to manage members in a group.

1954 If user wants to 'Create Members In A Group', then basic flow 1 is executed.

1955 If user wants to 'Retrieve Members From A Group', then basic flow 2 is executed.

1956 If user wants to 'Delete Members From A Group', then basic flow 3 is executed.

1957 1: Create Members In A Group.

1958 1.1: User provides the necessary information for retrieving the group.

1959 1.2: Functional Element adds members to the group and the use case ends.

1960 2: Retrieve Members In A Group.

1961 2.1: User provides the necessary information for retrieving the group.

1962 2.2: Functional Element returns the members and the use case ends.

1963 3: Delete Members From Group.

1964 3.1: User provides the necessary information for retrieving the group.

1965 3.2: User provides the necessary information for deleting members in the group.

1966 3.3: Functional Element deletes members from group and the use case ends.

1967 **2.4.7.2.2.2 Alternative Flows**

1968 1: Group Does Not Exist.

1969 1.1: In basic flow 1.1, 2.1 and 3.1, Functional Element cannot find the group requested.
1970 Functional Element returns an error message and the use case ends.

1971 2: Members Does Not Exist

1972 2.1: In basic flow 3.3, the Functional Element attempts to delete a non-existence member.
1973 Functional Element returns an error message and the use case ends.

1974 **2.4.7.2.3 Special Requirements**

1975 None.

1976 **2.4.7.2.4 Pre-Conditions**

1977 None.

1978 **2.4.7.2.5 Post-Conditions**

1979 None.

1980 **2.4.7.3 Manage Group Dynamic Definition**

1981 **2.4.7.3.1 Description**

1982 This use case describes scenario involved in managing the dynamic group definition.

1983 **2.4.7.3.2 Flow of Events**

1984 **2.4.7.3.2.1 Basic Flow**

1985 This use case starts when the user wants to manage dynamic group definition. This include
1986 create, retrieve, update and delete dynamic group definition.

1987 If user wants to '**Create Dynamic Definition For A Group**', then basic flow 1 is executed.

1988 If user wants to '**Retrieve Dynamic Definition For A Group**', then basic flow 2 is executed.

1989 If user wants to '**Delete Dynamic Definition For A Group**', then basic flow 3 is executed.

1990 If user wants to '**Update Dynamic Definition For A Group**', then basic flow 4 is executed.

1991

1992 1: Create Dynamic Definition For A Group.

1993 1.1: User provides the additional definition for the group.

1994 1.2: Functional Element creates the additional definition for the group and the use case ends.

1995 2: Retrieve Dynamic Definition For A Group.

1996 2.1: User provides the necessary information to retrieve a particular group.

1997 2.2: Functional Element returns the additional definition for the group and the use case ends.

1998 3: Delete Dynamic Definition For Group.

1999 3.1: User provides the necessary information to retrieve a particular group.

2000 3.2: Functional Element deletes the dynamic definition belonging to the group and the use
2001 case ends.

2002 4: Update Dynamic Definition For Group.

2003 4.1: User provides the necessary information to retrieve a particular group.

2004 4.2: User provides the necessary dynamic definition that needs to be updated.

2005 4.3: Functional Element update the dynamic definition and the use case ends.

2006 **2.4.7.3.2.2 Alternative Flows**

2007 1: Group Does Not Exist.

2008 1.1: In basic flow 1.1, 2.1, 3.1 and 4.1, Functional Element cannot find the group specified.
2009 Functional Element returns an error message and the use case ends.

2010 2: Dynamic Group Definition Already Exists.

2011 2.1: In basic flow 1.2, Functional Element returns the error message and the use case ends.

2012 3: Dynamic Group Definition Does Not Exist.

2013 3.1: In basic flow 4.3, Functional Element cannot update the dynamic group definition.
2014 Functional Element returns an error message and the use case ends.

2015 **2.4.7.3.3 Special Requirements**

2016 None.

2017 **2.4.7.3.4 Pre-Conditions**

2018 None.

2019 **2.4.7.3.5 Post-Conditions**

2020 None.

2021 **2.5 Identity Management Functional Element**

2022 **2.5.1 Motivation**

2023 As secured Web Services become rampant, with each having its own authentication and
2024 authorisation management, users are finding it difficult to keep track of their accounts and
2025 passwords. Through the use of Identity Management, users can now voluntarily establish links
2026 between their accounts so that they need not sign in multiple times to access enterprise-level
2027 Web Services. This mechanism is known as Single Sign-On (SSO). SSO can further be extended
2028 to access Web Services from across different business organisations that have prior agreements
2029 to trust and transact with each other (also known as a circle of trust). This mechanism, which
2030 involves federating and signing-in of identity's accounts across different trusted organisations, is
2031 known as Federated Identity Single Sign-On.

2032
2033 Identity Management is about the management of information pertaining to an entity as well as
2034 the process of identification, authentication and authorization of resources to that entity.

- 2035 • Identity management generally covers the following aspects:
- 2036 • Basic user accounts management facilities
 - 2037 • User authentication mechanism(s)
 - 2038 • User authorisation mechanism(s)
 - 2039 • Generation of audit trails for user activities

2040
2041 This Functional Element fulfills the following requirements from the Functional Elements
2042 Requirements Document 02:

- 2043 • Primary Requirements
 - 2044 ○ SECURITY-001,
 - 2045 ○ SECURITY-003 (all),
 - 2046 ○ SECURITY -004 (all),
 - 2047 ○ SECURITY -040 and
 - 2048 ○ SECURITY -041.
- 2049 • Secondary Requirements
 - 2050 ○ None

2051

2052 **2.5.2 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.

2053

2054 The following terms mentioned in this document are used in accordance with the terms defined in
2055 the Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) v1.1
2056 specification:

- 2057 • Assertion [section 2.3.2]
 - 2058 • AudienceRestrictionCondition [section 2.3.2.1.3]
 - 2059 • AuthenticationQuery [section 3.3.3]
 - 2060 • AuthenticationStatement [section 2.4.3]
 - 2061 • KeyInfo [section 5.4.5]
 - 2062 • Request [section 3.2.2]
 - 2063 • Response [section 3.4.2]
 - 2064 • Subject [section 2.4.2.1]
- 2065

2066

2.5.3 Key Features

2067

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

2068

2069

7. The Functional Element MUST be have the mechanism to access an Identity Repository.

2070

8. The Functional Element MUST provide the capability to manage the creation and deletion of instances of Identity in the said Identity Repository.

2071

2072

9. The Functional Element MUST have the mechanisms to manage all the information (attribute values) stored in such Identities. This includes the capability to:

2073

2074

4.1. Retrieve and update attribute's values belonging to a Identity,

2075

4.2. Encrypt sensitive user information,

2076

4.3. Authenticate a user, and

2077

4.4. Assign/Unassign Access Policy (or Policies).

2078

Example: Different levels of privileges to access protected resources.

2079

10. As part of Key Feature (3.3), the authentication of an Identity MUST be achieved at least through the use of a password.

2080

2081

11. As part of Key Feature (3.3), the Functional Element MUST also provide the capability to use an Assertion Authority for Single Sign-On (SSO) authentication.

2082

2083

12. As part of Key Feature (5), the SSO message exchange and protocol MUST use an approved standard. Recommendations are available in section 2.5.5.

2084

2085

13. As part of Key Feature (3.4), a mechanism MUST be provided to verify the Identity's Access Policy on protected Resources.

2086

2087

14. The Functional Element MUST provide the capability to create audit trails.

2088

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

2089

2090

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

2091

2092

1. The Functional Element MAY provide an Identity Repository.

2093

2. If Key Feature (1) is provided, the Functional Element MUST provide the capability to manage the creation and deletion of instances of Identities based on a pre-defined structure.

2094

2095

3. The Functional Element MAY provide additional storage in the Identity Repository for an Identity to customise its preferences.

2096

2097

Example: Identity's preferred subscription of notifications/alerts for news.

2098

4. The Functional Element MAY provide a capability to use an Identity Provider for Federated Identity SSO authentication.

2099

2100

5. If Key Feature (4) is provided, the Federated Identity SSO message exchange and protocol MUST use an approved standard.

2101

2102

2103

2.5.4 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.
--------------------------------	--

2104

2105 **2.5.5 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 2004, 01 March 2004
Security Assertion Markup Language (SAML) v1.1. [7]	<p>Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1 – OASIS Standard, 2 September 2003</p> <p>Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1 – OASIS Standard, 2 September 2003, in particular the two schemas below:</p> <ul style="list-style-type: none"> • Assertion Schema • Protocol Schema
Liberty Alliance Project Specifications	<p>Liberty Alliance ID-FF 1.2 Specifications [8]</p> <p>Liberty Alliance ID-WSF 1.0 Specifications [9]</p>
WS-Federation [10]	Web Services Federation Language (WS-Federation) - 08 July 2003

2106

2107

2.5.6 Model

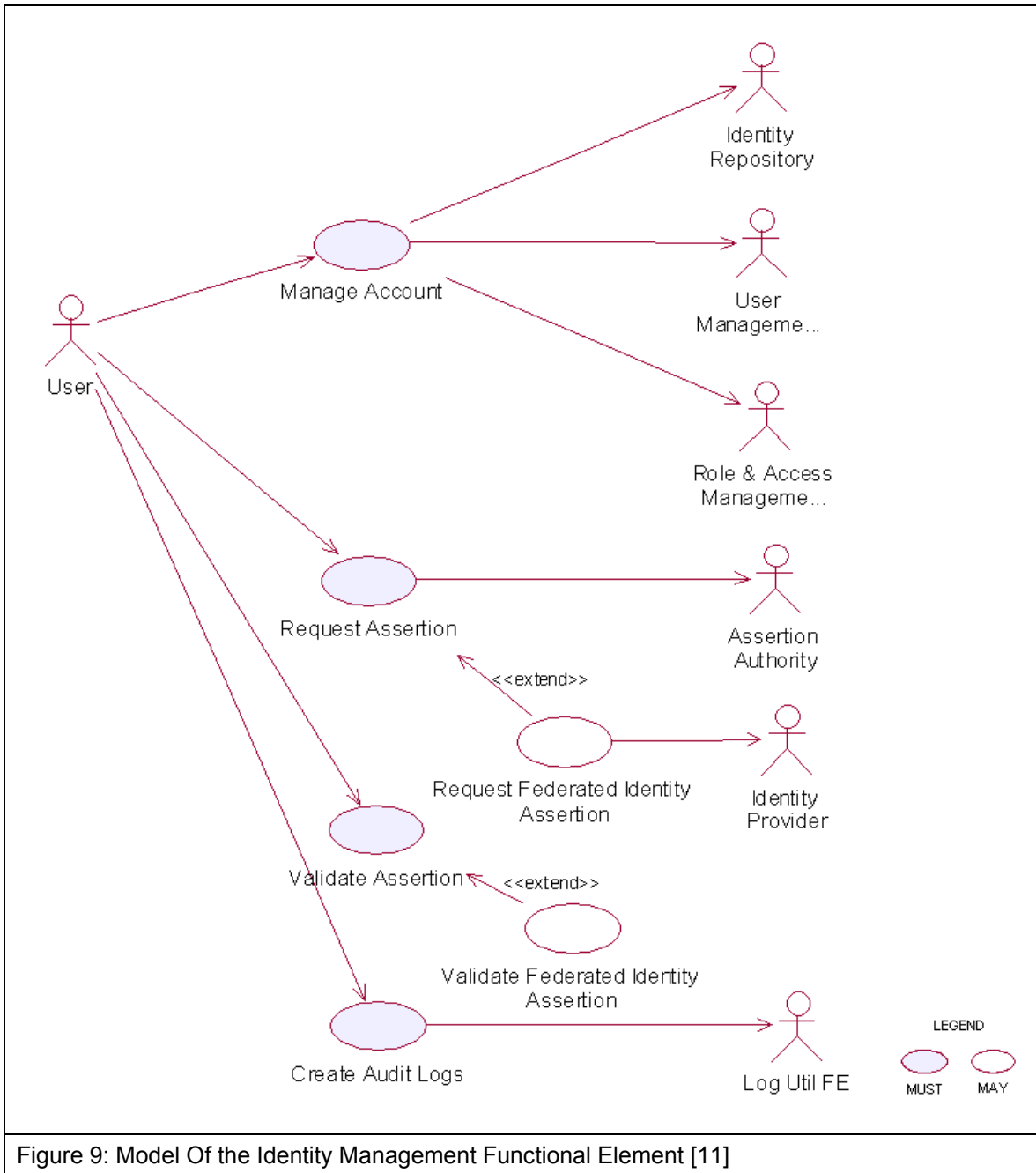


Figure 9: Model Of the Identity Management Functional Element [11]

2110 **2.5.7 Usage Scenarios**

2111 **2.5.7.1 Manage Account**

2112 **2.5.7.1.1 Description**

2113 This use case describes the creation/retrieval/update/deletion of an identity's account. An
2114 identity's account usually consists of two elements: i) the user information and ii) the associated
2115 access policy.

2116 As Identity Management Functional Element leverages on the User Management Functional
2117 Element and Role and Access Management Functional Element to provide for these
2118 functionalities, please refer to these Functional Elements' use cases for details.

2119 **2.5.7.2 Request Assertion**

2120 **2.5.7.2.1 Description**

2121 This use case describes the composition of either 1) an authentication query or 2) an
2122 authorisation decision query and sending it to the assertion authority.

2123 **2.5.7.2.2 Flow of Events**

2124 **2.5.7.2.2.1 Basic Flow**

2125 This use case starts when the user wants to compose a query to the assertion authority.

2126 If the user requests for an authentication query, then sub-flow 1 is executed.

2127 If the user requests for an authorisation decision query, then sub-flow 2 is executed.

2128 1: Request for Authentication Assertion

2129 1.1: The user composes a valid SAML Request with an AuthenticationQuery and sends it to
2130 the assertion authority.

2131 1.2: The user waits for an SAML Response from the assertion authority.

2132 1.3: The user obtains the SAML Assertion from the SAML Response and use case ends.

2133 2: Request for Authorisation Decision Assertion

2134 2.1: The user composes a valid SAML Request with an AuthorizationDecisionQuery and
2135 sends it to the assertion authority.

2136 2.2: The user waits for an SAML Response from the assertion authority.

2137 2.3: The user obtains the SAML Assertion from the SAML Response and use case ends.

2138 **2.5.7.2.2.2 Alternative Flows**

2139 1: Invalid Request

2140 1.1: If in basic flow 1.1 or 2.1, if any of the parameters passed into the request is invalid, the
2141 Functional Element flag an exception and use case ends.

2142 2: Error message from assertion authority

2143 2.1: If in basic flow 1.3 or 2.3, the assertion authority is unable to return an assertion (e.g.
2144 user has not logged on etc.), it returns an error code and an error message.

2145 2.2: The Functional Element flag an error with the error message attached and use case
2146 ends.

2147 **2.5.7.2.3 Special Requirements**

2148 None.

2149 **2.5.7.2.4 Pre-Conditions**

2150 None.

2151 **2.5.7.2.5 Post-Conditions**

2152 None.

2153 **2.5.7.3 Validate Assertion**

2154 **2.5.7.3.1 Description**

2155 This use case describes the validation of either 1) the Authentication Assertion or 2) the
2156 Authorisation Decision Assertion

2157 **2.5.7.3.2 Flow of Events**

2158 **2.5.7.3.2.1 Basic Flow**

2159 This use case starts when the user wants to check if the assertion it is a valid assertion from the
2160 assertion authority.

2161 1: The user passes the assertion to the Functional Element for validation.

2162 2: The Functional Element checks if the assertion is signed by the assertion authority.

2163 3: The Functional Element checks for an un-expired assertion.

2164 4: The Functional Element checks if the assertion has an AudienceRestrictionCondition and
2165 verifies that the service provider using the Functional Element is in the audience list.

2166 5: Based on the type of assertion, one of the sub-flows is executed.

2167 • If the user wants to check for a valid authentication assertion, then sub-flow 5.1 is executed.

2168 • If the user wants to check for a valid authorisation decision assertion, then sub-flow 5.2 is
2169 executed.

2170 5.1: Validate Authentication Statement

2171 5.1.1: The Functional Element checks if the assertion has indeed an
2172 AuthenticationStatement.

2173 5.1.2: The Functional Element checks if the Subject in the AuthenticationStatement
2174 matches the userid of the principal.

2175 5.1.3: The Functional Element verifies the Subject with its KeyInfo.

- 2176 5.1.4: The Functional Element returns the status code to the user and use case ends.
- 2177 5.2: Validate Authorisation Decision Statement
- 2178 5.2.1: The Functional Element checks if the assertion has indeed an
2179 AuthorizationDecisionStatement.
- 2180 5.2.2: The Functional Element checks if the Resource in the
2181 AuthorizationDecisionStatement matches the id of the requested resource.
- 2182 5.2.3: The Functional Element determines if the decision is Permit.
- 2183 5.2.4: The Functional Element returns the status code to the user and use case ends.
- 2184 **2.5.7.3.2 Alternative Flows**
- 2185 1: Signature Error
- 2186 1.1: If in basic flow 2, the Functional Element is unable to verify that the signature is from the
2187 assertion authority, it returns an error and use case ends.
- 2188 2: Expired Assertion
- 2189 2.1: If in basic flow 3, the Functional Element finds that the assertion has already expired, it
2190 returns an error and use case ends.
- 2191 3: Audience Error
- 2192 3.1: If in basic flow 4, the service provider is not in the AudienceRestrictionCondition, the
2193 Functional Element returns an error and use case ends.
- 2194 4: Invalid Authentication Assertion
- 2195 4.1: If in basic flow 5.1.1, the Functional Element is unable to find an
2196 AuthenticationStatement, it returns an error and use case ends.
- 2197 5: Mismatch Subject
- 2198 5.1: If in basic flow 5.1.2, the Functional Element is unable to match the Subject in
2199 AuthenticationStatement, it returns an error and use case ends.
- 2200 6: Subject Error
- 2201 6.1: If in basic flow 5.1.3, the Functional Element is unable to verify the Subject with the
2202 KeyInfo, it returns an error and use case ends.
- 2203 7: Invalid Authorisation Decision Assertion
- 2204 7.1: If in basic flow 5.2.1, the Functional Element is unable to find an
2205 AuthorizationDecisionStatement, it returns an error and use case ends.
- 2206 8: Mismatch Resource
- 2207 8.1: If in basic flow 5.2.2, the Functional Element is unable to match the resource in
2208 AuthorizationDecisionStatement, it returns an error and use case ends.
- 2209 **2.5.7.3.3 Special Requirements**
- 2210 None.

- 2211 **2.5.7.3.4 Pre-Conditions**
- 2212 None.
- 2213 **2.5.7.3.5 Post-Conditions**
- 2214 None.
- 2215 **2.5.7.4 Create Audit Logs**
- 2216 **2.5.7.4.1 Description**
- 2217 This use case describes logging all identity management activities for audit purposes.
- 2218 **2.5.7.4.2 Flow of Events**
- 2219 **2.5.7.4.2.1 Basic Flow**
- 2220 This use case starts when any of other Functional Element use cases are triggered.
- 2221 1: The Functional Element opens an audit log file.
- 2222 2: The Functional Element writes a timestamp identity management activity message into the
2223 audit log file.
- 2224 3: The Functional Element closes the audit log file and the use case ends.
- 2225 **2.5.7.4.2.2 Alternative Flows**
- 2226 1: Log File Not Created
- 2227 1.1: If in the basic flow 1, the Functional Element cannot open the audit file, it creates a new
2228 audit file and use case continues.
- 2229 2: Error Writing Log
- 2230 2.1: If in the basic flow 2, the Functional Element has error writing to file, it will flag an
2231 exception and the use case ends.
- 2232 **2.5.7.4.3 Special Requirements**
- 2233 None.
- 2234 **2.5.7.4.4 Pre-Conditions**
- 2235 None.
- 2236 **2.5.7.4.5 Post-Conditions**
- 2237 None.

2238 **2.6 Information Catalogue Functional Element (new)**

2239 **2.6.1 Motivation**

2240 There is a huge amount of information that is stored in the WWW that include product catalogues.
2241 Enable the capability to provide a generic facility to quickly and easily expose catalogues and/or
2242 orders as web services. Eg. Amazon.com Web Service, Google.com Web Service, etc.

2243 Provide a framework that will enable the ability to harness and access huge amount of product-
2244 related information and present them as catalogue for:

- 2245 • Quick and easy definition of product/information catalogues
- 2246 • Customisation of catalogues for specific needs or marketing niche
- 2247 • Easy maintenance of storefronts/catalogues over the network
- 2248 • Outsourcing of catalogue management together with multilingual support

2249

2250 This Functional Element fulfills the following requirements from the Functional Elements
2251 Requirements Document 02:

- 2252 • Primary Requirements
 - 2253 ○ PROCESS-200,
 - 2254 ○ PROCESS-201, and
 - 2255 ○ PROCESS-202.
- 2256 • Secondary Requirements
 - 2257 ○ PROCESS-203,
 - 2258 ○ PROCESS-204,
 - 2259 ○ PROCESS-205, and
 - 2260 ○ PROCESS-206.

2261

2262 **2.6.2 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.
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

2263 **2.6.3 Key Features**

2264 Implementations of the Information Catalogue Functional Element are expected to provide the
2265 following key features:

- 2266 1. The Functional Element MUST provide the capability to *define and maintain Catalogue*
2267 *Structures*.
 - 2268 1.1. The capability to define the name for the catalogue structure
 - 2269 1.2. The capability to *define the format* of the catalogue information
 - 2270 1.3. The capability to *choose the data source* to store and retrieve the catalogue information
- 2271 2. The Functional Element MUST provide the capability to *organize and manage all the*
2272 *information* stored in the Catalogue Structures.

- 2273 3. The Functional Element MUST provide the capability to *execute basic searches* like
 2274 categorical, names, keywords on the catalogue information.
 2275 4. The Functional Element MUST provide the capability to return results formatted based on the
 2276 Catalogue Structure.

2277

2278 In addition, the following key features could be provided to enhance the Information Catalogue
 2279 Functional Element further:

- 2280 1. The Functional Element MAY provide the ability to enable secured access to catalogue
 2281 structure as well as catalogue information.
 2282 2. The Functional Element MAY provide the ability to present catalogue information in different
 2283 languages, i.e. multi-lingual support.
 2284 3. The Functional Element MAY provide the ability to import catalogue structure and information
 2285 from different data sources.
 2286 4. The Functional Element MAY provide the ability to export catalogue structure and information
 2287 to different data sources.

2288

2289 **2.6.4 Interdependencies**

Direct Dependency	
Search Functional Element	The Search Functional Element helps to perform basic search on the catalogue information.

2290

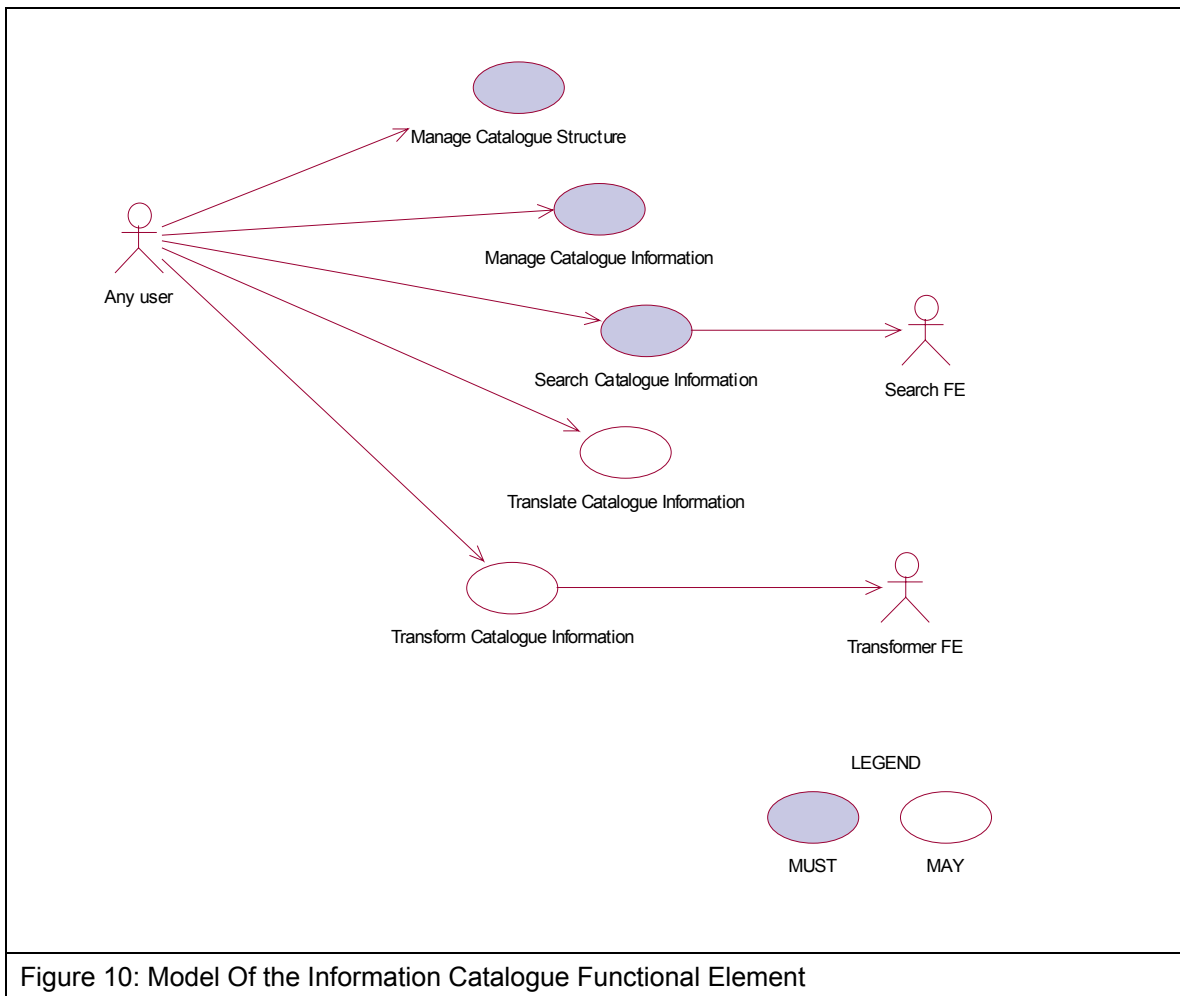
Interaction Dependency	
User Management Functional Element	The User Management Functional Element helps to provide user definition and management.
Role & Access Functional Element	The Role & Access Functional Element helps to provide role and access definition and management.
Transformer Functional Element	The Transformer Functional Element helps to provide the import and export catalogue information capabilities.

2291 **2.6.5 Related Technologies and Standards**

2292 None

2293

2.6.6 Model



2295

2296 2.6.7 Usage Scenario

2297 2.6.7.1 Manage Catalogue Structure

2298 2.6.7.1.1 Description

2299 This use case allows any users to configure and manage various data source(s), type(s) and
2300 structure(s) on which information is to be stored and retrieved.

2301 2.6.7.1.2 Flow of Events

2302 2.6.7.1.2.1 Basic Flow

2303 This use case starts when users / other Functional Elements wishes to configure and manage
2304 various data source(s), type(s) and structure(s).

2305 1. Users / Other Functional Elements initiates a request to configure data source, type and
2306 structure by providing name, format, and definition of the data source(s) to be added, removed or
2307 retrieved.

- 2308 2. The Functional Element checks whether the data source configuration file exists.
- 2309 3. Based on the operation it specified, one of the following sub-flows is executed:
- 2310 If the operation is '**Create Data Source, Type and Structure**', then sub-flow 3.1 is executed.
- 2311 If the operation is '**View Data Source, Type and Structure**', then sub-flow 3.2 is executed.
- 2312 If the operation is '**Remove Data Source, Type and Structure**', then sub-flow 3.3 is executed.
- 2313 3.1. Create Data Source, Type and Structure.
- 2314 3.1.1. The Functional Element checks whether the same data source, type, and structure
- 2315 has been created.
- 2316 3.1.2. The Functional Element appends the new data source, type and structure in the
- 2317 data source configuration specified.
- 2318 3.2. View Data Source, Type and Structure.
- 2319 3.2.1. The Functional Element retrieves all the data source, type and structure
- 2320 information from the data source configuration file.
- 2321 3.2.2. The Functional Element returns the data source(s), type(s) and structure(s).
- 2322 3.3. Delete Data Source, Type and Structure.
- 2323 3.3.1. The Functional Element checks whether the data source, type and structure exist
- 2324 in the data source configuration based on data source id from the data source
- 2325 configuration file.
- 2326 3.3.2. The Functional Element removes the old data source, type and structure from the
- 2327 data source configuration file.
- 2328 4. The Functional Element returns a success or failure flag indicating the status of the operation
- 2329 being performed and use case ends.

2330 **2.6.7.1.2.2 Alternative Flows**

- 2331 1. Data Source Configuration File Not Found.
- 2332 1.1. If in Basic Flow 2, the data source configuration does not exist, Functional Element
- 2333 creates empty data source configuration.
- 2334 2. Duplicate Data Source, Type and Structure.
- 2335 2.1. If in Sub Flow 3.1.1, the same data source, type and structure have been defined already
- 2336 in data source configuration, Functional Element throws an exception with error code as
- 2337 'Duplicate Data Source, Type, and Structure'.
- 2338 3. Data Source, Type, and Structure Do Not Exist.
- 2339 3.1. If in Sub Flow 3.2.1 and 3.3.1, a particular data source, type and structure cannot be
- 2340 found in the specified data source configuration, Functional Element throws an exception with
- 2341 error code as 'Data Source, Type, and Structure does not exist'.

2342 **2.6.7.1.3 Special Requirements**

- 2343 None.

2344 **2.6.7.1.4 Pre-Conditions**

2345 None.

2346 **2.6.7.1.5 Post-Conditions**

2347 None.

2348 **2.6.7.2 Manage Catalogue Information**

2349 **2.6.7.2.1 Description**

2350 This use case describes the management of catalogue information, namely the creation, deletion,
2351 retrieval and update of the catalogue information.

2352 **2.6.7.2.2 Flow of Events**

2353 **2.6.7.2.2.1 Basic Flow**

2354 The use case begins when the user wants to create/view/update/delete catalogue information.

2355 1. The user sends request to manipulate catalogue information.

2356 2. Based on the operation it specifies, one of the following sub-flows is executed:

2357 If the operation is '**Create Catalogue Information**', the sub-flow 2.1 is executed.

2358 If the operation is '**View Catalogue Information**', the sub-flow 2.2 is executed.

2359 If the operation is '**Update Catalogue Information**', the sub-flow 2.3 is executed.

2360 If the operation is '**Delete Catalogue Information**', the sub-flow 2.4 is executed.

2361 2.1. Create Catalogue Information

2362 2.1.1. User provides the basic information that is necessary for creating catalogue
2363 information.

2364 2.1.2. The Functional Element checks whether the catalogue information exists.

2365 2.1.3. The Functional Element creates the catalogue.

2366 2.2. View Catalogue Information

2367 2.2.1. User provides the necessary information for retrieving the complete catalogue's
2368 attributes.

2369 2.2.2. The Functional Element checks whether the catalogue information exists.

2370 2.2.3. The Functional Element returns the catalogue's information.

2371 2.3. Update Catalogue Information

2372 2.3.1. User provides the necessary information for updating the catalogue's attributes.

2373 2.3.2. The Functional Element checks whether the catalogue information exists.

2374 2.3.3. The Functional Element updates the catalogue.

2375 2.4. Delete Catalogue Information

- 2376 2.4.1. User provides the necessary information for deleting particular catalogue
2377 information.
- 2378 2.4.2. The Functional Element checks whether the catalogue information exists.
- 2379 2.4.3. Functional Element deletes the catalogue information.
- 2380 **2.6.7.2.2 Alternative Flows**
- 2381 1. Catalogue Information Exist.
- 2382 1.1. In Sub Flow 2.1.2, Function Element detects an identical catalogue information.
2383 Functional Element returns an error message and the use case ends.
- 2384 2. Catalogue Information Does Not Exist.
- 2385 2.1. In Sub Flow 2.2.2, 2.3.2, and 2.4.2, Functional Element cannot find the catalogue
2386 information that matches the user's criteria. Functional Element returns an error message
2387 and the use case ends.
- 2388 3. Save Updated Catalogue Information.
- 2389 3.1. In Sub Flow 2.1.3, 2.2.3, 2.3.3, and 2.4.3, Functional Element fails to save the updated
2390 catalogue information. Functional Element returns an error message and the use case ends.
- 2391 **2.6.7.2.3 Special Requirements**
- 2392 None.
- 2393 **2.6.7.2.4 Pre-Conditions**
- 2394 None.
- 2395 **2.6.7.2.5 Post-Conditions**
- 2396 None.
- 2397 **2.6.7.3 Search Catalogue Information**
- 2398 **2.6.7.3.1 Description**
- 2399 This use case allows any users to perform search on various types of disparate catalogues that
2400 are configured to be searched and returns the matching results.
- 2401 **2.6.7.3.2 Flow of Events**
- 2402 **2.6.7.3.2.1 Basic Flow**
- 2403 This use case starts when users / other Functional Elements wishes to perform information
2404 search on any given catalogue.
- 2405 1. Users / other Functional Elements initiates a request to perform information search on a given
2406 catalogue by providing information to be searched, the catalogue type(s) and the catalogue
2407 structure(s).
- 2408 2. The Functional Element checks for the existence of the specified catalogue type(s) and
2409 structure(s).

- 2410 3. The Functional Element validates the catalogue type(s) and structure(s) against the set of
2411 supported data type(s) and structure(s) configured within the Functional Element that are
2412 available for information search.
- 2413 4. The Functional Element performs information search based on the search parameters given by
2414 the users or the other Functional Elements.
- 2415 5. The Functional Element returns the result of the information search performed to the users or
2416 other Functional Elements and use case ends.

2417 **2.6.7.3.2.2 Alternative Flows**

- 2418 1. Catalogue(s) Are Not Available.
- 2419 1.1. In Basic Flow 2, if the identified catalogue is not available, Functional Element displays
2420 an error message and exits the use case.
- 2421 2. Invalid Catalogue Type and Structure.
- 2422 2.1. In Basic Flow 3, if the catalogue type and structure are invalid, Functional Element
2423 displays catalogue type and structure failure message and prompts for the data source type
2424 and structure again and performs another search.
- 2425 3. No Matching Result.
- 2426 3.1. In Basic Flow 4, if the search results in no matching results, Functional Element displays
2427 a message "No search results found" and performs another search.

2428 **2.6.7.3.3 Special Requirements**

2429 None.

2430 **2.6.7.3.4 Pre-Conditions**

2431 None.

2432 **2.6.7.3.5 Post-Conditions**

2433 None.

2434 **2.6.7.4 Translate Catalogue Information**

2435 **2.6.7.4.1 Description**

2436 This use case allows the user to translate a catalogue information file from one language to
2437 another language.

2438 **2.6.7.4.2 Flow of Events**

2439 **2.6.7.4.2.1 Basic Flow**

2440 This use case starts when a user wants to translate a catalogue information file from one
2441 language to another language.

- 2442 1. The user set the file name to be translated and the destination language.
- 2443 2. The system checks whether the particular destination language as output can be translated
2444 within all the supported translation methods available.

- 2445 4. Select the appropriate method based on the destination language.
- 2446 5. Invoke the translate method and save the catalogue information which is translated in that
2447 particular destination language.
- 2448 6: Return the results and the use case ends.
- 2449 **2.6.7.4.2.2 Alternative Flows**
- 2450 1. If in Basic Flow 2 there is no method to do the translation, the system return error message to
2451 the user and this use case ends.
- 2452 **2.6.7.4.3 Special Requirements**
- 2453 None.
- 2454 **2.6.7.4.4 Pre-Conditions**
- 2455 None.
- 2456 **2.6.7.4.5 Post-Conditions**
- 2457 None.
2458
- 2459 **2.6.7.5 Transform Catalogue Information**
- 2460 **2.6.7.5.1 Description**
- 2461 This use case allows the user to transform a catalogue information file from one format to another
2462 format.
- 2463 **2.6.7.5.2 Flow of Events**
- 2464 **2.6.7.5.2.1 Basic Flow**
- 2465 This use case starts when a user wants to transform a catalogue information file from one format
2466 to another format.
- 2467 1. The user set the file name to be transformed and the destination format.
- 2468 2. This use case call the TRANSFORMER functional elements' transform flow.
- 2469 3. Return the results from the transformer functional elements' transform flow and the use case
2470 ends.
- 2471 **2.6.7.5.2.2 Alternative Flows**
- 2472 1. If in Basic Flow 2 there is no method to do the transformation, the system return error message
2473 to the user and this use case ends.
- 2474 **2.6.7.5.3 Special Requirements**
- 2475 None.

2476 **2.6.7.5.4** **Pre-Conditions**

2477 None.

2478 **2.6.7.5.5** **Post-Conditions**

2479 None.

2480

2481 **2.7 Information Reporting Functional Element (new)**

2482 **2.7.1 Motivation**

2483 Information reporting is quite common in enterprise applications nowadays. In many scenarios,
2484 an enterprise does need to present its business information to, for example, business partners,
2485 sales representatives, and customers, in some form of information reporting. An information
2486 report is filled with the data that is retrieved from a data source using some type of queries. Such
2487 kind of information reporting is also used internally within an enterprise, or even within an
2488 individual department, to verify the business performance and other business scenarios.
2489

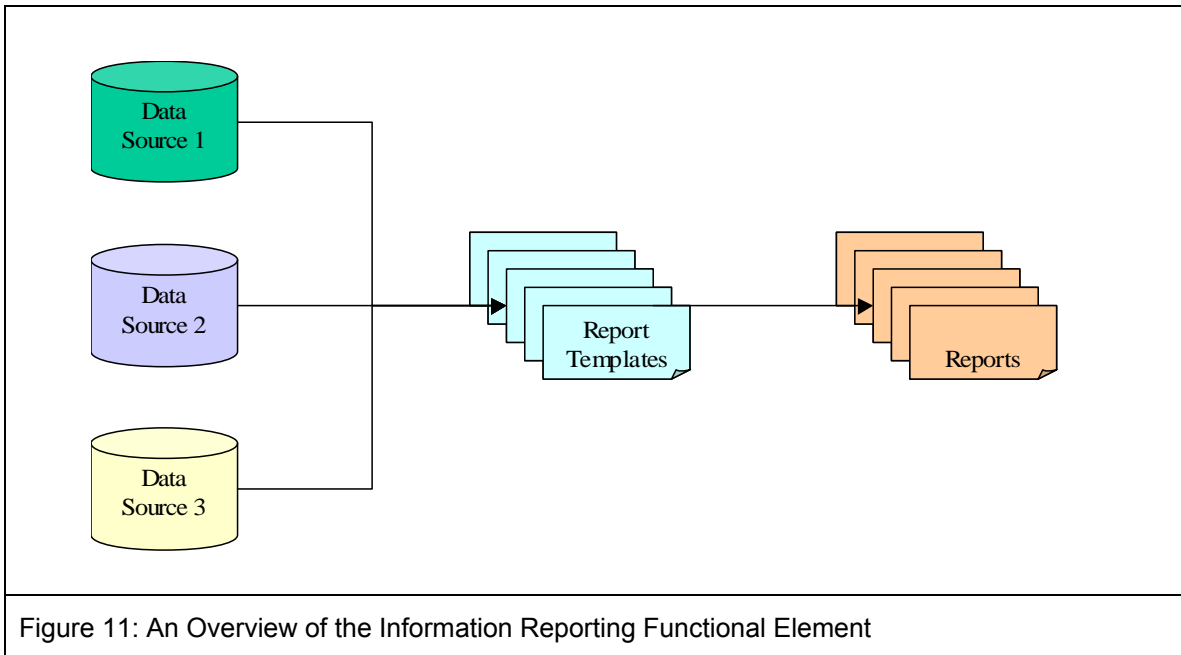


Figure 11: An Overview of the Information Reporting Functional Element

2490

2491 This Functional Element aims to provide the core features of information reporting solution to be
2492 used in general enterprise applications. It fulfills the following requirements from the Functional
2493 Elements Requirements Document 02:

- 2494 • Primary Requirements:
 - 2495 ○ DELIVERY-100,
 - 2496 ○ DELIVERY-101,
 - 2497 ○ DELIVERY-102,
 - 2498 ○ DELIVERY-103, and
 - 2499 ○ DELIVERY-104.
- 2500 • Secondary Requirements:
 - 2501 ○ DELIVERY-105, and
 - 2502 ○ DELIVERY-106.

2503
2504
2505
2506
2507

2508
2509

2.7.2 Terms Used

Terms	Description
Data source	A Data Source refers to any kind of information storage and retrieval databases like RDBMS, LDAP, ODBMS, XMLDB, XML Files, TEXT Files, etc.
Query	A query refers to a predefined method to query a data source to retrieve information stored in that data source. An example is SQL SELECT statement, which is used to retrieve information from a relational database.
Report Template	A report template is a document (such as an XML file) that is used to describe or show the report format and related settings.

2510

2.7.3 Key Features

2512 Implementations of the Information Reporting Functional Element are expected to provide the
2513 following key features:

- 2514 1. The Functional Element MUST provide an approach to capture the report templates and
2515 provide the guidelines how to secure the report templates.
- 2516 2. The Functional Element MUST be able to generate reports in the format defined by report
2517 templates.
- 2518 3. The Functional Element MUST provide a way to specify data sources where information is
2519 retrieved to fill out the generated reports.
- 2520 4. The Functional Element MUST provide an approach to capture user-defined queries, and
2521 MUST be able to execute user-defined queries to retrieve information to fill out the generated
2522 reports.
- 2523 5. The Functional Element MUST be able to store and retrieve generated reports as stated in
2524 key feature #2.
- 2525 6. The Functional Element MUST provide a security approach to control report access. A
2526 considered approach is to use user, role, and access management.

2527

2528 In addition, the following key features could be provided to enhance the Information Reporting
2529 Functional Element further:

- 2530 1. The Functional Element MAY provide an approach, such as an IDE, to design report
2531 templates.
- 2532 2. The Functional Element MAY provide the capability to export reports to different electronic
2533 file formats.
- 2534 3. The Functional Element MAY provide the capability to log the activities of report access.
- 2535 4. The Functional Element MAY allow the users to subscribe to the reports they want to
2536 receive.

2537

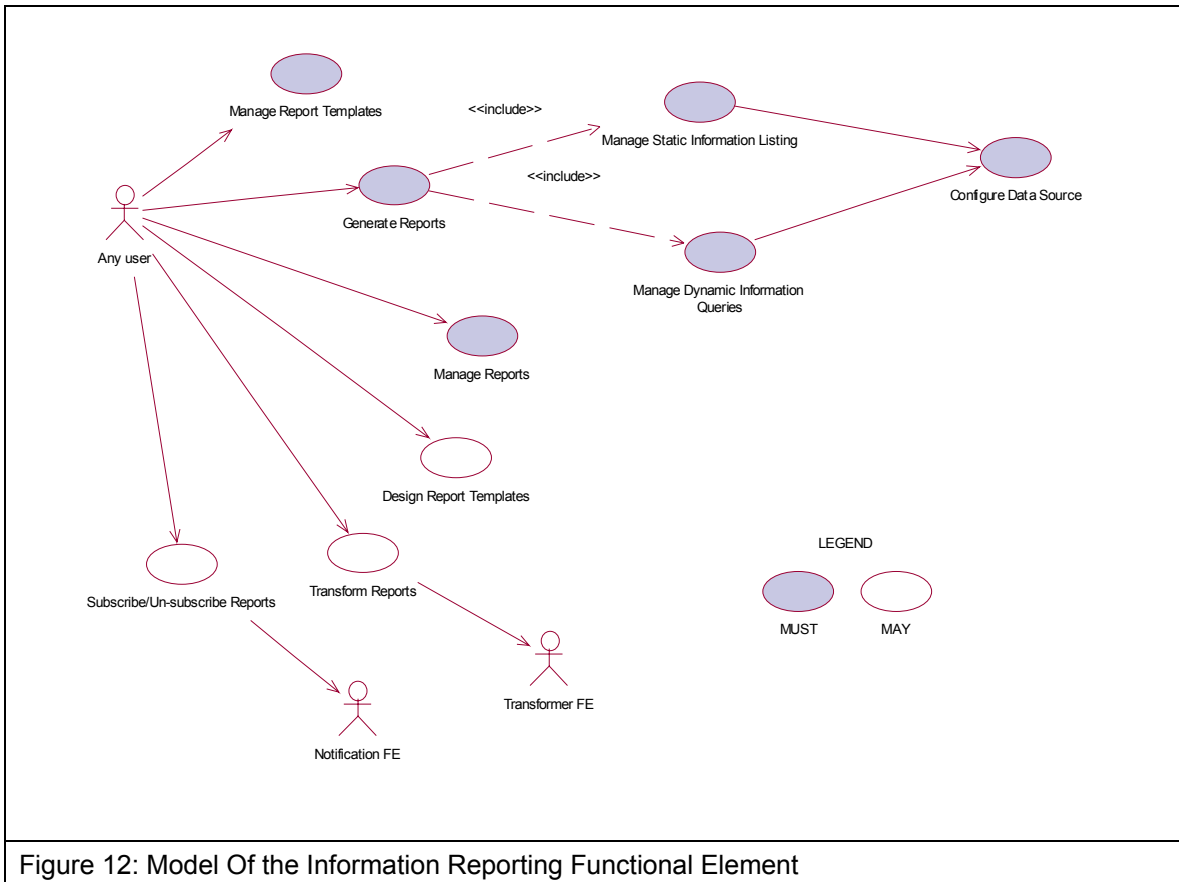
2.7.4 Interdependencies

Interaction Dependency	
Transformer Functional Element	The Transformer Functional Element helps to provide the import and export report information capabilities.

2539 **2.7.5 Related Technologies and Standards**

2540 None.

2541

2542 **2.7.6 Model**

2543

2544 **2.7.7 Usage Scenario**2545 **2.7.7.1 Manage Report Templates**2546 **2.7.7.1.1 Description**

2547 This use case allows any users to create, update, remove and view reporting templates.

2548 **2.7.7.1.2 Flow of Events**2549 **2.7.7.1.2.1 Basic Flow**

2550 The use case begins when the user wants to create/view/update/delete reporting templates.

2551 1: Any user initiates a request type to the Functional Element stating whether to create, view,
2552 update, or delete reporting templates.

2553 2: The Functional Element checks whether the reporting template exists.

2554 3: Based on the operation it specified, one of the following sub-flows is executed:

2555 • If the operation is 'Create Reporting Template', then sub-flow 3.1 is executed.

2556 • If the operation is 'View Reporting Template', then sub-flow 3.2 is executed.

2557 • If the operation is 'Update Reporting Template', then sub-flow 3.3 is executed.

2558 • If the operation is 'Delete Reporting Template', then sub-flow 3.4 is executed.

2559 3.1: Create Reporting Template.

2560 3.1.1: Any user provides reporting template information to be created.

2561 3.1.2: The Functional Element checks for the duplicate reporting template information.

2562 3.1.3: The Functional Element creates the reporting template information, if it does not
2563 exist and the use case ends.

2564 3.2: View Reporting Template.

2565 3.2.1: The Functional Element retrieves all the reporting templates.

2566 3.2.2: The Functional Element returns the reporting template information to any user and
2567 the use case ends.

2568 3.3: Update Reporting Template.

2569 3.3.1: Any user provides reporting template information to be updated.

2570 3.3.2: The Functional Element checks for the availability of reporting template
2571 information.

2572 3.3.3: The Functional Element updates the reporting template information, if it exist and
2573 the use case ends.

2574 3.4: Delete Reporting Template.

2575 3.4.1: Any user provides reporting template information to be removed.

2576 3.4.2: The Functional Element removes the reporting template information.

2577 4: The Functional Element responses the status of the operation whether it is successful or failure
2578 to any user and the use case ends.

2579 **2.7.7.1.2.2 Alternative Flows**

2580 1: Reporting Template Information Not Found.

2581 1.1: In the Sub Flow 3.2.1, 3.3.2, & 3.4.1, if the reporting template information cannot be
2582 found, Functional Element throws exception with error code as 'Reporting Template does not
2583 exist'.

2584 2: Duplicate Reporting Template Information.

2585 2.1: In the Sub Flow 3.1.2, If the same reporting template information has been defined,
2586 Functional Element throws exception with error code as 'Duplicate reporting template
2587 information'.

2588 **2.7.7.1.3 Special Requirements**

2589 None.

2590 **2.7.7.1.4 Pre-Conditions**

2591 None.

2592 **2.7.7.1.5 Post-Conditions**

2593 None.

2594

2595 **2.7.7.2 Generate Reports**

2596 This use case allows any user to generate reports, which includes Static Information Listing and
2597 Dynamic Information Queries.

2598 **2.7.7.2.1 Flow of Events**

2599 **2.7.7.2.1.1 Basic Flow**

2600 This use case starts when the user of the data source wishes to generate reports that include
2601 Static Information Listing and Dynamic Information Queries.

2602 1: Any user initiates a request type to the Functional Element stating whether to generate reports
2603 that includes Static Information Listing or Dynamic Information Queries.

2604 2: Based on the operation it specified, one of the following basic flows is executed:

- 2605 • If the operation is 'Manage Static Information Listing', then Manage Static Information
2606 Listing Basic Flow is executed.
- 2607 • If the operation is 'Manage Dynamic Information Queries', then Manage Dynamic
2608 Information Queries Basic Flow is executed.

2609 3: Whenever a report is generated using a particular reporting template, the respective report
2610 subscribers are notified via email using NOTIFICATION Functional Element and the use case
2611 ends.

2612

2613 **2.7.7.3 Manage Static Information Listing**

2614 **2.7.7.3.1 Description**

2615 This use case allows any users to create, view, update, and delete Static Information Listing.

2616 **2.7.7.3.2 Flow of Events**

2617 **2.7.7.3.2.1 Basic Flow**

2618 This use case starts when the users of the data source wishes to create, view, update, and delete
2619 Static Information Listing.

2620 1: Any user initiates a request type to the Functional Element stating whether to create, view,
2621 update, or delete Static Information Listing.

2622 2: The Functional Element checks whether the Static Information Listing exists.

2623 3: Based on the operation it specified, one of the following sub-flows is executed:

- 2624 • If the operation is '**Create Static Information Listing**', then sub-flow 3.1 is executed.
- 2625 • If the operation is '**View Static Information Listing**', then sub-flow 3.2 is executed.
- 2626 • If the operation is '**Update Static Information Listing**', then sub-flow 3.3 is executed.
- 2627 • If the operation is '**Delete Static Information Listing**', then sub-flow 3.4 is executed.

2628 3.1: Create Static Information Listing.

2629 3.1.1: Any user provides Static Information Listing to be created.

2630 3.1.2: The Functional Element checks for the duplicate Static Information Listing.

2631 3.1.3: The Functional Element creates the Static Information Listing, if it does not exist and the
2632 use case ends.

2633 3.2: View Static Information Listing.

2634 3.2.1: The Functional Element retrieves all the Static Information Listing.

2635 3.2.2: The Functional Element returns the Static Information Listing to any user and the use case
2636 ends.

2637 3.3: Update Static Information Listing.

2638 3.3.1: Any user provides Static Information Listing to be updated.

2639 3.3.2: The Functional Element checks for the availability of Static Information Listing.

2640 3.3.3: The Functional Element updates the Static Information Listing, if it exist and the use case
2641 ends.

2642 3.4: Delete Static Information Listing.

2643 3.4.1: Any user provides Static Information Listing to be removed.

2644 3.4.2: The Functional Element removes the Static Information Listing.

2645 4: The Functional Element responses the status of the operation whether it is successful or failure
2646 to any user and the use case ends.

2647 **2.7.7.3.2.2 Alternative Flows**

2648 1: Static Information Listing Not Found.

2649 1.1: In the Sub Flow 3.2.1, 3.3.2, & 3.4.1, if the Static Information Listing cannot be found,
2650 Functional Element throws exception with error code as 'Static Information Listing does not
2651 exist'.

2652 2: Duplicate Static Information Listing.

2653 2.1: In the Sub Flow 3.1.2, If the same Static Information Listing has been defined, Functional
2654 Element throws exception with error code as 'Duplicate Static Information Listing'.

2655 **2.7.7.3.3 Special Requirements**

2656 This use case requires the following three elements:

- 2657 • A data source
- 2658 • A static information query
- 2659 • A reporting template

2660 **2.7.7.3.4 Pre-Conditions**

2661 None.

2662 **2.7.7.3.5 Post-Conditions**

2663 None.

2664

2665 **2.7.7.4 Manage Dynamic Information Queries**

2666 **2.7.7.4.1 Description**

2667 This use case allows any users to create, view, update, and delete dynamic information queries.

2668 **2.7.7.4.2 Flow of Events**

2669 This use case starts when the users of the data source wishes to create, view, update, or delete
2670 dynamic information queries.

2671 1: Any user initiates a request type to the Functional Element stating whether to create, view,
2672 update, or delete Dynamic Information Queries.

2673 2: The Functional Element checks whether the Dynamic Information Query exists.

2674 3: Based on the operation it specified, one of the following sub-flows is executed:

- 2675 • If the operation is '**Create Dynamic Information Query**', then sub-flow 3.1 is executed.
- 2676 • If the operation is '**View Dynamic Information Query**', then sub-flow 3.2 is executed.
- 2677 • If the operation is '**Update Dynamic Information Query**', then sub-flow 3.3 is executed.
- 2678 • If the operation is '**Delete Dynamic Information Query**', then sub-flow 3.4 is executed.

2679 3.1: Create Dynamic Information Query.

2680 3.1.1: Any user provides Dynamic Information Query to be created.

2681 3.1.2: The Functional Element checks for the duplicate Dynamic Information Query.

2682 3.1.3: The Functional Element creates the Dynamic Information Query, if it does not exist
2683 and the use case ends.

2684 3.2: View Dynamic Information Query.

2685 3.2.1: The Functional Element retrieves all the Dynamic Information Queries.

2686 3.2.2: The Functional Element returns the Dynamic Information Query to any user and
2687 the use case ends.

2688 3.3: Update Dynamic Information Query.

2689 3.3.1: Any user provides Dynamic Information Query to be updated.

2690 3.3.2: The Functional Element checks for the availability of Dynamic Information Query.

2691 3.3.3: The Functional Element updates the Dynamic Information Query, if it exist and the
2692 use case ends.

2693 3.4: Delete Dynamic Information Query.

2694 3.4.1: Any user provides Dynamic Information Query to be removed.

2695 3.4.2: The Functional Element removes the Dynamic Information Query.

2696 4: The Functional Element responses the status of the operation whether it is successful or failure
2697 to any user and the use case ends.

2698 **2.7.7.4.2.1 Alternative Flows**

2699 1: Dynamic Information Query Not Found.

2700 1.1: In the Sub Flow 3.2.1, 3.3.2, & 3.4.1, if the Dynamic Information Query cannot be found,
2701 Functional Element throws exception with error code as 'Dynamic Information Query does not
2702 exist'.

2703 2: Duplicate Dynamic Information Query.

2704 2.1: In the Sub Flow 3.1.2, If the same Dynamic Information Query has been defined,
2705 Functional Element throws exception with error code as 'Duplicate Dynamic Information
2706 Query'.

2707 **2.7.7.4.3 Special Requirements**

2708 This use case requires the following three elements:

- 2709 • A data source
- 2710 • A dynamic information query
- 2711 • A reporting template

2712 **2.7.7.4.4 Pre-Conditions**

2713 None.

2714 **2.7.7.4.5 Post-Conditions**

2715 None.

2716

2717 **2.7.7.5 Manage Reports**

2718 **2.7.7.5.1 Description**

2719 This use case allows any users to view, update, and delete reports.

2720 **2.7.7.5.2 Flow of Events**

2721 **2.7.7.5.2.1 Basic Flow**

2722 This use case starts when the users of the data source wishes to view, update, or delete reports.

2723 1: Any user initiates a request type to the Functional Element stating whether to view, update, or
2724 delete reports.

2725 2: The Functional Element checks whether the report exists.

2726 3: Based on the operation it specified, one of the following sub-flows is executed:

- 2727 • If the operation is '**View Report**', then sub-flow 3.1 is executed.
2728 • If the operation is '**Update Report**', then sub-flow 3.2 is executed.
2729 • If the operation is '**Delete Report**', then sub-flow 3.3 is executed.

2730 3.1: View Report.

2731 3.1.1: The Functional Element retrieves all the reports.

2732 3.1.2: The Functional Element returns the report information to any user and the use
2733 case ends.

2734 3.2: Update Report.

2735 3.2.1: Any user provides report information to be updated.

2736 3.2.2: The Functional Element checks for the availability of report information.

2737 3.2.3: The Functional Element updates the report information, if it exist and the use case
2738 ends.

2739 3.3: Delete Report.

2740 3.3.1: Any user provides report information to be removed.

2741 3.3.2: The Functional Element removes the report information.

2742 4: The Functional Element responses the status of the operation whether it is successful or failure
2743 to any user and the use case ends.

2744 **2.7.7.5.2.2 Alternative Flows**

2745 1: Report Information Not Found.

2746 1.1: In the Sub Flow 3.1.1, 3.2.2, & 3.3.1, if the report information cannot be found, Functional
2747 Element throws exception with error code as 'Report does not exist'.

2748 **2.7.7.5.3 Special Requirements**

2749 None.

2750 **2.7.7.5.4 Pre-Conditions**

2751 None.

2752 **2.7.7.5.5 Post-Conditions**

2753 None.

2754

2755 **2.7.7.6 Configure Data Source**

2756 **2.7.7.6.1 Description**

2757 This use case allows any users to create, view, update, and delete data source.

2758 **2.7.7.6.2 Flow of Events**

2759 **2.7.7.6.2.1 Basic Flow**

2760 This use case starts when the users of the data source wishes to create, view, update, or delete
2761 data source.

2762 1: Any user initiates a request type to the Functional Element stating whether to create, view,
2763 update, or delete source.

2764 2: The Functional Element checks whether the data source exists.

2765 3: Based on the operation it specified, one of the following sub-flows is executed:

- 2766 • If the operation is '**Create Data Source**', then sub-flow 3.1 is executed.
- 2767 • If the operation is '**View Data Source**', then sub-flow 3.2 is executed.
- 2768 • If the operation is '**Update Data Source**', then sub-flow 3.3 is executed.
- 2769 • If the operation is '**Delete Data Source**', then sub-flow 3.4 is executed.

2770 3.1: Create Data Source.

2771 3.1.1: Any user provides data source information to be created.

2772 3.1.2: The Functional Element checks for the duplicate data source information.

2773 3.1.3: The Functional Element creates the data source information, if it does not exist and the use
2774 case ends.

2775 3.2: View Data Source.

2776 3.2.1: The Functional Element retrieves all the data sources.

2777 3.2.2: The Functional Element returns the data source information to any user and the use case
2778 ends.

2779 3.3: Update Data Source.

2780 3.3.1: Any user provides data source information to be updated.
2781 3.3.2: The Functional Element checks for the availability of data source information.
2782 3.3.3: The Functional Element updates the data source information, if it exist and the use case
2783 ends.
2784 3.4: Delete Data Source.
2785 3.4.1: Any user provides data source information to be removed.
2786 3.4.2: The Functional Element removes the data source information.
2787 4: The Functional Element responses the status of the operation whether it is successful or failure
2788 to any user and the use case ends.

2789 **2.7.7.6.2 Alternative Flows**

2790 1: Data Source Information Not Found.
2791 1.1: In the Sub Flow 3.2.1, 3.3.2, & 3.4.1, if the data source information cannot be found,
2792 Functional Element throws exception with error code as 'Data source does not exist'.
2793 2: Duplicate Data Source Information.
2794 2.1: In the Sub Flow 3.1.2, If the same data source information has been defined, Functional
2795 Element throws exception with error code as 'Duplicate data source information'.

2796 **2.7.7.6.3 Special Requirements**

2797 None.

2798 **2.7.7.6.4 Pre-Conditions**

2799 None.

2800 **2.7.7.6.5 Post-Conditions**

2801 None.

2802

2803 **2.7.7.7 Design Report Templates**

2804 **2.7.7.7.1 Description**

2805 This use case allows any users to design reporting templates.

2806 **2.7.7.7.2 Flow of Events**

2807 **2.7.7.7.2.1 Basic Flow**

2808 The use case begins when the user wants to design reporting templates.

- 2809 1: Any user provides reporting template information to be designed.
2810 2: The Functional Element checks for the duplicate reporting template information designed.
2811 3: The Functional Element designs and saves the reporting template information, if it does not
2812 exist and the use case ends.

2813 **2.7.7.7.2 Alternative Flows**

- 2814 1: Duplicate Reporting Template Design Information.
2815 1.1: In the Basic Flow 2, if the same reporting template information has been designed,
2816 Functional Element throws exception with error code as 'Duplicate reporting template design
2817 information'.

2818 **2.7.7.7.3 Special Requirements**

2819 None.

2820 **2.7.7.7.4 Pre-Conditions**

2821 None.

2822 **2.7.7.7.5 Post-Conditions**

2823 None.

2824

2825 **2.7.7.8 Transform Reports**

2826 **2.7.7.8.1 Description**

2827 This use case allows the user to transform a report information file from one format to another
2828 format.

2829 **2.7.7.8.2 Flow of Events**

2830 **2.7.7.8.2.1 Basic Flow**

2831 This use case starts when a user wants to transform a report information file from one format to
2832 another format.

2833 1: The user set the file name to be transformed and the destination format.

2834 2: This use case call the TRANSFORMER functional elements' transform flow.

2835 3: Return the results from the transformer functional elements' transform flow and the use case
2836 ends.

2837 **2.7.7.8.2.2 Alternative Flows**

2838 1: If in Basic Flow 2 there is no method to do the transformation, the system return error message
2839 to the user and this use case ends.

2840 **2.7.7.8.3 Special Requirements**

2841 None.

2842 **2.7.7.8.4 Pre-Conditions**

2843 None.

2844 **2.7.7.8.5 Post-Conditions**

2845 None.

2846

2847 **2.7.7.9 Subscribe/Un-subscribe Reports**

2848 **2.7.7.9.1 Description**

2849 This use case performs the subscription or un-subscription on desired reports for any user.

2850 **2.7.7.9.2 Flow of Events**

2851 **2.7.7.9.2.1 Basic Flow**

2852 The use case begins when the user wants to subscribe or un-subscribe those desired reports.

2853 1: The user sends a request.

2854 2: Based on the operation it specifies, one of the following sub-flows is executed:

- 2855 • If the operation is '**Subscribe to Report**', then sub-flow 2.1 is executed.
- 2856 • If the operation is '**Un-Subscribe to Report**', then sub-flow 2.2 is executed.

2857 2.1: Subscribe To Report.

2858 2.1.1: The Functional Element gets user id, together with those desired report name.

2859 2.1.2: The Functional Element checks whether the report exists.

2860 2.1.3: The Functional Element adds the subscription of the user to the report.

2861 2.2: Un-Subscribe To Report.

2862 2.2.1: The Functional Element gets user id, together with those desired report name.

2863 2.2.2: The Functional Element checks whether the report exists.

2864 2.2.3: The Functional Element removes the subscription of the user to the report.

2865 3: The Functional Element returns the results of the operation to the user and the use case ends.

2866 **2.7.7.9.2.2 Alternative Flows**

2867 1: Report Not Found.

2868 1.1: If in the basic flow 2.1.2 and 2.2.2, the report specified does not exist, Functional
2869 Element will return an error message to the user and the use case ends.

2870 2: User Not Found.

2871 2.1: If in the basic flow 2.1.2 and 2.2.2, the user related does not exist, Functional Element
2872 will return an error message to the user and the use case ends.

2873 **2.7.7.9.3** **Special Requirements**

2874 None.

2875 **2.7.7.9.4** **Pre-Conditions**

2876 None.

2877 **2.7.7.9.5** **Post-Conditions**

2878 None.

2879 **2.8 Key Management Functional Element (new)**

2880 **2.8.1 Motivation**

2881 The Key Management Functional Element is expected to be related Web Services security. To
2882 enable Web Services security, cryptographic keys are used for digital signatures and encryption.
2883 XKMS defines a Web services interface to a public key infrastructure. With development of
2884 XKMS standard, more and more PKI providers adopt XKMS to remove its complexity without
2885 sacrificing its benefits. Application developers will only ever need to worry about implementing
2886 XKMS clients for key management.

2887

2888 This Functional Element fulfills the following requirements from the Functional Elements
2889 Requirements Document 02:

- 2890
 - Primary Requirements
 - SECURITY-010.
 - Secondary Requirements
 - None.

2894

2895 **2.8.2 Terms Used**

Terms	Description
PKI	PKI is a system of digital certificates, Certificate Authorities, and other registration authorities that verify and authenticate the validity of each party involved in an Internet transaction.
XML Key Management Specification (XKMS)	This specification addresses protocols for distributing and registering public keys, suitable for use in conjunction with the standards for XML Signature, XML Encryption and WS-Security.
the XML Key Information Service Specification (X-KISS)	The X-KISS is a specification that defines a protocol for a XKMS-compliant service that resolves public key information. It allows a client of such a service to delegate part or all of the tasks required to process <ds:KeyInfo>.
X-KRSS	XML Key Registration Service Specification defines a protocol for a web service that accepts registration of public key information.
Proof of Possession (POP)	Performing an action with a private key to demonstrate possession of it. An example is to create a signature using a registered private signing key to prove possession of it.

2896

2897 **2.8.3 Key Features**

2898 Implementations of the Key Management Functional Element are expected to provide the
2899 following key features:

- 2900 1. The Functional Element MUST provide the capability to register a key or a key pair with an
2901 XKMS-compliant service.

- 2902 2. The Functional Element MUST provide the capability to revoke a registered key or key pair
2903 with an XKMS-compliant service.
- 2904 3. The Functional Element MUST provide the capability to recover a registered key or key pair
2905 with an XKMS-compliant service.
- 2906 4. The Functional Element MUST provide the capability to retrieve a public key registered with
2907 an XKMS-compliant service. The public can in turn be used to encrypt a document or verify
2908 a signature.
- 2909 5. The Functional Element MUST provide the capability to ensure that a public key registered
2910 with an XKMS-compliant service is valid and has not expired or been revoked.

2911

2912 In addition, the following key features could be provided to enhance the Functional Element
2913 further:

- 2914 1. The Functional Element MAY provide the capability to generate key pairs.

2915

2916 2.8.4 Interdependencies

Interaction Dependencies	
SecureSOAP Management	The SecureSOAP Management Functional Element may make use key management facilities provided by this functional element to do security related operations.
Identity Management	The Identity Management Functional Element may make use of key management facility to obtain KeyInfo.

2917

2918 2.8.5 Related Technologies and Standards

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

XML-Encryption Syntax and Processing, W3C Recommendation 10 th Dec 2002	This specification addresses data privacy by defining a process for encrypting data and representing the result in XML document. In this Functional Element, the encryption and decryption of SOAP messages are done based on this specification.
XML Key Management Specification (XKMS)	This specification addresses protocols for distributing and registering public keys, suitable for use in conjunction with the standards for XML Signature, XML Encryption and WS-Security. It comprises two parts – the XML Key Information Service Specification (X-KISS) and the XML Key Registration Service Specification (X-KRSS).

2919 **2.8.6 Model**

2920

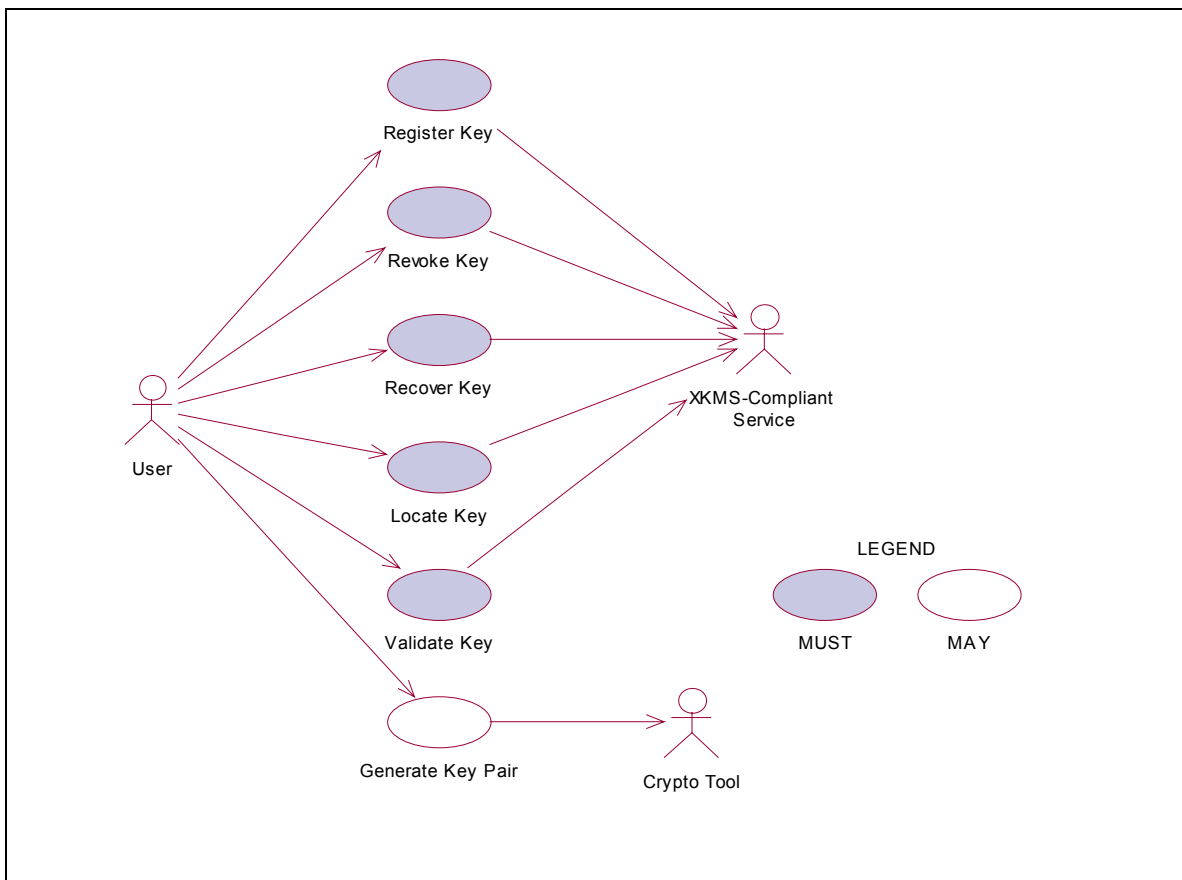


Figure 13: Model of the Key Management Functional Element.

2921

2922 **2.8.7 Usage Scenarios**

2923 **2.8.7.1 Register Key**

2924 **2.8.7.1.1 Description**

2925 This use case allows any user to register a key or key pair with a XKMS-compliant service.

2926 **2.8.7.1.2 Flow of Events**

2927 **2.8.7.1.2.1 Basic Flow**

2928 This use case starts when any user wants to register a key or key pair with a XKMS-compliant
2929 service. The register request is used to assert a binding of information to a public key pair.
2930 Generation of the public key pair MAY be performed by either the client or the XKMS-compliant
2931 service.

2932 1: The user sends request to register a key or key pair by providing necessary registering
2933 information, which include key information, a prototype of the requested assertion, optional
2934 additional information to authenticate the user. If the public key pair to be registered is generated
2935 by the user, the user may provide Proof of Possession of the private key.

2936 2: On receipt of a registering request from the user, the functional element transforms the request
2937 to X-KRSS request format and sends to targeted XKMS-compliant service.

2938 3: The XKMS-compliant service verifies the authentication and Proof of Possession information
2939 provided if any. If the service accepts the request, an assertion is registered. The service returns
2940 part or all of the registered assertion in format of X-KRSS to the functional element.

2941 4: The Functional Element passes the response from the service to the user and the use case
2942 ends.

2943 **2.8.7.1.2.2 Alternative Flows**

2944 1: Information Not Enough.

2945 1.1: If in the basic flow 2, Functional Element detects the information provided by the user is
2946 not enough to form a X-KRSS request, Functional Element returns general error message
2947 and ends the use case.

2948 2: POP Needed.

2949 2.1: If in the basic flow 2, Functional Element checks that key pair is generated but the POP
2950 is not provided by the user in the request message, the Functional Element returns an error
2951 and ends the use case.

2952 **2.8.7.1.3 Special Requirements**

2953 **2.8.7.1.4 Pre-Conditions**

2954 None.

2955 **2.8.7.1.5 Post-Conditions**

2956 None.

2957

2958 **2.8.7.2 Revoke Key**

2959 **2.8.7.2.1 Description**

2960 The use case allows any user to revoke previously issued assertions.

2961 **2.8.7.2.2 Flow of Events**

2962 **2.8.7.2.2.1 Basic Flow**

2963 This use case starts when any user wants to revoke previous issued assertions.

2964 1: The user sends request to revoke a key or key pair by providing information, which include key
2965 information, a prototype of the requested assertion, optional additional information to authenticate
2966 the user. If the public key pair to be registered is generated by the user, the user may provide
2967 Proof of Possession of the private key.

2968 2: On receipt of a revoking request from the user, the Functional Element transforms the request
2969 to X-KRSS request format and sends to targeted XKMS-compliant service.

2970 3: The XKMS-compliant service verifies the authentication and Proof of Possession information
2971 provided if any. If the service accepts the request, an assertion is revoked. The service returns
2972 response in X-KRSS to indicate that the assertion is revoked.

2973 4: The Functional Element passes the response from the service to the user and the use case
2974 ends.

2975 **2.8.7.2.3 Alternative Flows**

2976 1: Information Not Enough.

2977 1.1: If in the basic flow 2, Functional Element detects the information provided by the user is
2978 not enough to form an X-KRSS request, Functional Element returns general error message
2979 and ends the use case.

2980 2: POP Needed.

2981 2.1: If in the basic flow 2, Functional Element checks that key pair is generated but the POP
2982 is not provided by the user in the request message, the Functional Element returns an error
2983 and ends the use case.

2984 **2.8.7.2.4 Special Requirements**

2985 None.

2986 **2.8.7.2.5 Pre-Conditions**

2987 None.

2988 **2.8.7.2.6 Post-Conditions**

2989 If the use case was successful, the assertion issued previously would be revoked.

2990

2991

2992 **2.8.7.3 Recover Key**

2993 This use case allows any user to recover previously issued assertions.

2994 **2.8.7.3.1 Flow of Events**

2995 **2.8.7.3.1.1 Basic Flow**

2996 This use case starts when any user wants to recover previous issued assertions.

2997 1: The user sends request to recover a key or key pair by providing information, which include
2998 key information, a prototype of the requested assertion, optional additional information to
2999 authenticate the user. If the public key pair to be registered is generated by the user, the user
3000 may provide Proof of Possession of the private key.

3001 2: On receipt of a recover request from the user, the Functional Element transforms the request
3002 to X-KRSS request format and sends to targeted XKMS-compliant service.

3003 3: The XKMS-compliant service verifies the authentication and Proof of Possession information
3004 provided if any. If the service accepts the request, an assertion is recovered. The service returns
3005 response in X-KRSS to indicate that the assertion is recovered.

3006 4: The Functional Element passes the response from the service to the user and the use case
3007 ends.

3008 **2.8.7.3.1.2 Alternative Flows**

3009 1: Information Not Enough.

3010 1.1: If in the basic flow 2, Functional Element detects the information provided by the user is
3011 not enough to form an X-KRSS request, Functional Element returns general error message
3012 and ends the use case.

3013 2: POP Needed.

3014 2.1: If in the basic flow 2, Functional Element checks that key pair is generated but the POP
3015 is not provided by the user in the request message, the Functional Element returns an error
3016 and ends the use case.

3017 **2.8.7.3.2 Special Requirements**

3018 None.

3019 **2.8.7.3.3 Pre-Conditions**

3020 None.

3021 **2.8.7.3.4 Post-Conditions**

3022 If the use case successes, the registered assertion is recovered in the XKMS-compliant service.

3023

3024 **2.8.7.4 Locate Key**

3025 **2.8.7.4.1 Description**

3026 This use case allows users to retrieve a public key registered with an XKMS-compliant service.
3027 The public key can be in turn be used to encrypt a document or verify a signature.

3028 **2.8.7.4.1.1 Basic Flow**

3029 This use case starts when any user wants to retrieve a public key registered with an XKMS-
3030 compliant service.

3031 1: The user sends request to retrieve a public key registered with an XKMS-compliant service by
3032 providing related information.

3033 2: On receipt of a recover request from the user, the Functional Element transforms the request
3034 to X-KISS request format and sends to targeted XKMS-compliant service.

3035 3: The XKMS-compliant service may obtain an X509V3 certificate. The certificate is parsed to
3036 obtain the public key value that is return to the Functional Element in the format of X-KISS.

3037 4: The Functional Element checks the response message is issued by the target XKMS-compliant
3038 service; ensures that the response message has not been modified; and confirms that the
3039 response message corresponds to the request that made by the user.

3040 5: The Functional Element passes the response from the service to the user and the use case
3041 ends.

3042 **2.8.7.4.1.2 Alternative Flows**

3043 1: Information Not Enough.

3044 1.1: If in the basic flow 2, Functional Element detects the information provided by the user is
3045 not enough to form an X-KISS request, Functional Element returns general error message
3046 and ends the use case.

3047 2: Fault Response.

3048 2.1: If in basic flow 4, Functional Element detects the response message has problem in
3049 authenticity, integrity and does not correspond to the request, Functional Element returns
3050 general error message and ends the use case.

3051 **2.8.7.4.2 Special Requirements**

3052 None.

3053 **2.8.7.4.3 Pre-Conditions**

3054 None.

3055 **2.8.7.4.4 Post-Conditions**

3056 None.

3057

3058 **2.8.7.5 Validate Key**

3059 This use case enables the user to obtain an assertion specifying the status of the binding
3060 between the public key and other data, for example a name or a set of extended attributes.

3061 **2.8.7.5.1 Flow of Events**

3062 **2.8.7.5.1.1 Basic Flow**

3063 This use case starts when the user wants to obtain the status of the binding of a public key with
3064 an assertion.

3065 1: The user sends request to validate a key or key pair by providing necessary validating
3066 information defined in X-KISS, which include key information, a prototype of the requested
3067 assertion, optional additional information to authenticate the user. If the public key pair to be
3068 registered is generated by the user, the user may provide Proof of Possession of the private key.

3069 2: On receipt of a registering request from the user, the Functional Element transforms the
3070 request to XKRSS request format and sends to targeted XKMS-compliant service.

3071 3: The XKMS-compliant service verifies the authentication and Proof of Possession information
3072 provided if any. If the service accepts the request, an assertion is registered. The service returns
3073 part or all of the registered assertion in format of XKRSS to the functional element.

3074 4: The Functional Element checks the response message is issued by the target XKMS-compliant
3075 service; ensures that the response message has not been modified; and confirms that the
3076 response message corresponds to the request that made by the user.

3077 5: The Functional Element passes the response from the service to the user and the use case
3078 ends.

3079 **2.8.7.5.1.2 Alternative Flows**

3080 1: Information Not Enough.

3081 1.1: If in the basic flow 2, Functional Element detects the information provided by the user is
3082 not enough to form an X-KISS request, Functional Element returns general error message
3083 and ends the use case.

3084 2: Fault Response.

3085 2.1: If in basic flow 4, Functional Element detects the response message has problem in
3086 authenticity, integrity and does not correspond to the request, Functional Element returns
3087 general error message and ends the use case.

3088 **2.8.7.5.2 Special Requirements**

3089 None.

3090 **2.8.7.5.3 Pre-Conditions**

3091 None.

3092 **2.8.7.5.4 Post-Conditions**

3093 None.

3094

3095 **2.8.7.6 Generate Key Pair**

3096 This use case enables the user to generate key pair using the desired cryptographic tool.

3097 **2.8.7.6.1 Flow of Events**

3098 **2.8.7.6.1.1 Basic Flow**

3099 This use case starts when the user wants to obtain generate key pair using the desired
3100 cryptographic tool.

3101 1: The user sends request to generate key pair by specifying related information.

3102 2: On receipt of request from the user, the functional element validates the provided information
3103 and dispatch the request to Crypto Tool to generate key pair.

3104 3: The Crypto Tool generates key pair and returns them to the Functional Element according to
3105 the request.

3106 4: The Functional Element checks and dispatches the message to the user and the use case
3107 ends.

3108 **2.8.7.6.1.2 Alternative Flows**

3109 1: Invalid Input Parameter.

3110 1.1: If in the basic flow 2, Functional Element detects the information provided by the user is
3111 not valid to generate key pair, Functional Element returns general error message and ends
3112 the use case.

3113 **2.8.7.6.2 Special Requirements**

3114 None.

3115 **2.8.7.6.3 Pre-Conditions**

3116 None.

3117 **2.8.7.6.4 Post-Conditions**

3118 If the use case successes, a key pair is generated and stored in the key store specified by the
3119 user.

3120

3121

3122 **2.9 Log Utility Functional Element**

3123 **2.9.1 Motivation**

3124 In a Web Service-enabled implementation, the Log Utility Functional Element can help to
3125 organise the diagnostic output that may be generated by the implementation. In order to achieve
3126 that, the following capabilities should be provided. They include:

- 3127 • Logging information into different data sources,
- 3128 • Allowing user defined log format to be used,
- 3129 • Capability for storing log information, and
- 3130 • Providing the capability to analyse the information log.

3131

3132 This Functional Element fulfills the following requirements from the Functional Elements
3133 Requirements Document 02:

- 3134 • Primary Requirements
 - 3135 ○ MANAGEMENT-007,
 - 3136 ○ MANAGEMENT-110,
 - 3137 ○ MANAGEMENT-112 to MANAGEMENT-114, and
 - 3138 ○ PROCESS-009.
- 3139 • Secondary Requirements
 - 3140 ○ MANAGEMENT-006,
 - 3141 ○ MANAGEMENT-095,
 - 3142 ○ MANAGEMENT-111,
 - 3143 ○ PROCESS-008,
 - 3144 ○ PROCESS-115, and
 - 3145 ○ PROCESS-118.

3146

3147 **2.9.2 Terms Used**

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

3148

3149 **2.9.3 Key Features**

3150 Implementations of the Log Utility Functional Element are expected to provide the following key
3151 features:

- 3152 1. The Functional Element MUST provide the capability to define a Log Category and manage
3153 it. This includes:

- 3154 1.1. The capability to define the format of the log information,
3155 1.2. The capability to choose the data source to logged to, and
3156 1.3. The capability to define the name of the log category.
- 3157 2. The Functional Element MUST provide the capability to manage logging of events/records.
3158 This includes:
- 3159 2.1. The capability to insert a new record into the log,
3160 *Examples of a log record could include events, transactions status, usages status or*
3161 *users' activities.*
- 3162 2.2. The capability to search and return result sets of search on log records, and
3163 2.3. The capability to archive or delete obsolete log records.

3164

3165 In addition, the following key features could be provided to enhance the Functional Element
3166 further:

- 3167 1. The Functional Element MAY also provide the capability to perform conditional search or
3168 viewing of log records.
- 3169 2. The Functional Element MAY provide the capability to perform basic statistical analysis on
3170 log records. Basic statistical analysis capabilities include:
- 3171 2.1. Minimum and maximum value calculations on numerical values,
3172 2.2. Mean values calculations on numerical values, and
3173 2.3. Standard deviation calculations on numerical values.

3174

3175 **2.9.4 Interdependencies**

3176 None

3177 **2.9.5 Related Technologies and Standards**

3178 None

3181 **2.9.6 Model**

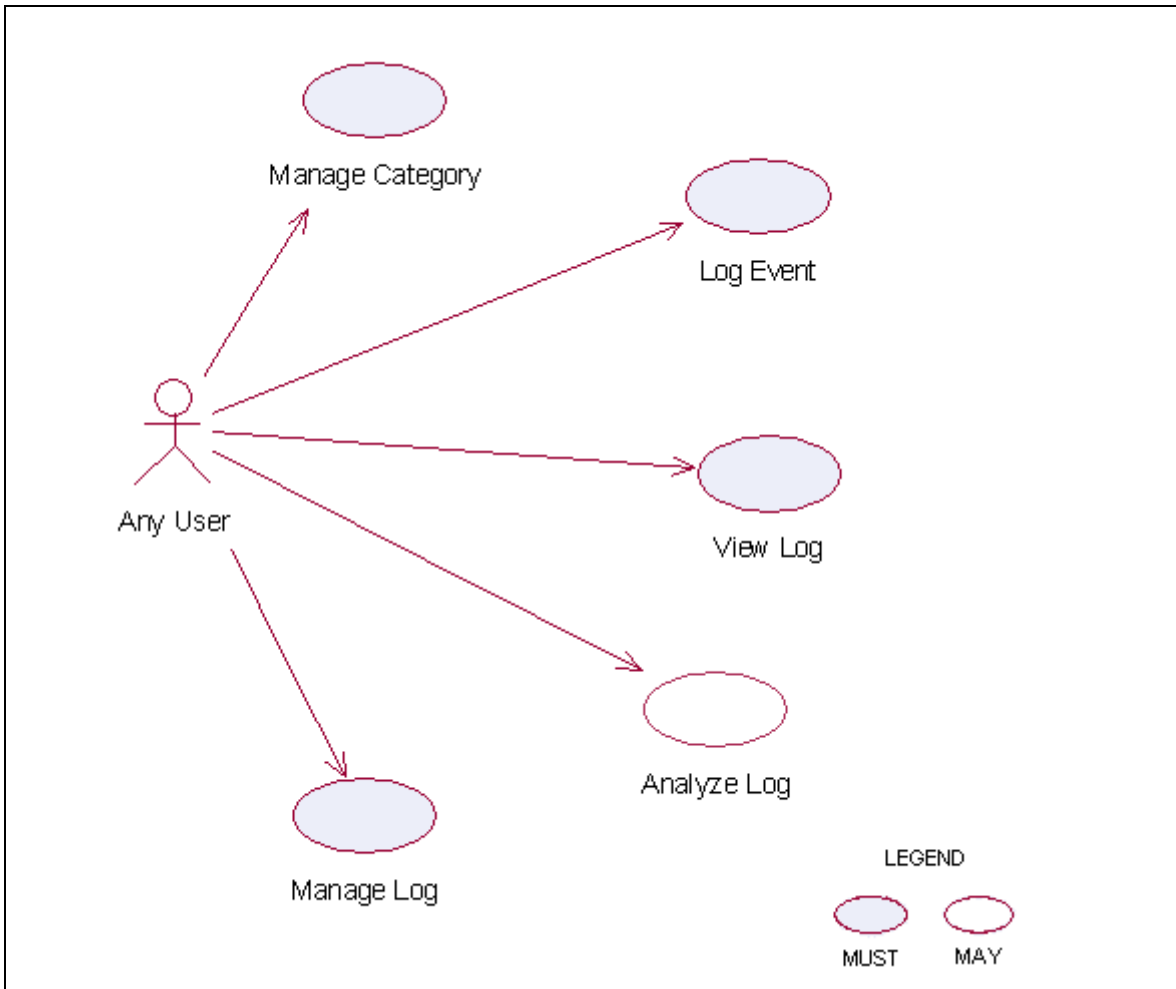


Figure 14: Model Of the Log Utility Functional Element [12]

3182

3183 **2.9.7 Usage Scenarios**

3184 **2.9.7.1 Manage Category**

3185 **2.9.7.1.1 Description**

3186 This use case allows any user to manage log category. Log category defines the data fields that
3187 the user wants to log.

3188 **2.9.7.1.2 Flow of Events**

3189 **2.9.7.1.2.1 Basic Flow**

3190 This use case starts when users wants to manage the log category.

3191 1: The users send the request to the Functional Element. The request contains the operations
3192 the users want to perform.

- 3193 2: The Functional Element receives the request. Based on the operation specified, one of the
3194 following sub-flows is executed.
- 3195 If the operation is 'Create Log Category', then sub-flow 2.1 is executed.
- 3196 If the operation is 'Retrieve Log Category Information', then sub-flow 2.2 is executed.
- 3197 If the operation is 'Delete Log Category', then sub-flow 2.3 is executed.
- 3198 2.1: Create Log Category.
- 3199 2.1.1: The Functional Element gets the following data from the users.
- 3200 o Category name
- 3201 o The definition of category
- 3202 o The data source where the log is located
- 3203 2.1.2: The Functional Element checks the uniqueness of the category name.
- 3204 2.1.3: The Functional Element connects to the data source according to the specified
3205 data source.
- 3206 2.1.4: The Functional Element creates the empty log in the data source.
- 3207 2.1.5: The Functional Element writes the category name and its definition in its own
3208 category definition record and the use case end.
- 3209 2.2: Retrieve Log Category Information.
- 3210 2.2.1: The Functional Element gets the category name.
- 3211 2.2.2: The Functional Element checks the existence of this category.
- 3212 2.2.3: The Functional Element retrieves the definition of this category.
- 3213 2.2.4: The Functional Element returns the definition of this category to the user and the
3214 use case ends.
- 3215 2.3: Delete Log Category.
- 3216 2.3.1: The Functional Element gets the category name.
- 3217 2.3.2: The Functional Element checks the existence of this category.
- 3218 2.3.3: The Functional Element deletes its own records of category definition and the use
3219 case ends.

3220 **2.9.7.1.2.2 Alternative Flows**

- 3221 1: Category Already Exists.
- 3222 1.1: In sub-flow 2.1.2, if the category name is already used by others, the Functional Element
3223 returns an error message and the use case ends.
- 3224 2: Data Source Not Available.
- 3225 2.1: In sub-flow 2.1.3, if the data source is not available, the Functional Element returns an
3226 error message and the use case ends.
- 3227 3: Create Log Error.

- 3228 3.1: In sub-flow 2.1.4, if the log cannot be created on the specified data source, the
3229 Functional Element returns an error message and the use case ends.
- 3230 4: Category Does Not Exist.
- 3231 4.1: In sub-flow 2.2.1 and 2.3.1, the category cannot be found in Functional Element category
3232 definition, the Functional Element returns an error message and the use case ends.
- 3233 5: Delete Category Error.
- 3234 5.1: In sub-flow 2.3.3, the log category cannot be deleted, the Functional Element returns an
3235 error message and the use case ends.
- 3236 **2.9.7.1.3 Special Requirements**
- 3237 None
- 3238 **2.9.7.1.4 Pre-Conditions**
- 3239 None.
- 3240 **2.9.7.1.5 Post-Conditions**
- 3241 If the use case was successful, the category definition is saved to the Functional Element and an
3242 empty log is created in the specified data source. Otherwise, the Functional Element's state is
3243 unchanged.
- 3244 **2.9.7.2 Log Event**
- 3245 **2.9.7.2.1 Description**
- 3246 The use case allows any user to log any event.
- 3247 **2.9.7.2.2 Flow of Events**
- 3248 **2.9.7.2.2.1 Basic Flow**
- 3249 This use case starts when users want to write to a log.
- 3250 1: The users provide the event data, category name he/she wants to log to the Functional
3251 Element.
- 3252 2: The Functional Element gets the definition of the category.
- 3253 3: The Functional Element connects the log data source.
- 3254 4: The Functional Element writes the log record into the end of the log file and the use case ends.
- 3255 **2.9.7.2.2.2 Alternative Flows**
- 3256 1: Category Does Not Exist.
- 3257 1.1: If in basic flow 2, the category that the users want to write does not exist, the Functional
3258 Element returns an error message and the use case ends.
- 3259 2: Data Source Not Available.

3260 2.1: If in basic flow 3, the data source is not available, the Functional Element returns an error
3261 message and the use case ends.

3262 3: Data Not Match.

3263 3.1: If in basic flow 4, the data provided by the users for logging does not match with the
3264 category definition in the Functional Element, the Functional Element returns an error
3265 message and the use case ends.

3266 **2.9.7.2.3 Special Requirements**

3267 None.

3268 **2.9.7.2.4 Pre-Conditions**

3269 None.

3270 **2.9.7.2.5 Post-Conditions**

3271 If the use case was successful, the log record is saved to the Functional Element. Otherwise, the
3272 Functional Element's state is unchanged.

3273 **2.9.7.3 View Log**

3274 **2.9.7.3.1 Description**

3275 The use case allows users to retrieve the log content.

3276 **2.9.7.3.2 Flow of Events**

3277 **2.9.7.3.2.1 Basic Flow**

3278 This use case starts when users want to view a log.

3279 1: The users specify the category name and the search criteria, such as searching by event type
3280 or searching by time period (starting time and end time).

3281 2: The Functional Element connects to the data storage where the log records are stored.

3282 3: The Functional Element retrieves the log content and returns to the service users and the use
3283 case ends.

3284 **2.9.7.3.2.2 Alternative Flows**

3285 1: Search Criteria Not Valid.

3286 1.1: If in basic flow 1 and 3, the search criteria specified by the users is invalid for Search
3287 Service, the Functional Element returns an error message and the use case ends.

3288 **2.9.7.3.3 Special Requirements**

3289 None.

3290 **2.9.7.3.4 Pre-Conditions**

3291 None.

3292 **2.9.7.3.5 Post-Conditions**

3293 None.

3294 **2.9.7.4 Analyze Log Data**

3295 **2.9.7.4.1 Description**

3296 The use case allows users to analyze the log data, i.e., to get statistics of certain event. The
3297 service users may get statistical results on the log data, such as the cumulative events and mean
3298 of two numerical values.

3299 **2.9.7.4.2 Flow of Events**

3300 **2.9.7.4.2.1 Basic Flow**

3301 This use case starts when users want to analyze the log data.

3302 1: The users specify the items to analyze, i.e. field name and category name.

3303 2: The users specify the analysis method, option among max, min and mean.

3304 3: The Functional Element retrieves the definition of the category and validates the parameters
3305 provided by the users.

3306 4: The Functional Element connects to the data source and retrieves the log data.

3307 5: The Functional Element analyses the log data and does statistics on the data with respect to
3308 what is specified in Step 1 and 2.

3309 6: The Functional Element returns the analyzed result and the use case ends.

3310 **2.9.7.4.2.2 Alternative Flows**

3311 1: Invalid Item Specified.

3312 1.1: If in basic flow 1, the analyze items specified by the users are invalid, i.e. invalid field and
3313 invalid data source, the Functional Element returns an error message and the use case ends.

3314 2: Category Does Not Exist.

3315 2.1: If in basic flow 3, the category that the users want to write to does not exist, the
3316 Functional Element returns an error message and the use case ends.

3317 3: Data Source Not Available.

3318 3.1: If in basic flow 4, the data source is not available, the Functional Element returns an error
3319 message and the use case ends.

3320 **2.9.7.4.3 Special Requirements**

3321 **2.9.7.4.3.1 Supportability**

3322 Only basic statistic methods of numerical value are supported.

3323 **2.9.7.4.4 Pre-Conditions**

3324 None.

3325 **2.9.7.4.5 Post-Conditions**

3326 None.

3327 **2.9.7.5 Manage Log**

3328 **2.9.7.5.1 Description**

3329 The use case allows users to drop log and backup log.

3330 **2.9.7.5.2 Flow of Events**

3331 **2.9.7.5.2.1 Basic Flow**

3332 The use case starts when the users want to drop and backup a log of a specific data source.

3333 1: The users specify the function name to the Functional Element.

3334 2: Based on the operation specified, one of the following sub-flows is executed.

3335 If the operation is '**Delete Log**', then sub-flow 2.1 is executed.

3336 If the operation is '**Backup Log**', then sub-flow 2.2 is executed.

3337 2.1: Delete Log

3338 2.1.1: The Functional Element gets category name from the users.

3339 2.1.2: The Functional Element retrieves the definition of the category.

3340 2.1.3: The Functional Element connects to the corresponding data source.

3341 2.1.4: The Functional Element deletes the log from the data source.

3342 2.2: Backup Log

3343 2.2.1: The Functional Element gets the category name and the destination file name from
3344 the users.

3345 2.2.2: The Functional Element retrieves the definition of the category.

3346 2.2.3: The Functional Element connects to the corresponding data source.

3347 2.2.4: The Functional Element read the original log and writes it to the destination file.

3348 **2.9.7.5.2.2 Alternative Flows**

3349 1: Category Does Not Exist.

3350 1.1: If in basic flow 2.1.2 and 2.2.2 the category that the users want to write does not exist,
3351 the Functional Element returns an error message and the use case ends.

3352 2: Data Source Not Available.

3353 2.1: If in basic flow 2.1.4 and 2.2.4, the data source is not available, the Functional Element
3354 returns an error message and the use case ends.

3355 **2.9.7.5.3 Special Requirements**

3356 None.

3357 **2.9.7.5.4 Pre-Conditions**

3358 None.

3359 **2.9.7.5.5 Post-Conditions**

3360 None.

3361 **2.10 Notification Functional Element**

3362 **2.10.1 Motivation**

3363 In a Web Service-enabled implementation, timely information is crucial for the management of
3364 resources that it encompasses. Other uses of this Functional Element include broadcasting of
3365 information to other services and this could span across both the wired and wireless medium. In
3366 order to fulfill these needs, this Functional Element will cover the following aspects which include:

- 3367 • Providing the capability to configure and link with the various gateways so as to enable
3368 messages dissemination, and
- 3369 • Providing the capability to send instantaneous or scheduled messages to the intended
3370 audiences.

3371

3372 This Functional Element fulfills the following requirements from the Functional Elements
3373 Requirements Document 02

- 3374 • Primary Requirements
 - 3375 ○ DELIVERY-003, and
 - 3376 ○ PROCESS-118.
- 3377 • Secondary Requirements
 - 3378 ○ MANAGEMENT-205,
 - 3379 ○ PROCESS-005,
 - 3380 ○ PROCESS-102,
 - 3381 ○ PROCESS-107, and
 - 3382 ○ PROCESS-110.

3383

3384 **2.10.2 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.

3385

3386 2.10.3 Key Features

3387 Implementations of the Notification Functional Element are expected to provide the following key
3388 features:

- 3389 1. The Functional Element MUST support notifications using both the SMS and SMTP
3390 protocols.
- 3391 7. The Functional Element MUST provide the capability to configure supported SMS gateway(s)
3392 and the SMTP servers where applicable.
- 3393 *Example: The capability to configure the username and password for SMTP server's*
3394 *authentication.*
- 3395 8. The Functional Element MUST provide the capability to send notifications to single and
3396 multiple recipients.
- 3397 9. The Functional Element MUST provide the capability to structure a notification based on the
3398 selected protocol(s).

3399

3400 In addition, the following key features could be provided to enhance the Functional Element
3401 further:

- 3402 1. The Functional Element MAY provide the capability to send notifications either instantly or
3403 based on a pre-defined schedule.
- 3404 2. If Key Feature (1) is provided, the Functional Element MAY also provide the capability to
3405 send scheduled messages in the following manner:
- 3406 2.1. Hourly,
3407 2.2. Daily,
3408 2.3. Weekly, and
3409 2.4. Monthly (based on a particular date or particular day of the week).

3410

3411 2.10.4 Interdependencies

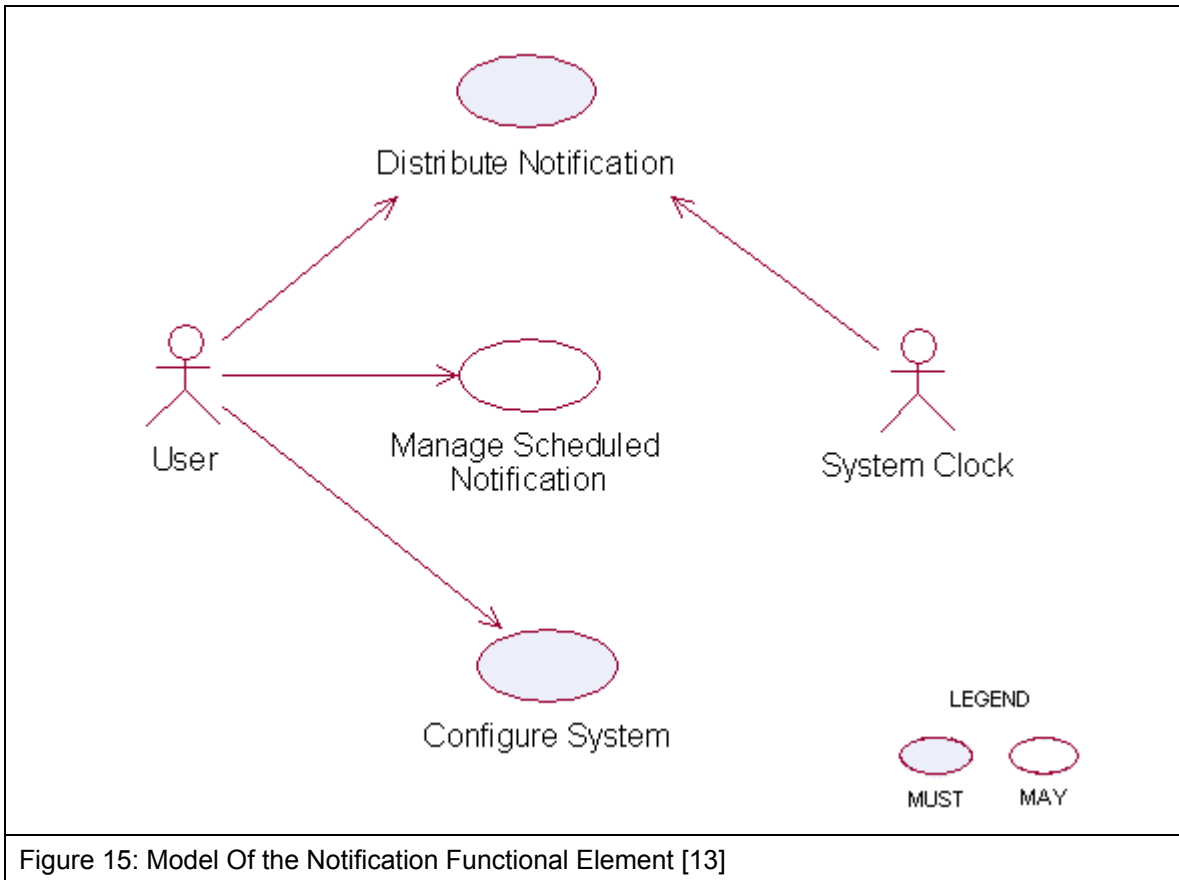
3412 None

3413 2.10.5 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.

3414

3415 **2.10.6 Model**



3416 **2.10.7 Usage Scenarios**

3417 **2.10.7.1 Distribute Notification**

3418 **2.10.7.1.1 Description**

3419 This use case allows the Functional Element to distribute messages to intended recipients.

3420 **2.10.7.1.2 Flow of Events**

3421 **2.10.7.1.2.1 Basic Flow**

3422 This use case starts when the service user or system clock wishes to send message to recipient.

3423 1: The Functional Element decides to send messages to recipients. Based on the operation
3424 specified, one of the following sub-flows is executed.

3425 If the request is **'Initiated By The User'**, then sub-flow 1.1 is executed.

3426 If the request is **'Initiated By The System Clock'** then sub-flow 1.2 is executed.

3427 1.1: Initiated By The User

3428 1.1.1: The Functional Element receives the request from the service user.

3429 1.1.2: The Functional Element validates passed parameters such as message type,
3430 recipient address, and message key and message length.

3431 1.1.3: The Functional Element checks the availability of the connection.

3432 1.1.4: The Functional Element sends message to recipient(s) and the use case end

3433 1.2 : Initiated By The System Clock

3434 1.2.1: The Functional Element checks scheduled message(s) and end date for scheduled
3435 message.

3436 1.2.2: Once the Functional Element detects scheduled messages, one of the sub-flows is
3437 executed.

- 3438 • If the Functional Element detects the scheduled notification is once, the '**Activate**
3439 **Once Notification**' sub-flow 1.2.2.1 is executed.
- 3440 • If the Functional Element detects the scheduled notification is daily, the '**Activate**
3441 **Daily Notification**' sub-flow 1.2.2.2 is executed.
- 3442 • If the Functional Element detects the scheduled notification is weekly, the
3443 '**Activate Weekly Notification**' sub-flow 1.2.2.3 is executed.
- 3444 • If the Functional Element detects the scheduled notification is Monthly, the
3445 '**Activate Monthly Notification**' sub-flow 1.2.2.4 is executed.

3446 1.2.2.1: Activate Once Notification.

3447 1.2.2.1.1: The Functional Element compares the system time with the scheduled
3448 message's time and gets notification details if both times are match.

3449 1.2.2.2: Activate Daily Notification.

3450 1.2.2.2.1: The Functional Element compares the system time with the scheduled
3451 message's time and gets notification details if both times are match.

3452 1.2.2.3: Activate Weekly Notification.

3453 1.2.2.3.1: The Functional Element compares the system date and time with the
3454 scheduled message's date and time and gets notification details if both date &
3455 time are match.

3456 1.2.2.4: Activate Monthly Notification.

3457 1.2.2.4.1: The Functional Element compares the system date and time with the
3458 scheduled message's date and time and gets notification ID if both date & time
3459 are match.

3460 1.2.3: The Functional Element extracts the list of recipient(s) and message(s).

3461 1.2.4: The Functional Element checks the availability of connection.

3462 1.2.5: The Functional Element sends message to recipient(s) and the use case ends.

3463 **2.10.7.1.2.2 Alternative Flows**

3464 1: Unsupported Message Type/Recipient Address/Message.

3465 1.1: If in basic flow 1.1.2, Functional Element detects unsupported message type, recipient
3466 address or message, the Functional Element returns an error message and the use case
3467 ends.

3468 2: Connection Fail.

3469 2.1: If in basic flow 1.1.3 and 1.2.4, the Functional Element is unable to detect connection
3470 type, the Functional Element returns an error message and the use case ends

3471 3: Delete Scheduled Message.

3472 3.1: If in basic flow 1.2.1, if the Functional Element detects that the scheduled message has
3473 expired, the Functional Element will proceed to delete those messages.

3474 **2.10.7.1.3 Special Requirements**

3475 **2.10.7.1.3.1 Supportability**

3476 None

3477 **2.10.7.1.4 Pre-Conditions**

3478 None.

3479 **2.10.7.1.5 Post-Conditions**

3480 None.

3481 **2.10.7.2 Manage Scheduled Notification**

3482 **2.10.7.2.1 Description**

3483 This use case allows the service user to maintain the notification information. This includes
3484 adding, changing and deleting notification information from the Functional Element.

3485 **2.10.7.2.2 Flow of Events**

3486 **2.10.7.2.2.1 Basic Flow**

3487 This use case starts when the service user wishes to schedule notification message(s).

3488 1: The Functional Element requests the service user to specify the function he/she would like to
3489 perform (such as create, update and delete notification message).

3490 2: Once the Functional Element user provides the requested information, one of the sub-flows is
3491 executed.

3492 If the service user provides '**Create Notification**', then sub-flow 2.1 is executed.

3493 If the service user provides '**Delete Notification**', then sub-flow 2.2 is executed.

3494 2.1 Create Notification

3495 2.1.1: The Functional Element receives the request from the service user.

3496 2.1.2: The Functional Element validates passed parameters such as schedule type,
3497 message type, recipient address, message key and the message length.

3498 2.1.3: The Functional Element generates and assigns a unique Notification ID and adds
3499 the notification information to the Functional Element and ends use case.

3500 2.2: Delete Notification

3501 2.2.1: The Functional Element requests the service user to provide the Notification
3502 information.

3503 2.2.2: The Functional Element retrieves the existing Notification information.

3504 2.2.3: The Functional Element deletes the Notification record and use case ends.

3505 **2.10.7.2.2.2 Alternative Flows**

3506 1: Invalid Parameters.

3507 1.1: If in basic flow 2.1.2, if the Functional Element detects invalid parameters such as
3508 schedule type, date & time, recipient address, message key and message, the Functional
3509 Element returns an error message and the use case ends.

3510 **2.10.7.2.3 Special Requirements**

3511 None.

3512 **2.10.7.2.4 Pre-Conditions**

3513 None.

3514 **2.10.7.2.5 Post-Conditions**

3515 If the use case was successful, the schedule message information is added to Functional
3516 Element. Otherwise, the Functional Element's state is unchanged.

3517 **2.10.7.3 Configure System**

3518 **2.10.7.3.1 Description**

3519 This use case allows the service user to maintain the notification Functional Element behaviors.
3520 This includes configuration of supported Notification Channel, Default Notification Channel,
3521 Schedule Types, and SMS and SMTP Gateway.

3522 **2.10.7.3.2 Flow of Events**

3523 **2.10.7.3.2.1 Basic Flow**

3524 1: The Functional Element requests the service user to specify or configure the function he/she
3525 would like to perform (such as create, update and delete configuration parameters).

3526 2: Once the Functional Element user provides the requested information, one of the sub-flows is
3527 executed.

3528 If user wishes to configure '**Notification Channel**', then sub-flow 2.1 is executed.

3529 If user wishes to configure '**Default Notification Channel**', then sub-flow 2.2 is executed.

3530 If user wishes to configure '**Schedule Types**', then sub-flow 2.3 is executed.

3531 If user wishes to configure '**SMTP server and SMS Gateway**', then sub-flow 2.4 is executed.

- 3532 2.1 Notification Channel.
- 3533 2.1.1: The Functional Element receives the request from the service user.
- 3534 2.1.2: The Functional Element validates passed parameters such as Notification Channel
3535 information.
- 3536 2.1.3: The Functional Element generates and assigns a unique Notification Channel ID
3537 and adds the notification information to the Functional Element and the use case ends.
- 3538 2.2: Default Notification Channel.
- 3539 2.2.1: The Functional Element requests the service user to provide the Default
3540 Notification information.
- 3541 2.2.2: The Functional Element validates passed parameters such as Default Notification
3542 Channel information.
- 3543 2.2.3: The Functional Element updates existing Default Notification or create new Default
3544 Notification information and the use case ends.
- 3545 2.3 Schedule Types.
- 3546 2.3.1: The Functional Element receives the request from the service user.
- 3547 2.3.2: The Functional Element validates passed parameters such as Schedule Type.
- 3548 2.3.3: The Functional Element generates and assigns a unique Schedule Type ID and
3549 adds the Schedule Type information to the Functional Element and the use case ends.
- 3550 2.4: SMTP server and SMS Gateway.
- 3551 2.4.1: The Functional Element requests the service user to provide the SMTP server and
3552 SMS Gateway information.
- 3553 2.4.2: The Functional Element validates passed parameters such as SMTP server and
3554 SMS Gateway information.
- 3555 2.4.3: The Functional Element updates existing SMTP server and SMS Gateway or
3556 create new SMTP server and SMS Gateway information and the use case ends.
- 3557 **2.10.7.3.2 Alternative Flows**
- 3558 1: Invalid Parameters.
- 3559 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
3560 parameters such as Notification Channel, Default Notification Channel, and SMTP server,
3561 Schedule Types and SMS Gateway information, the Functional Element returns an error
3562 message and the use case ends
- 3563 **2.10.7.3.3 Special Requirements**
- 3564 None.
- 3565 **2.10.7.3.4 Pre-Conditions**
- 3566 None.

3567 **2.10.7.3.5 Post-Conditions**

3568 None.

3569 **2.11 Phase and Lifecycle Management Functional Element**

3570 **2.11.1 Motivation**

3571 The Phase and Lifecycle Management Functional Element is expected to be an integral part of
3572 the User Access Management (UAM) functionalities that is expected to be needed by a Web
3573 Service-enabled implementation. This FE is expected to fulfill the needs arising out of managing
3574 the dynamic status of user information across the whole lifecycle. As such it will cover aspects
3575 that include:

- 3576 • Basic lifecycle management facilities,
- 3577 • Basic phase management facilities, and
- 3578 • Management of user information in phases across the whole lifecycle.

3579

3580 This Functional Element fulfills the following requirements from the Functional Elements
3581 Requirements Document 02:

- 3582 • Primary Requirements
 - 3583 ○ MANAGEMENT-070 to MANAGEMENT-078
- 3584 • Secondary Requirements
 - 3585 ○ None

3586

3587 **2.11.2 Terms Used**

Terms	Description
Group	A Group is a collection of individual users, and are typically grouped together as they have certain commonalities
Namespace	Namespace is use to segregate the instantiation of the application across different application domains. If a company has two separate standalone application, for example, an email application and an equipment booking application, then these two are considered as separate application domains
Phase/lifecycle	Phase/lifecycle refers to the phases a project goes through between when it is conceived and when it is completed. As an example, a construction related project could have the following phases: <ul style="list-style-type: none">• Project Initiation• Design• Construction• Maintenance.
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> <p>Defining a set of basic user information that should be stored in any enterprise application.</p> <p>Providing a means to extend this basic set of user information when needed..</p> <p>Simplifying management by grouping related users together through certain criteria.</p> <p>Having the flexibility of adopting both coarse/fine grain access control.</p>
------------------------------	---

3588

3589 **2.11.3 Key Features**

3590 Implementations of the Phase and Lifecycle Management Functional Element are expected to
 3591 provide the following key features:

- 3592 1. The Functional Element MUST provide basic structures based on a set of pre-defined
 3593 attributes for Lifecycle and Phase.
- 3594 2. The Functional Element MUST provide the capability to manage the creation and deletion of
 3595 instances of Lifecycle and Phase based on the pre-defined structures.
- 3596 3. The Functional Element MUST provide a means to manage the lifecycles and phases
 3597 contained within. This includes:
- 3598 3.1. The capability to retrieve and update a lifecycle or phase
- 3599 3.2. The capability to add/remove phases from a lifecycle
- 3600 4. The Functional Element MUST provide a mechanism to manage the collection of users in a
 3601 Phase. This includes:
- 3602 4.1. The capability to assign and un-assign users belonging to a Phase.
- 3603 4.2. The users could be individual Users or Groups.
- 3604 10. The Functional Element MUST provide a mechanism for managing Groups across different
 3605 application domains.

3606 *Example: Namespace control mechanism*

3607

3608 **2.11.4 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..

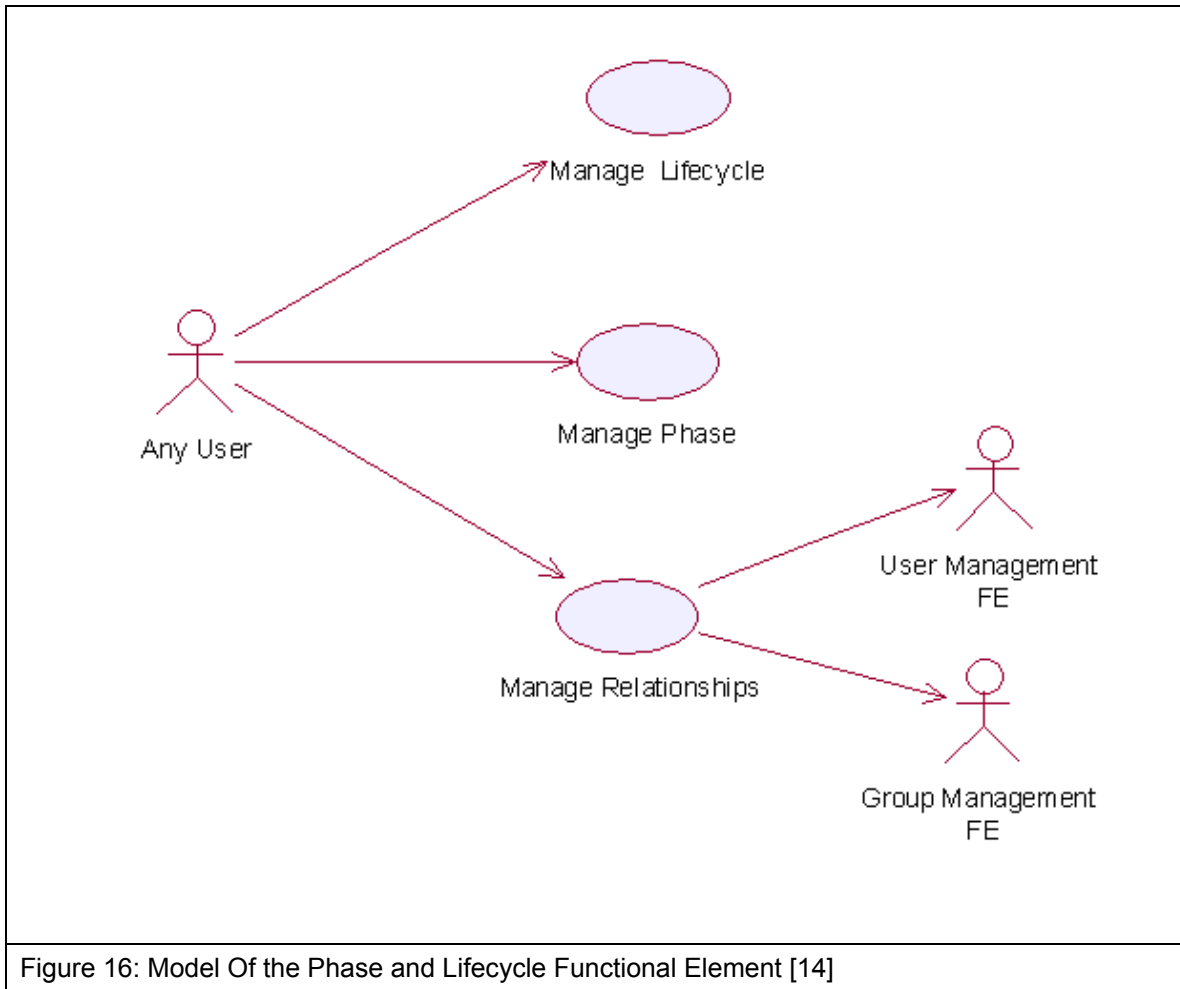
3609

3610 **2.11.5 Related Technologies and Standards**

3611 None.

3612 **2.11.6 Model**

3613



3614 **2.11.7 Usage Scenarios**

3615 **2.11.7.1 Manage Lifecycle**

3616 **2.11.7.1.1 Description**

3617 This use case is used to create, update, retrieve and delete the lifecycle.

3618 **2.11.7.1.2 Flow of Events**

3619 **2.11.7.1.2.1 Basic Flow**

3620 This use case starts when the user wants to manage phase in lifecycle.

3621 If user wants to **'Create Lifecycle'**, then basic flow 1 is executed.

3622 If user wants to **'Retrieve Lifecycle'**, then basic flow 2 is executed.

3623 If user wants to **'Update Lifecycle'**, then basic flow 3 is executed.

3624 If user wants to **'Delete Lifecycle'**, then basic flow 4 is executed.

3625 1: Create Lifecycle.

- 3626 1.1: User provides information to create lifecycle.
- 3627 1.2: Functional Element creates lifecycle and the use case ends.
- 3628 2: Retrieve Lifecycle
- 3629 2.1: User provides the lifecycle name, lifecycle namespace.
- 3630 2.2: Functional Element returns the lifecycle information and the use case ends.
- 3631 3: Update Lifecycle.
- 3632 3.1: User provides the lifecycle information.
- 3633 3.2: Functional Element updates the lifecycle-phase and the use case ends.
- 3634 4: Delete Lifecycle.
- 3635 4.1: User provides lifecycle name and lifecycle namespace.
- 3636 4.2: Functional Element deletes the lifecycle and the use case ends.
- 3637 **2.11.7.1.2.2 Alternative Flows**
- 3638 1: Lifecycle Does Not Exist.
- 3639 1.1: In basic flow 2.1, 3.1 and 4.1, if lifecycle can not be found based on lifecycle name and
3640 lifecycle namespace provided by user, Functional Element returns an error message and the
3641 use case ends.
- 3642 2: Creation Of Lifecycle Fails.
- 3643 2.1: In basic flow 1.2, if lifecycle cannot be created, the Functional Element returns an error
3644 message and the use case ends
- 3645 **2.11.7.1.3 Special Requirements**
- 3646 None.
- 3647 **2.11.7.1.4 Pre-Conditions**
- 3648 None.
- 3649 **2.11.7.1.5 Post-Conditions**
- 3650 None.
- 3651 **2.11.7.2 Manage Phase**
- 3652 **2.11.7.2.1 Description**
- 3653 This use case describes the management of different phases in a project.
- 3654 **2.11.7.2.2 Flow of Events**
- 3655 **2.11.7.2.2.1 Basic Flow**
- 3656 This use case starts when the user wants to manage phase.

- 3657 If user wants to '**Create Phase**', then basic flow 1 is executed.
- 3658 If user wants to '**Retrieve Phase**', then basic flow 2 is executed.
- 3659 If user wants to '**Update Phase**', then basic flow 3 is executed.
- 3660 If user wants to '**Delete Phase**', then basic flow 4 is executed.
- 3661 1: Create Phase.
- 3662 1.1: User provides information to create phase.
- 3663 1.2: Functional Element creates phase and the use case ends.
- 3664 2: Retrieve Phase.
- 3665 2.1: User provides phase name, lifecycle name and lifecycle namespace.
- 3666 2.2: Functional Element returns the phase information and the use case ends.
- 3667 3: Update Phase.
- 3668 3.1: User provides the phase information.
- 3669 3.2: Functional Element updates the phase and the use case ends.
- 3670 4: Delete Phase.
- 3671 4.1: User provides phase name, lifecycle name and lifecycle namespace
- 3672 4.2: Functional Element deletes phase and the use case ends.

3673 **2.11.7.2.2.2 Alternative Flows**

- 3674 1: Phase Does Not Exist.
- 3675 1.1: In basic flow 2.1, 3.1 and 4.1 if phase cannot be found based on phase name, lifecycle
- 3676 name and lifecycle namespace provided by user, Functional Element returns an error
- 3677 message and the use case ends.
- 3678 2: Creation of phase fails.
- 3679 2.1: In basic flow 1.2, if phase cannot be created, the Functional Element returns an error
- 3680 message and the use case ends

3681 **2.11.7.2.3 Special Requirements**

3682 None.

3683 **2.11.7.2.4 Pre-Conditions**

3684 None.

3685 **2.11.7.2.5 Post-Conditions**

3686 None.

3687 **2.11.7.3 Manage Relationship**

3688 **2.11.7.3.1 Description**

3689 This use case describes the management of the relationship between user/group and phase in a
3690 lifecycle.

3691 **2.11.7.3.2 Flow of Events**

3692 **2.11.7.3.2.1 Basic Flow**

3693 This use case starts when the user wants to manage the relationship between the user/group and
3694 phase.

3695 If user refers to '**Create Relationship**', basic flow 1 is executed.

3696 If user refers to '**Update Relationship**', basic flow 2 is executed.

3697 If user wants to '**Retrieve Relationship**', basic flow 3 is executed.

3698 If user refers to '**Delete Relationship**', basic flow 4 is executed.

3699 1: Create Relationship.

3700 1.1: User provides user/group, phase and phase information.

3701 1.2: Functional Element creates relationship and the use case ends.

3702 2: Update Relationship.

3703 2.1: User provides user/group name and user/group namespace.

3704 2.2: Functional Element updates the relationship and the use case ends.

3705 3: Retrieve Relationship.

3706 3.1: User provides user/group name and user/group namespace.

3707 3.2: Functional Element returns the relationship and the use case ends.

3708 4: Delete Relationship.

3709 4.1: User provides user/group name and user/group namespace.

3710 4.2: Functional Element deletes relationship between phases and users/groups and the use
3711 case ends.

3712 **2.11.7.3.2.2 Alternative Flows**

3713 1: Phase Does Not Exist.

3714 1.1: In basic flow 1,2, 2.2, 3.2 and 4.2, if the phase does not exist, the Functional Element
3715 returns an error message and the use case ends.

3716 2: User/Group Does Not Exist.

3717 1.1: In basic flow 1,2, 2.2, 3.2 and 4.2, if the user/group does not exist, the Functional
3718 Element returns an error message and the use case ends.

3719 **2.11.7.3.3 Special Requirements**

3720 None.

3721 **2.11.7.3.4 Pre-Conditions**

3722 None.

3723 **2.11.7.3.5 Post-Conditions**

3724 None.

3725 **2.12 Policy Management Functional Element (new)**

3726 **2.12.1 Motivation**

3727

3728 The Policy Management Functional Element helps enterprise to meet new challenges for IT
3729 security as the enterprise applications are now accessible from both the external partners and the
3730 customer applications. This Functional Element also helps to build consolidated view of the
3731 security configuration across all applications to ensure consistent application of a security policy
3732 across all Web Services. It also provides the mechanism for security policy management,
3733 establishment, selection and viewing for enterprises to dynamically configure the relevant policy
3734 required to protect their interests. The enforcement part is covered under Policy Enforcement
3735 Functional Element.

3736

3737 This Functional Element fulfills the following requirements from the Functional Elements
3738 Requirements Document 02:

- 3739
 - Primary Requirements
 - SECURITY-110 to SECURITY-119.
 - Secondary Requirements
 - None
- 3740
- 3741
- 3742

3743 **2.12.2 Terms Used**

Terms	Description
XACML	eXtensible Access Control Markup Language. It is an XML-based language for access control that has been standardized in OASIS

3744

```

<?xml version="1.0"?>
<policy ...>
  <xacml>
    <object href="/user_info/name" />
    <rule>
      <acl>
        <subject>
          <uid>normal_user</uid>
        </subject>
        <action name="read" permission="permit" />
      </acl>
    </rule>
  </xacml>
  <xacml>
    <object href="/user_info/salary" />
    <rule>
      <acl>
        <subject>
          <uid>supervisor</uid>
        </subject>
        <action name="read" permission="permit" />
        <action name="write" permission="permit" />
      </acl>
    </rule>
  </xacml>
</policy>

```

Permit "normal_user" read access to "name"

Permit "supervisor" read and write access to "salary"

Figure 17: An example of an security policy in XACML format

3745

3746 Figure 17 shows an example of a security policy used in Policy Management Functional Element.
 3747 The security policy is in XACML format.

3748 2.12.3 Key Features

3749 Implementations of the Policy Management Functional Element are expected to provide the
 3750 following key features:

- 3751 11. The Functional Element MUST provide the capability to define and manage Policy
 3752 Categories.
- 3753 12. The Functional Element MUST provide the capability to define and manage Policies.
- 3754 13. The Functional Element MUST provide version control capability to defined Policies.
- 3755 14. The Functional Element MUST provide the ability to manage Policies within a Policy
 3756 Category; including insertion, update, retrieval and removal of attached Policies.
- 3757 15. The Functional Element MUST provide the ability to retrieve Policies that are attached to a
 3758 Policy Category.

3759

3760 In addition, the following key feature could be provided to enhance the Functional Element
 3761 further:

- 3762 1. The Functional Element MAY provide the ability to translate Policy into multi-lingual.

3763

3764

2.12.4 Interdependency

Direct Dependency	
Policy Enforcement Functional Element	The Policy Enforcement Functional Element provides the mechanism to enforce the policy associated to a service. The enforcement is based on a pre-identified access structure. The access structure could be provided by the Role & Access Management Functional Element.
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.
Role & 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.

3765

3766

2.12.5 Related Technologies and Standards

Standards / Specifications	Specific References
Extensible Access Control Markup Language (XACML) v1.0	Extensible Access Control Markup Language (XACML) v1.0 [OASIS 200301], February 2003.

3767

2.12.6 Model

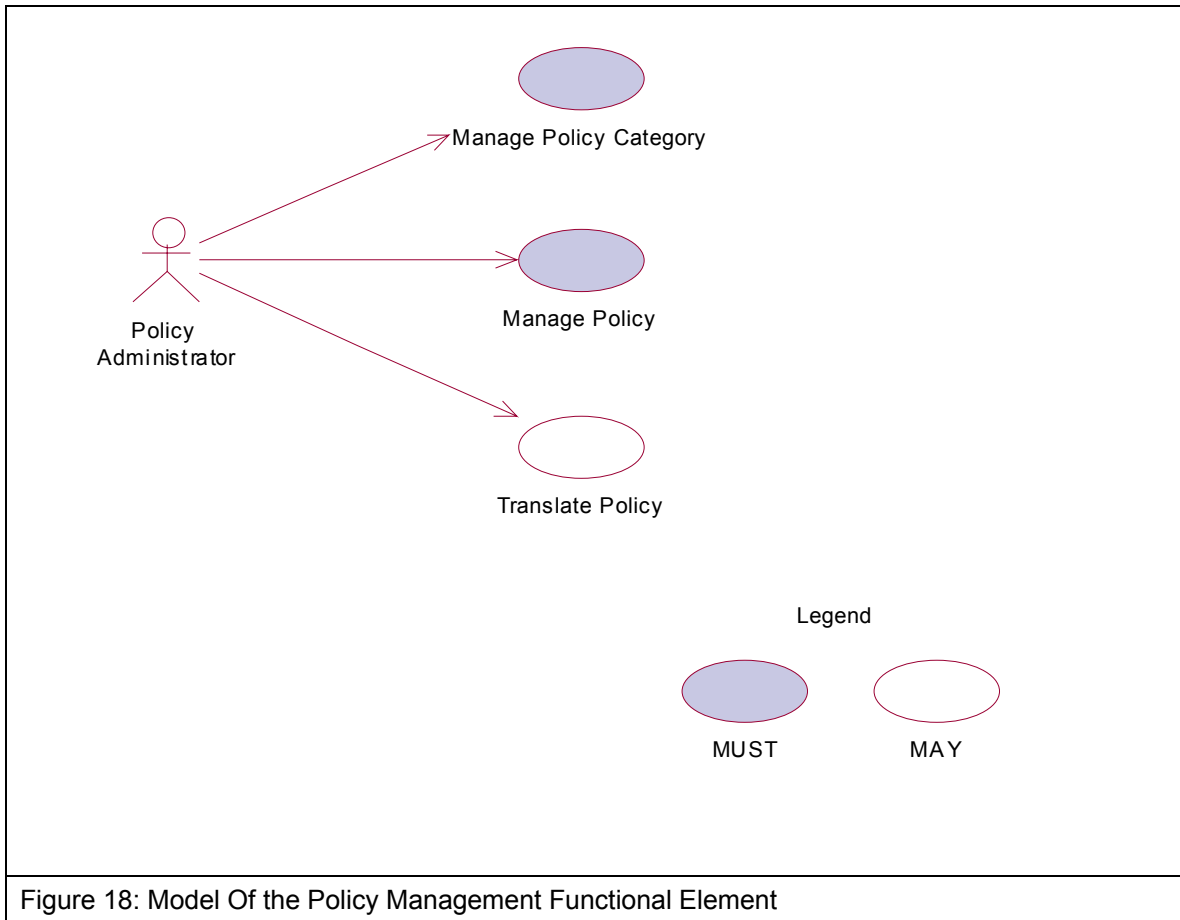


Figure 18: Model Of the Policy Management Functional Element

3768

3769 2.12.7 Usage Scenarios

3770 2.12.7.1 Manage Policy Category

3771 2.12.7.1.1 Description

3772 This use case allows the policy administrator to manage policy category.

3773 2.12.7.1.2 Flow of Events

3774 2.12.7.1.2.1 Basic Flow

3775 The use case begins when the policy administrator wants to create/retrieve/update/delete a policy
3776 category.

3777 1: The policy administrator sends a request to manipulate a policy category.

3778 2: Based on the operation it specifies, one of the following sub-flows is executed:

3779 If the operation is 'Create Policy Category', the sub-flow 2.1 is executed.

3780 If the operation is 'Retrieve Policy Category', the sub-flow 2.2 is executed.

3781 If the operation is '**Update Policy Category**', the sub-flow 2.3 is executed.
3782 If the operation is '**Delete Policy Category**', the sub-flow 2.4 is executed.

3783 2.1: Create Policy Category.

3784 2.1.1: The Functional Element gets the category name and definition.
3785 2.1.2: The Functional Element checks whether the category exists.
3786 2.1.3: The Functional Element creates the category.

3787 2.2: Retrieve Policy Category.

3788 2.2.1: The Functional Element gets the category name.
3789 2.2.2: The Functional Element checks whether the category exists.
3790 2.2.3: The Functional Element retrieves the category's information.

3791 2.3: Update Policy Category.

3792 2.3.1: The Functional Element gets the category name and definition.
3793 2.3.2: The Functional Element checks whether the category exists.
3794 2.3.3: The Functional Element updates the category's information.

3795 2.4: Delete Policy Category.

3796 2.4.1: The Functional Element gets the category name.
3797 2.4.2: The Functional Element checks whether the category exists.
3798 2.4.3: The Functional Element removes the category.

3799 3: The Functional Element returns the results of the operation to the policy administrator and the
3800 use case ends.

3801 **2.12.7.1.2.2 Alternative Flows**

3802 1: Category Already Exists.

3803 1.1: If in the basic flow 2.1.2, the category is already defined, Functional Element returns an
3804 error message and the use case ends.

3805 2: Category Not Found.

3806 2.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the category does not exist, Functional Element
3807 returns an error message and the use case ends.

3808 **2.12.7.1.3 Special Requirements**

3809 None.

3810 **2.12.7.1.4 Pre-Conditions**

3811 None.

3812 **2.12.7.1.5 Post-Conditions**

3813 None.

3814

3815 **2.12.7.2 Manage Policy**

3816 **2.12.7.2.1 Description**

3817 This use case allows the policy administrator to manage policy.

3818 **2.12.7.2.2 Flow of Events**

3819 **2.12.7.2.2.1 Basic Flow**

3820 The use case begins when the policy administrator wants to create/retrieve/update/delete a
3821 policy.

3822 1: The policy administrator sends a request to manipulate a policy.

3823 2: Based on the operation it specifies, one of the following sub-flows is executed:

3824 If the operation is '**Create Policy**', the sub-flow 2.1 is executed.

3825 If the operation is '**Retrieve Policy**', the sub-flow 2.2 is executed.

3826 If the operation is '**Update Policy**', the sub-flow 2.3 is executed.

3827 If the operation is '**Delete Policy**', the sub-flow 2.4 is executed.

3828 2.1: Create Policy.

3829 2.1.1: The Functional Element gets the policy name, content and the Policy Category
3830 where the policy is to be created.

3831 2.1.2: The Functional Element checks whether the policy exists.

3832 2.1.3: The Functional Element creates the policy.

3833 2.2: Retrieve Policy.

3834 2.2.1: The Functional Element gets the policy name and the Policy Category.

3835 2.2.2: The Functional Element checks whether the policy exists.

3836 2.2.3: The Functional Element retrieves the policy's information.

3837 2.3: Update Policy.

3838 2.3.1: The Functional Element gets the policy name, information and the Policy Category.

3839 2.3.2: The Functional Element checks whether the policy exists.

3840 2.3.3: The Functional Element updates the policy.

3841 2.4: Delete Policy.

3842 2.4.1: The Functional Element gets the policy name and the Policy Category.

3843 2.4.2: The Functional Element checks whether the policy exists.

3844 2.4.3: The Functional Element removes the policy from the Policy Category.

3845 3: The Functional Element returns the results of the operation to the policy administrator and the
3846 use case ends.

3847 **2.12.7.2.2 Alternative Flows**

3848 1: Policy Already Exists.

3849 1.1: If in the basic flow 2.1.2, the policy is already created, Functional Element returns an
3850 error message and the use case ends.

3851 2: Policy Not Found.

3852 2.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the policy does not exist, Functional Element
3853 returns an error message and the use case ends.

3854 **2.12.7.2.3 Special Requirements**

3855 None.

3856 **2.12.7.2.4 Pre-Conditions**

3857 None.

3858 **2.12.7.2.5 Post-Conditions**

3859 None.

3860

3861 **2.12.7.3 Translate Policy**

3862 **2.12.7.3.1 Description**

3863 This use case allows the policy administrator to translate policy into desired languages.

3864 **2.12.7.3.2 Flow of Events**

3865 **2.12.7.3.2.1 Basic Flow**

3866 The use case begins when the policy administrator wants to translate a policy.

3867 1: The policy administrator sends a request to translate a policy.

3868 2: The Functional Element gets the policy name and the language desired.

3869 3: The Functional Element checks whether the policy exists.

3870 4: The Functional Element retrieves the policy for translation.

3871 5: The Functional Element returns the results of the operation to the policy administrator and the
3872 use case ends.

3873 **2.12.7.3.2.2 Alternative Flows**

3874 1: Policy Not Found.

3875 1.1: If in the basic flow 3, the policy does not exist, Functional Element returns an error
3876 message and the use case ends.

3877 **2.12.7.3.3 Special Requirements**

3878 None.

3879 **2.12.7.3.4 Pre-Conditions**

3880 None.

3881 **2.12.7.3.5 Post-Conditions**

3882 None.

3883 **2.13 Policy Enforcement Functional Element (new)**

3884 **2.13.1 Motivation**

3885 The Policy Enforcement Functional Element helps enterprise to enforce policy for both the
3886 external partners and the customer applications that are authorized to access the enterprise
3887 applications. This Functional Element helps to ensure that the enterprise's interests and its
3888 confidential information are protected.

3889

3890 This Functional Element fulfills the following requirements from the Functional Elements
3891 Requirements Document 02:

- 3892 • Primary Requirements
 - 3893 ○ SECURITY-140 to SECURITY-144
- 3894 • Secondary Requirements
 - 3895 ○ None

3896

3897 **2.13.2 Terms Used**

Terms	Description
XACML	eXtensible Access Control Markup Language. It is an XML-based language for access control that has been standardized in OASIS

3898

3899 **2.13.3 Key Features**

3900 Implementations of the Policy Enforcement Functional Element are expected to provide the
3901 following key features:

- 3902 1. The Functional Element MUST provide the ability to identify Policy Categories and/or
3903 Policies that are to be enforced.
- 3904 2. The Functional Element MUST provide the ability to access enforced Policies for
3905 accepting/rejecting the policy.
- 3906 3. The Functional Element MUST provide the ability to associate a policy to a service.
- 3907 4. The Functional Element MUST provide the capability to associate a policy to its service's
3908 access privileges through a pre-identified Access structure.
- 3909 5. The Functional Element MUST provide a mechanism to enforce policy upon acceptance of
3910 the policy.
- 3911 6. The Functional Element MUST provide the ability to enforce policies either based on
3912 individual or groups of services.
- 3913 7. The Functional Element MUST provide the capability to reject access.

3914 **2.13.4 Interdependency**

Direct Dependency	
Policy Management Functional Element	The Policy Management Functional Element provides the mechanism for security policy management, establishment, selection and viewing for enterprises to dynamically configure the relevant policy required to protect their interests.
Role & Access Management Functional Element	The Role & Access Management Functional Element may be used to manage the user's access rights by virtue of its association with a group, phase or even the complete lifecycle of the project.

3915

3916 **2.13.5 Related Technologies and Standards**

Standards / Specifications	Specific References
Extensible Access Control Markup Language (XACML) v1.0	Extensible Access Control Markup Language (XACML) v1.0 [OASIS 200301], February 2003.

3917

3918 **2.13.6 Model**

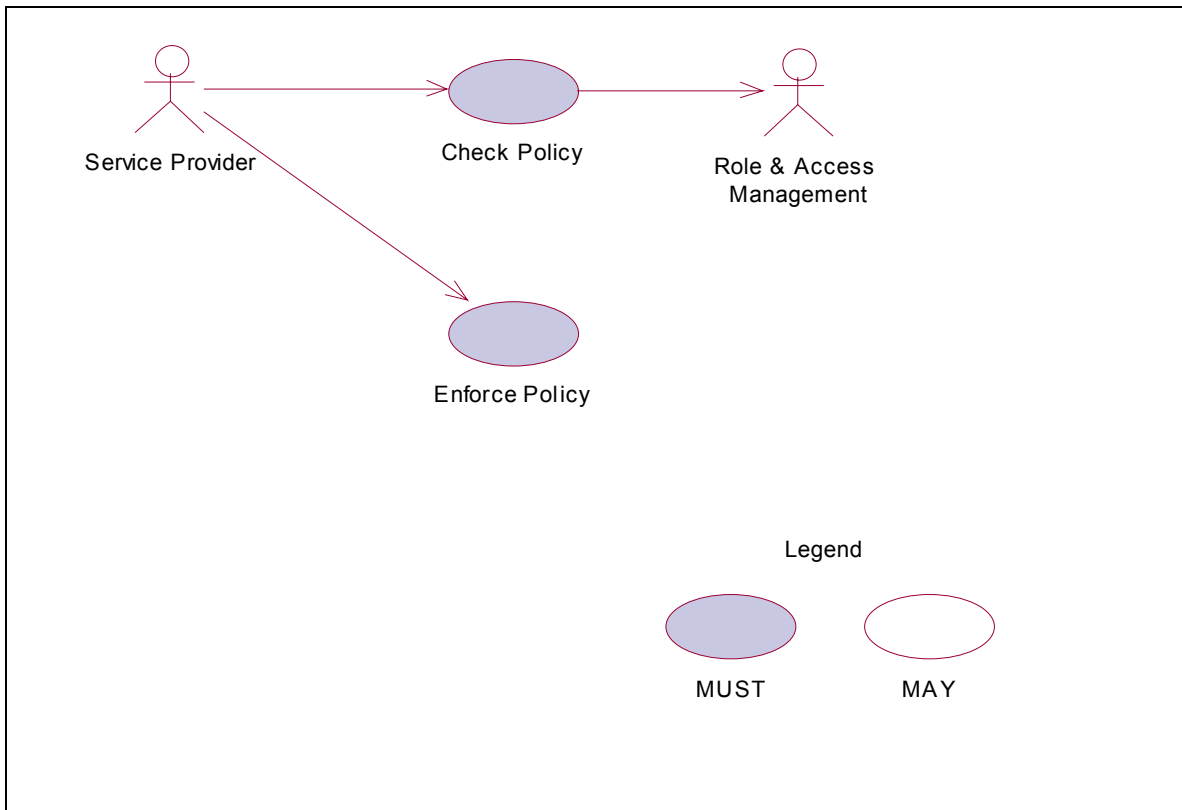


Figure 19: Model Of the Policy Enforcement Functional Element

3919

- 3920 **2.13.7 Usage Scenarios**
- 3921 **2.13.7.1 Check Policy**
- 3922 **2.13.7.1.1 Description**
- 3923 This use case allows the service provider to check policy.
- 3924 **2.13.7.1.2 Flow of Events**
- 3925 **2.13.7.1.2.1 Basic Flow**
- 3926 The use case begins when the service provider wants to check a policy.
- 3927 1: The service provider sends a request to check a policy.
- 3928 2: The Functional Element gets the policy and the requested service names.
- 3929 3: The Functional Element checks whether the policy and the requested service exist.
- 3930 4: The Functional Element evaluates the policy.
- 3931 5: The Functional Element resolves any policy conflict.
- 3932 6: The Functional Element returns the outcome to the service provider and the use case ends.
- 3933 **2.13.7.1.2.2 Alternative Flows**
- 3934 1: Policy Not Found.
- 3935 1.1: If in the basic flow 3, the policy does not exist, Functional Element returns an error
- 3936 message and the use case ends.
- 3937 2: Requested Service Not Found.
- 3938 2.1: If in the basic flow 3, the requested service does not exist, Functional Element returns an
- 3939 error message and the use case ends.
- 3940 3: Cannot Evaluate Policy.
- 3941 3.1: If in the basic flow 4, the policy cannot be evaluated, Functional Element returns an error
- 3942 message and the use case ends.
- 3943 4: Cannot Resolve Policy Conflict.
- 3944 4.1: If in the basic flow 5, the policy conflict cannot be resolved, Functional Element returns
- 3945 an error message and the use case ends.
- 3946 **2.13.7.1.3 Special Requirements**
- 3947 None.
- 3948 **2.13.7.1.4 Pre-Conditions**
- 3949 None.

3950 **2.13.7.1.5 Post-Conditions**

3951 None.
3952

3953 **2.13.7.2 Enforce Policy**

3954 **2.13.7.2.1 Description**

3955 This use case allows the service provider to enforce policy based on the pre-identified access
3956 structure.

3957 **2.13.7.2.2 Flow of Events**

3958 **2.13.7.2.2.1 Basic Flow**

3959 The use case begins when the service provider wants to enforce policy on a specific service.

3960 1: The service provider sends a request to enforce a policy.

3961 2: The Functional Element gets the policy name and the service activated.

3962 3: The Functional Element checks whether the policy and the service exist.

3963 4: The Functional Element gets the decision outcome.

3964 5: The Functional Element enforces the policy to the service and the use case ends.

3965 **2.13.7.2.2.2 Alternative Flows**

3966 1: Policy Not Found.

3967 1.1: If in the basic flow 3, the policy does not exist, Functional Element returns an error
3968 message and the use case ends.

3969 2: Service Not Found.

3970 2.1: If in the basic flow 3, the service does not exist, Functional Element returns an error
3971 message and the use case ends.

3972 3: Cannot Make Decision.

3973 3.1: If in the basic flow 4, decision cannot be made based on the information provided,
3974 Functional Element returns an error message and the use case ends.

3975 **2.13.7.2.3 Special Requirements**

3976 None.

3977 **2.13.7.2.4 Pre-Conditions**

3978 None.

3979 **2.13.7.2.5 Post-Conditions**

3980 None.
3981

3982 **2.14 Presentation Transformer Functional Element**

3983 **[Deprecated]**

3984 This Functional Element has been deprecated in this version. Please refer to its replacement,
3985 2.26 Transformer Functional Element (new) for further details.

3986 **2.15 Quality of Service (QoS) Functional Element (new)**

3987 **2.15.1 Motivation**

3988 With the widespread use of Web Services, Quality of Service (QoS) becomes a significant factor
3989 in distinguishing the success of service providers. On the other hand, poor QoS can potentially
3990 translate into frustrated customers, which in-turn lead to lost business opportunities. QoS
3991 determines the service usability and utility, both of which influence the popularity of the service.

3992

3993 This Functional Element fulfills the following requirements from the Functional Elements
3994 Requirements Document 02:

- 3995
- Primary Requirements
 - 3996 ○ MANAGEMENT-320,
 - 3997 ○ MANAGEMENT-321,
 - 3998 ○ MANAGEMENT-323,
 - 3999 ○ MANAGEMENT-324,
 - 4000 ○ [MANAGEMENT-325 and
 - 4001 ○ MANAGEMENT-312.
 - Secondary Requirements
 - 4002 ○ MANAGEMENT-311 and
 - 4003 ○ MANAGEMENT-310.

4005

4006 **2.15.2 Terms Used**

Terms	Description
Availability	Availability refers to the quality aspect of whether the Web Service is present or ready for immediate use.
Performance	Performance refers to the quality aspect of Web service. It is measured in terms of throughput and latency. Higher throughput and lower latency values represent good performance of a Web Service.
Reliability	Reliability refers to the quality aspect of a Web Service that represents the degree of being capable of maintaining the service and service quality.
Accessibility	Accessibility refers to the quality aspect of a service that represents the degree it is capable of serving a Web service request. It denotes the success rate or chance of a successful service instantiation at a point in time.
Security	Security is the quality aspect of the Web service of providing confidentiality and non-repudiation by authenticating the parties involved, encrypting messages, and providing access control

4007

4008 Figure 20 depicts the basic concepts of 2 steps approach of QoS Functional Element. Step 1
4009 begins when the user (service requester) requests to measure QOS of a known web service. The
4010 Function Element then returns a Reference ID once it receives that request. It also takes

4011 necessary measurements and logs them. Step 2 begins when the user requests for the result of
 4012 measurement. The user provides the Functional Element a Reference ID. With this Reference
 4013 ID, the Functional Element calculates and returns the result to the user.
 4014

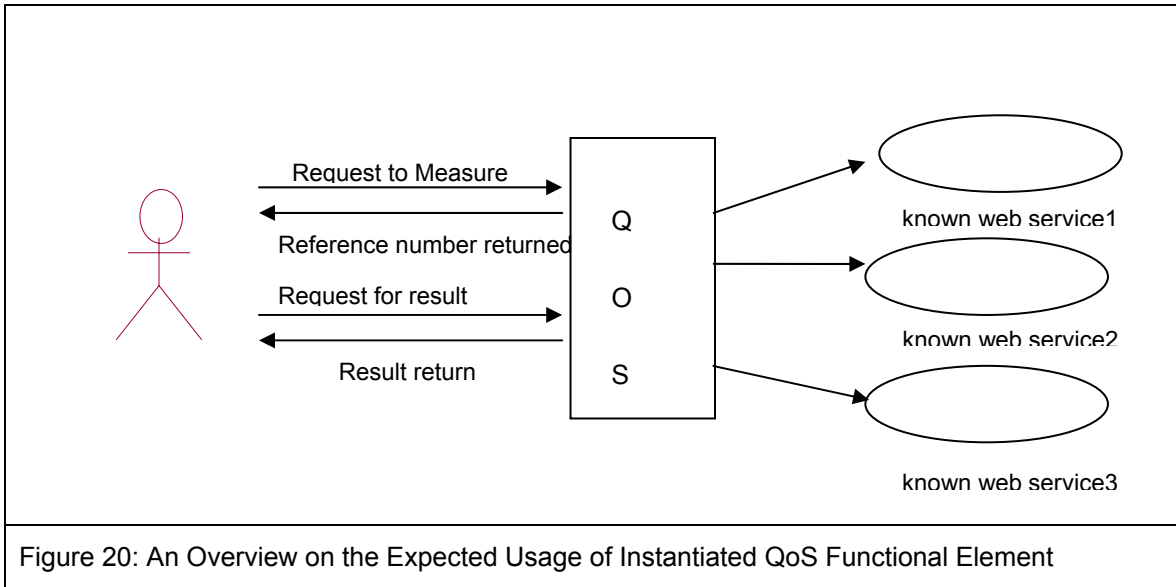


Figure 20: An Overview on the Expected Usage of Instantiated QoS Functional Element

4015

4016 2.15.3 Key Features

4017 Implementations of the QoS Functional Element are expected to provide the following key
 4018 features:

- 4019 1. The Functional Element MUST provide the capability to measure Availability.
- 4020 2. The Functional Element MUST provide the capability to measure Performance.
- 4021 3. The Functional Element MUST provide the capability to measure Reliability.
- 4022 4. The Functional Element MUST provide the capability to measure Accessibility.

4023

4024 In addition, the following key features could be provided to enhance the Functional Element
 4025 further:

- 4026 1. The Functional Element MAY provide confidentiality and non-repudiation by authenticating
 4027 the parties involved, encrypting messages and providing access control as in the Security
 4028 aspect.

4029

4030 2.15.4 Interdependencies

Direct Dependency	
Log Utility Functional Element	The Log Utility Functional Element is used to record the data.

4031

4032

4033

4034

Interaction Dependency

Secure SOAP Management Functional Element

The Secure SOAP Management Functional Element is used to provide authentication to the user, encrypting messages and providing access control

4035

2.15.5 Related Technologies and Standards

Specifications

Description

WSDL 1.1

The ability to parse the WSDL document and generate a client is heavily dependent on it being a conforming WSDL document.

4037

2.15.6 Model

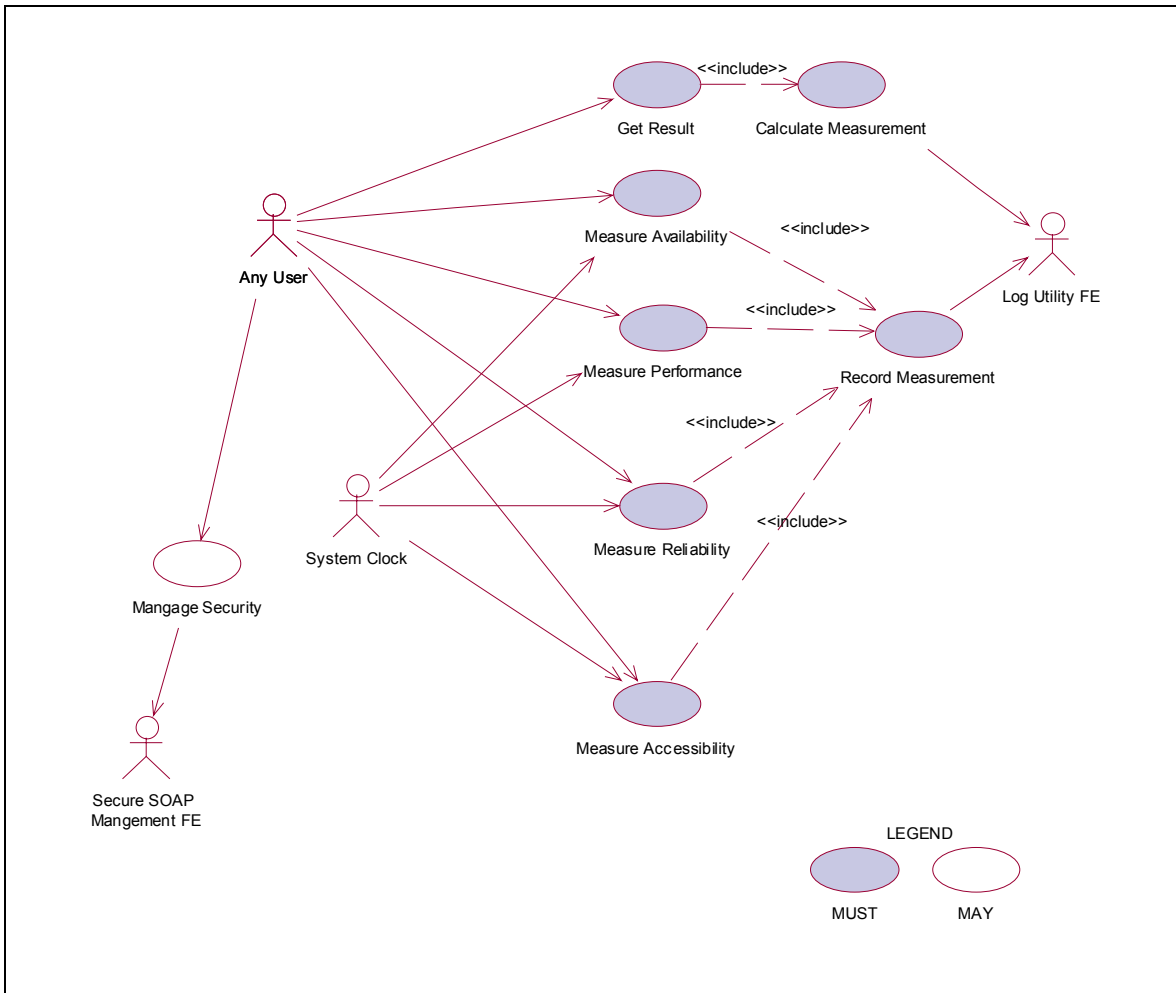


Figure 21: Model Of the QoS Functional Element

4039

4040 **2.15.7 Usage Scenarios**

4041 **2.15.7.1 Measure Availability**

4042 **2.15.7.1.1 Description**

4043 This use case allows the user to measure the availability of a known Web Service. User sets a
4044 period of measurement and the frequency of invocation. The result of this measure is in
4045 percentage. It is derived by using the successful invocations divided by total number of
4046 invocations for the given period of measurement.

4047
$$\text{Total uptime} - \text{downtime} / \text{Total uptime} \times 100 \%$$

4048
$$= ((\text{number of successful invocation} \times \text{frequency of invocation}) / \text{period of measurement}) \times 100\%$$

4049
$$= (\text{number of successful invocations} / \text{total invocations}) \times 100\%$$

4050

4051 **2.15.7.1.2 Flow of Events**

4052 **2.15.7.1.2.1 Basic Flow**

4053 This use case starts when the user wants to measure the availability of a Web Service.

- 4054 1. User sets a period of measurement.
- 4055 2. User determines the acceptable invocation interval.
- 4056 3. User submits the WSDL of a known web service.
- 4057 4. Functional Element parses the URL of the WSDL document and extracts the necessary
4058 information.
- 4059 5. Functional Element generates client base on the extracted information.
- 4060 6. Functional Element invokes the known web service using the generated client
- 4061 7. Functional Element generates a Reference ID.
- 4062 8. Functional Element returns Reference ID to the user.
- 4063 9. Functional Element logs the Reference ID to the Record Measurement Use Case.
- 4064 10. Functional Element logs the Measurement Type to the Record Measurement Use Case.
- 4065 11. Functional Element logs each invocation at every interval to the Record Measurement Use
4066 Case.
- 4067 12. Functional Element logs successful invocation at every interval to the Record Measurement
4068 Use Case.
- 4069 13. Functional Element continues to invoke the known web service at every interval until the
4070 period of measurement is reached and the use case ends.

4071 **2.15.7.1.2.2 Alternative Flows**

- 4072 1. If the structure of the WSDL does not comply with the standard, the Functional Element
4073 returns an error message and the use case ends.

- 4074 2. If the Functional Element fails to generate the client, the Functional Element returns an error
4075 message and the use case ends.
- 4076 3. If the Functional Element fails to find the known web service, the Functional Element returns
4077 an error message and the use case ends.
- 4078 4. If the Functional Element fails to invoke the known web service, the Functional Element
4079 returns an error message and the use case ends.
- 4080 5. If the Functional Element fails to return a reference ID, the Functional Element returns an
4081 error message and the use case ends.
- 4082 6. If the Functional Element gets a wrong a reference ID, the Functional Element returns an
4083 error message and the use case ends.

4084 **2.15.7.1.3 Special Requirements**

4085 None.

4086 **2.15.7.1.4 Pre-Conditions**

4087 None

4088 **2.15.7.1.5 Post-Conditions**

4089 None.

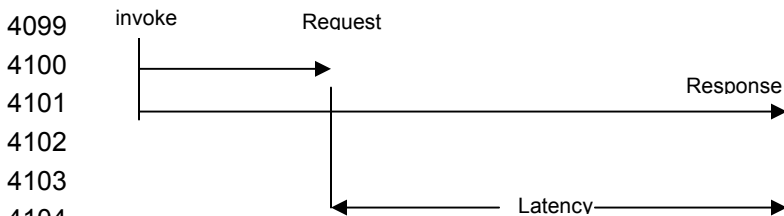
4090

4091 **2.15.7.2 Measure Performance**

4092 **2.15.7.2.1 Description**

4093 This use case allows the user to measure the performance of a Web Service. In Performance
4094 both Latency and Throughput are measured. For throughput, user sets a period of measurement.
4095 Throughput is derived as the total number of invocations for the given period of measurement.
4096 For Latency, user logs the request time and response time of invocation. Latency is derived by
4097 the response time minus the request time of the invocation, as indicated below:

4098



4106 **2.15.7.2.2 Flow of Events**

4107 **2.15.7.2.2.1 Basic Flow**

4108 This use case starts when a user wants to measure the Performance of a Web Service.

- 4109 1. Based on the operation it specified, one of the following sub-flows is expected
- 4110 • If the operation is 'Measure Throughput', then sub-flow 1.1 is executed.

- 4111 • If the operation is 'Measure Latency', then sub-flow 1.2 is executed.
- 4112 1.1. Measure Throughput
- 4113 This use case starts when a user wants to measure the Throughput of a Web
- 4114 Service.
- 4115 1.1.1. User sets a period of measurement.
- 4116 1.1.2. User submits the WSDL of a known web service.
- 4117 1.1.3. Functional Element parses the URL of the WSDL document and extracts the
- 4118 necessary information.
- 4119 1.1.4. Functional Element generates a Reference ID.
- 4120 1.1.5. Functional Element returns a Reference ID to user.
- 4121 1.1.6. Functional Element logs the Reference ID to the Record Measurement Use
- 4122 Case.
- 4123 1.1.7. Functional Element logs the measurement type to the Record Measurement
- 4124 Use Case.
- 4125 1.1.8. Functional Element waits and logs any invocation to this WSDL until the
- 4126 period of measurement is reached and the use case ends.
- 4127 1.2. Measure Latency
- 4128 1.2.1. User submits the WSDL of a known web service.
- 4129 1.2.2. Functional Element parses URL of the WSDL document and extracts the
- 4130 necessary information.
- 4131 1.2.3. Functional Element invokes the known web service.
- 4132 1.2.4. Functional Element generates a Reference ID.
- 4133 1.2.5. Functional Element returns a Reference ID to user.
- 4134 1.2.6. Functional Element logs the Reference ID to the Record Measurement Use
- 4135 Case.
- 4136 1.2.7. Functional Element logs the measurement type to the Record Measurement
- 4137 Use Case.
- 4138 1.2.8. Functional Element logs the request time to the Record Measurement Use
- 4139 Case.
- 4140 1.2.9 Functional Element logs the response time to the Record Measurement Use
- 4141 Case and the use case ends.
- 4142 **2.15.7.2.2.2 Alternative Flows**
- 4143 1. If the structure of the WSDL does not comply with the standard, the Functional Element
- 4144 returns an error message and the use case ends.
- 4145 2. If the Functional Element fails to generate the client, the Functional Element returns an error
- 4146 message and the use case ends.

- 4147 3. If the Functional Element fails to find the known web service, the Functional Element returns
4148 an error message and the use case ends.
- 4149 4. If the Functional Element fails to invoke the known web service, the Functional Element
4150 returns an error message and the use case ends.
- 4151 5. If the Functional Element fails to return a reference ID, the Functional Element returns an
4152 error message and the use case ends.
- 4153 6. If the Functional Element gets a wrong a reference ID, the Functional Element returns an
4154 error message and the use case ends.

4155 **2.15.7.2.3 Special Requirements**

4156 None.

4157 **2.15.7.2.4 Pre-Conditions**

4158 None.

4159 **2.15.7.2.5 Post-Conditions**

4160 None.

4161

4162 **2.15.7.3 Measure Reliability**

4163 **2.15.7.3.1 Description**

4164 This use case allows the user to measure the reliability of a known Web Service. User sets a
4165 period of measurement. The number of failures over a period of time is the measure of Reliability.
4166 It is derived as the unsuccessful invocations for the given period of measurement.

4167 **2.15.7.3.2 Flow of Events**

4168 **2.15.7.3.2.1 Basic Flow**

- 4169 1. User sets a period of measurement.
- 4170 2. User submits the WSDL of a known web service.
- 4171 3. Functional Element parses the URL of the WSDL document and extracts the necessary
4172 information.
- 4173 4. Functional Element generates a Reference ID.
- 4174 5. Functional Element returns a Reference ID to user.
- 4175 6. Functional Element logs the Reference ID to the Record Measurement Use Case.
- 4176 7. Functional Element logs measurement type to the Record Measurement Use Case.
- 4177 8. Functional Element waits for any invocation to the known WSDL.
- 4178 9. Functional Element logs unsuccessful invocations to this WSDL until the period of
4179 measurement is reached and the use case ends.

4180 **2.15.7.3.2 Alternative Flows**

- 4181 1. If the structure of the WSDL does not comply with the standard, the Functional Element
4182 returns an error message and the use case ends.
- 4183 2. If the Functional Element fails to generate the client, the Functional Element returns an error
4184 message and the use case ends.
- 4185 3. If the Functional Element fails to find the known web service, the Functional Element returns
4186 an error message and the use case ends.
- 4187 4. If the Functional Element fails to invoke the known web service, the Functional Element
4188 returns an error message and the use case ends.
- 4189 5. If the Functional Element fails to return a reference ID, the Functional Element returns an
4190 error message and the use case ends.
- 4191 6. If the Functional Element gets a wrong a reference ID, the Functional Element returns an
4192 error message and the use case ends.

4193 **2.15.7.3.3 Special Requirements**

4194 None.

4195 **2.15.7.3.4 Pre-Conditions**

4196 None.

4197 **2.15.7.3.5 Post-Conditions**

4198 None.

4199 **2.15.7.4 Measure Accessibility**

4200 **2.15.7.4.1 Description**

4201 This use case allows the user to measure the accessibility of a known Web Service. It is a
4202 measure denoting the success rate or chance of a successful service instantiation at a point of
4203 time. User sets the number of times of invocations. User invokes the known web service at the
4204 number of times set by the user at one go. The result of this measure is in percentage. It is
4205 derived by using the successful invocations divided by total invocations for the given period of
4206 measurement:

4207 $success\ rate = successful\ invocations / total\ invocations \times 100\%$ (invocations are fired simultaneously)

4208

4209 **2.15.7.4.2 Flow of Events**

4210 **2.15.7.4.2.1 Basic Flow**

- 4211 1. User sets number of invocations.
- 4212 2. User submits the WSDL of a known web service.
- 4213 3. Functional Element parses the URL of the WSDL document and extracts the necessary
4214 information.
- 4215 4. Functional Element generates client base on the extracted information.

- 4216 5. Functional Element invokes the known web service simultaneously at the number of times set
4217 by the user using the generated client.
- 4218 6. Functional Element generates a Reference ID.
- 4219 7. Functional Element returns a Reference ID to user.
- 4220 8. Functional Element logs the Reference ID to the Record Measurement Use Case.
- 4221 9. Functional Element logs measurement type to the Record Measurement Use Case.
- 4222 10. Functional Element logs each invocation to the Record Measurement Use Case.
- 4223 11. Functional Element logs successful invocation and the use case ends.

4224 **2.15.7.4.2 Alternative Flows**

- 4225 1. If the structure of the WSDL does not comply with the standard, the Functional Element
4226 returns an error message and the use case ends.
- 4227 2. If the Functional Element fails to generate the client, the Functional Element returns an error
4228 message and the use case ends.
- 4229 3. If the Functional Element fails to find the known web service, the Functional Element returns
4230 an error message and the use case ends.
- 4231 4. If the Functional Element fails to invoke the known web service, the Functional Element
4232 returns an error message and the use case ends.
- 4233 5. If the Functional Element fails to return a reference ID, the Functional Element returns an
4234 error message and the use case ends.
- 4235 6. If the Functional Element gets a wrong a reference ID, the Functional Element returns an
4236 error message and the use case ends.

4237 **2.15.7.4.3 Special Requirements**

4238 None.

4239 **2.15.7.4.4 Pre-Conditions**

4240 None.

4241 **2.15.7.4.5 Post-Conditions**

4242 None.

4243

4244

4245 **2.15.7.5 Record Measurement**

4246 **2.15.7.5.1 Description**

4247 This use case records the Measurement taken. It records type of Measurement, Reference ID,
4248 and the invocation data (invocation status (Successful or Unsuccessful), request time and
4249 response time)

4250 **2.15.7.5.2 Flow of Events**

4251 **2.15.7.5.2.1 Basic Flow**

4252 This use case starts when the user record the Measurement.

- 4253 1. The Functional Element logs Reference ID into a log file using Log Utility FE.
- 4254 2. The Functional Element logs Measurement type into a log file using Log Utility FE.
- 4255 3. The Functional Element logs the invocation data into a log file using Log Utility FE.

4256 **2.15.7.5.2.2 Alternate Flow**

- 4257 1. Log file not available, the Functional Element returns an error and the user case ends.
- 4258 2. If the Functional Element fails to get a reference ID, the Functional Element returns an error
4259 message and the use case ends.

4260 **2.15.7.5.3 Special Requirements**

4261 None.

4262 **2.15.7.5.4 Pre-Conditions**

4263 None.

4264 **2.15.7.5.5 Post-Conditions**

4265 None.

4266

4267 **2.15.7.6 Calculate Measurement**

4268 **2.15.7.6.1 Description**

4269 This use case calculates the Measurement.

4270 **2.15.7.6.2 Flow of Events**

4271 **2.15.7.6.2.1 Basic Flow**

4272 This use case starts when user wants to calculate Measurement.

- 4273 1. The Functional Element gets the Reference ID.
- 4274 2. The Functional Element opens up the log file.
- 4275 3. The Functional Element reads the data in the log file base on Reference ID given.
- 4276 4. The Functional Element calculates the measurement using the data read from the log file.
- 4277 5. The Functional Element sends the calculated result to the user.

4278 **2.15.7.6.2.2 Alternative Flows**

- 4279 1. Log file not available, the Functional Element returns an error and the user case ends.

4280 2. If the Functional Element fails to get a reference ID, the Functional Element returns an error
4281 message and the use case ends.

4282 **2.15.7.6.3 Special Requirements**

4283 None.

4284 **2.15.7.6.4 Pre-Conditions**

4285 None.

4286 **2.15.7.6.5 Post-Conditions**

4287 None.

4288

4289 **2.15.7.7 Get Result**

4290 **2.15.7.7.1 Description**

4291 This use case calculates the Measurement logged.

4292 **2.15.7.7.2 Flow of Events**

4293 **2.15.7.7.2.1 Basic Flow**

4294 This use case starts when user wanted to get result base on the Reference ID.

4295 1. The Functional Element gets the Reference ID from user

4296 2. The Functional Element passes the Reference ID to Calculate Measurement Use Case.

4297 3. The Functional Element gets calculated result.

4298 4. The Functional Element returns the result to the user.

4299 **2.15.7.7.2.2 Alternative Flows**

4300 1. Log file not available, the Functional Element returns an error and the user case ends.

4301 2. If the Functional Element fails to get a reference ID, the Functional Element returns an error
4302 message and the use case ends.

4303 **2.15.7.7.3 Special Requirements**

4304 None.

4305 **2.15.7.7.4 Pre-Conditions**

4306 None.

4307 **2.15.7.7.5 Post-Conditions**

4308 None.

4309

4310 **2.15.7.8 Manage Security**

4311 **2.15.7.8.1 Description**

4312 This use case allows user to check that the known web service is securely managed.

4313 **2.15.7.8.2 Flow of Events**

4314 **2.15.7.8.2.1 Basic Flow**

- 4315 1. The service provider sends a request to check security of the known web service.
- 4316 2. User submits the WSDL of a known web service.
- 4317 3. Functional Element parses the URL of the WSDL document and extracts the necessary
4318 information.
- 4319 4. Functional Element generates client base on the extracted information.
- 4320 5. Functional Element invokes the known web service with a username.
- 4321 6. User sends a message to the known web service.
- 4322 7. The Functional Element checks whether username is authenticated.
- 4323 8. The Functional Element checks whether message is encrypted.
- 4324 9. The Functional Element checks whether the whole process is access controlled.
- 4325 10. The Functional Element returns the outcome to the user and the use case ends.

4326 **2.15.7.8.2.2 Alternative Flows**

- 4327 1. If the structure of the WSDL does not comply with the standard, the Functional Element
4328 returns an error message and the use case ends.
- 4329 2. If the Functional Element fails to generate the client, the Functional Element returns an error
4330 message and the use case ends.
- 4331 3. If the Functional Element fails to find the known web service, the Functional Element returns
4332 an error message and the use case ends.
- 4333 4. If the Functional Element fails to invoke the known web service, the Functional Element
4334 returns an error message and the use case ends.
- 4335 5. If the web service fails to return result, the Functional Element returns an error message and
4336 the use case ends.

4337 **2.15.7.8.3 Special Requirements**

4338 None.

4339 **2.15.7.8.4 Pre-Conditions**

4340 None.

4341 **2.15.7.8.5 Post-Conditions**

4342 None.

4343 **2.16 Role and Access Management Functional Element**

4344 **2.16.1 Motivation**

4345 The Role and Access Management Functional Element is expected to be an integral part of the
4346 User Access Management (UAM) functionalities that is expected to be needed by a Web Service-
4347 enabled implementation. This Functional Element is expected to fulfill the needs arising out of
4348 managing access to resources within an application, based on role-based access control
4349 mechanism. As such it will cover aspects that include:

- 4350
- Management of roles and access privileges, and
 - Assignment of roles to entities that will be accessing the resources that is being managed.
- 4351
4352

4353

4354 This Functional Element fulfills the following requirements from the Functional Elements
4355 Requirements Document 02:

- 4356
- Primary Requirements
 - MANAGEMENT-030 to MANAGEMENT-034, and
 - MANAGEMENT-200 to MANAGEMENT-205.
 - Secondary Requirements
 - SECURITY-040 to SECURITY-041.
- 4357
4358
4359
4360
4361

4362 **2.16.2 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> <p>Determine who (user) is requesting access.</p> <p>Determine the role(s) of the user</p> <p>Determine the type of access that is allowed based on the role(s) of the user</p> <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> <p>Defining a set of basic user information that should be stored in any enterprise application.</p> <p>Providing a means to extend this basic set of user information when needed..</p> <p>Simplifying management by grouping related users together through certain criteria.</p> <p>Having the flexibility of adopting both coarse/fine grain access controls.</p>

4363

4364 **2.16.3 Key Features**

4365 Implementations of the Secure SOAP Functional Element are expected to provide the following
4366 key features:

- 4367 1. The Functional Element MUST provide the capability to manage the creation and deletion of
4368 instances of the following concepts based on a pre-defined structure:
- 4369 1.1. Role,
 - 4370 1.2. Access, and
 - 4371 1.3. Resource
- 4372 16. The Functional Element MUST provide the capability to manage all the information (attribute
4373 values) stored in such concepts. This includes the capability to retrieve and update attribute's
4374 values belonging to a concept like Role, Access or Resource.
- 4375 17. The Functional Element MUST provide the capability to associate a Role to its access
4376 privileges through the Access structure.
- 4377 18. The Functional Element MUST provide the capability to determine a Role's accessibility to
4378 Resources based on the access privileges that have been assigned.
- 4379 19. The Functional Element MUST provide the ability to manage the association of users to
4380 Roles via assignments of Roles to users. This will include:
- 4381 1.4. Assignment/Un-assignment of Roles to individual Users, and
 - 4382 1.5. Assignment/Un-assignment of Roles to Groups.
- 4383 This will provide an indirect linkage between the accessibility of specific Users to Resources
4384 through the concept of Role and Access.

4385 20. The Functional Element MUST provide a mechanism for managing the concepts of Role,
4386 Access and Resource across different application domains.

4387 *Example: Namespace control mechanism*

4388

4389 In addition, the following key features could be provided to enhance the Functional Element
4390 further:

4391 1. The Functional Element MAY provide a mechanism to enable different Access instances to
4392 be related to one another.

4393 2. The Functional Element MAY also provide a mechanism to enable hierarchical
4394 relationships between Access instances.

4395 *Example: Parent and Child Relationship*

4396 3. The Functional Element MAY provide the ability for Roles to be temporal sensitive.

4397 *Example: A Role is assigned to a particular Phase in a Lifecycle.*

4398

4399 **2.16.4 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.

4400 **2.16.5 Related Technologies and Standards**

4401 None

4402

4403 **2.16.6 Model**

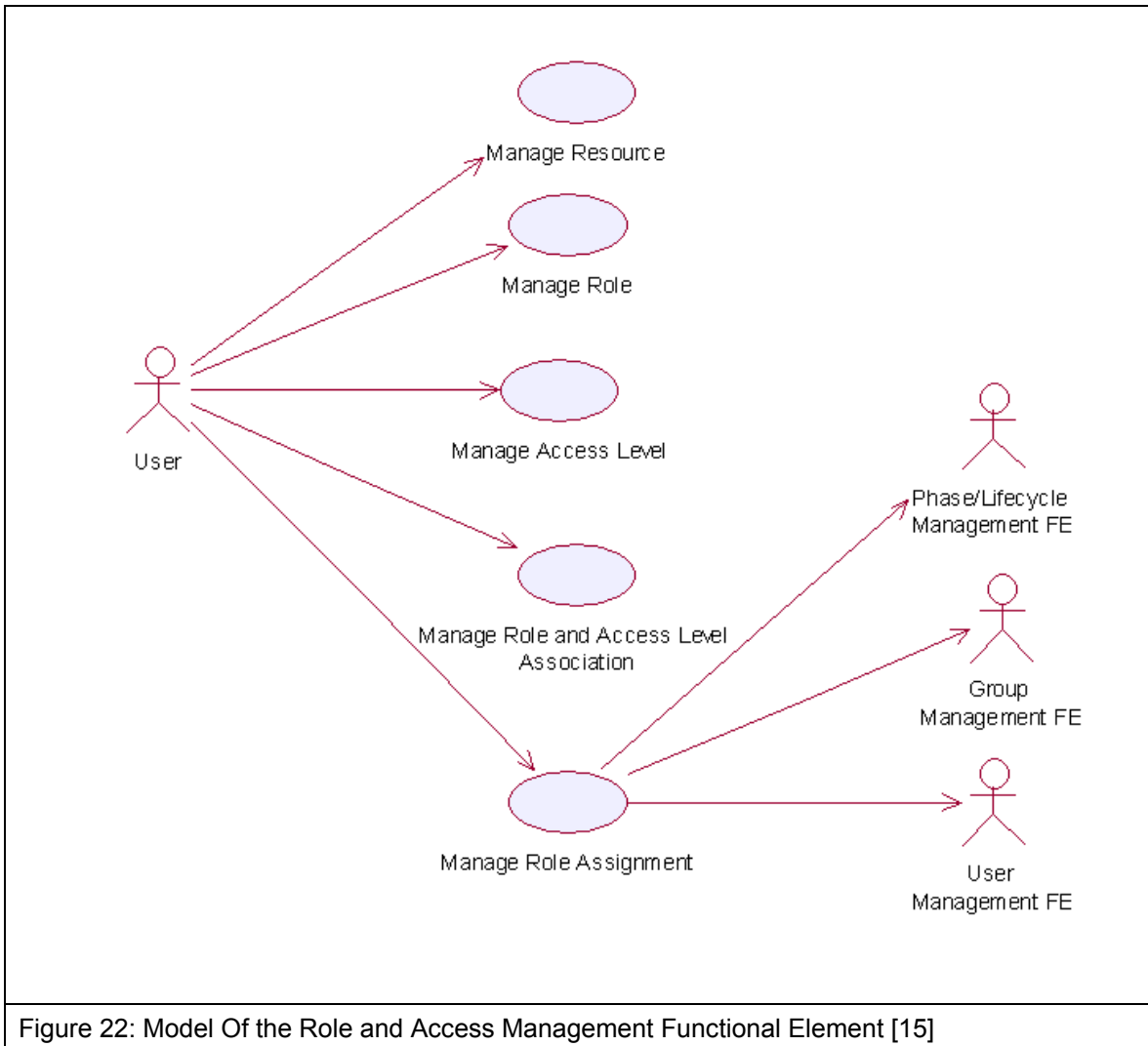


Figure 22: Model Of the Role and Access Management Functional Element [15]

4404

4405 **2.16.7 Usage Scenario**

4406 **2.16.7.1 Manage Role**

4407 **2.16.7.1.1 Description**

4408 This use case allows the service user to manipulate the role information such as adding,
4409 changing and deleting role information in the Functional Element.

4410 **2.16.7.1.2 Flow of Events**

4411 **2.16.7.1.2.1 Basic Flow**

4412 This use case starts when any user wants to create, change or delete a role.

- 4413 1: Service user specifies the function it would like to perform (either create a role, update a role or
4414 delete a role).
- 4415 2: Once the service user provides the requested information, one of the sub-flows is executed.
- 4416 If the service user provides '**Create a Role**', then sub-flow 2.1 is executed.
- 4417 If the service user provides '**Retrieve a Role**', then sub-flow 2.2 is executed.
- 4418 If the service user provides '**Update a Role**', then sub-flow 2.3 is executed.
- 4419 If the service user provides '**Delete a Role**', then sub-flow 2.4 is executed.
- 4420 2.1: Create a Role.
- 4421 2.1.1: The service user specifies role information such as the role name and description.
- 4422 2.1.2: The Functional Element connects to the data storage.
- 4423 2.1.3: The Functional Element checks whether the role exists in the Functional Element
4424 or not, saves the role information in the data storage and the use case ends.
- 4425 2.2: Retrieve a Role.
- 4426 2.2.1: The service user specifies the role name for retrieval.
- 4427 2.2.2: The Functional Element connects to the data storage.
- 4428 2.2.3: The Functional Element retrieves the role information in the data storage and the
4429 use case ends.
- 4430 2.3: Update a Role.
- 4431 2.3.1: The service user specifies the role name to update.
- 4432 2.3.2: The service user specifies the target field name and value of the role.
- 4433 2.3.3: The Functional Element connects to the data storage.
- 4434 2.3.4: The Functional Element updates the role information in the data storage and the
4435 use case ends.
- 4436 2.4: Delete a Role.
- 4437 2.4.1: The service user specifies the role name to delete.
- 4438 2.4.2: The Functional Element connects to the data storage.
- 4439 2.4.3: The Functional Element removes the record of the role in the data storage and the
4440 use case ends.

4441 **2.16.7.1.2.2 Alternative Flows**

- 4442 1: Data Storage Not Available.
- 4443 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
4444 available, an error message is returned and the use case ends.
- 4445 2: Role Already Exists.

4446 2.1: If in basic flow 2.1.3, the Functional Element checks that the role already exists in the
4447 data storage, an error message is returned and the use case ends.

4448 3: Role Does Not Exist.

4449 3.1: If in basic flow 2.2.3, 2.3.4 and 2.4.3, the Functional Element checks that the role does
4450 not exist in the data storage, an error message is returned and the use case ends.

4451 4: Role Cannot Be Deleted.

4452 4.1: If in basic flow 2.4.3, the other information associated with the role, such as any access
4453 level assigned, still exists, the role information may not be removed. An error message is
4454 returned and the use case ends.

4455 **2.16.7.1.3 Special Requirements**

4456 None

4457 **2.16.7.1.4 Pre-Conditions**

4458 None.

4459 **2.16.7.1.5 Post-Conditions**

4460 If the use case was successful, the role is saved/updated/removed in the Functional Element.
4461 Otherwise, the Functional Element state is unchanged.

4462 **2.16.7.2 Manage Resource**

4463 **2.16.7.2.1 Description**

4464 This use case allows the service user to manipulate the resource information such as adding,
4465 changing and deleting resource information in the Functional Element.

4466 **2.16.7.2.2 Flow of Events**

4467 **2.16.7.2.2.1 Basic Flow**

4468 This use case starts when any user wants to create, change or delete a resource.

4469 1: The user specifies the function it would like to perform.

4470 2: The user provides the requested information, one of the sub-flows is executed.

4471 If the user provides '**Create a Resource**', then sub-flow 2.1 is executed.

4472 If the user provides '**Retrieve a Resource**', then sub-flow 2.2 is executed.

4473 If the user provides '**Update a Resource**', then sub-flow 2.3 is executed.

4474 If the user provides '**Delete a Resource**', then sub-flow 2.4 is executed.

4475 2.1: Create a Resource.

4476 2.1.1: The user specifies resource information such as the resource name and
4477 description.

4478 2.1.2: The Functional Element connects to the data storage.

4479 2.1.3: The Functional Element checks whether the resource exists in the Functional
4480 Element, saves the resource information in the data storage and the use case ends.

4481 2.2: Retrieve a Resource.

4482 2.2.1: The service user specifies the resource name for retrieval.

4483 2.2.2: The Functional Element connects to the data storage.

4484 2.2.3: The Functional Element retrieves the resource information in the data storage and
4485 the use case ends.

4486 2.3: Update a Resource.

4487 2.3.1: The service user specifies the resource name to update.

4488 2.3.2: The Functional Element connects to the data storage.

4489 2.3.3: The Functional Element updates the resource information in the data storage and
4490 the use case ends.

4491 2.4: Delete a Resource.

4492 2.4.1: The service user specifies the resource name to delete.

4493 2.4.2: The Functional Element connects to the data storage.

4494 2.4.3: The Functional Element removes the record of the resource in the data storage
4495 and the use case ends.

4496 **2.16.7.2.2 Alternative Flows**

4497 1: Data Storage Not Available.

4498 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
4499 is not available, an error message is returned and the use case ends.

4500 2: Resource Already Exists.

4501 2.1: If in basic flow 2.1.3, the Functional Element checks that the resource already exists in
4502 the data storage, an error message is returned and the use case ends.

4503 3: Resource Does Not Exist.

4504 3.1: If in basic flow 2.2.3, 2.3.3 and 2.4.3, the Functional Element checks that the resource
4505 does not exist in the data storage, an error message is returned and the use case ends.

4506 **2.16.7.2.3 Special Requirements**

4507 None

4508 **2.16.7.2.4 Pre-Conditions**

4509 None.

4510 **2.16.7.2.5 Post-Conditions**

4511 None

4512 **2.16.7.3 Manage Access Level**

4513 **2.16.7.3.1 Description**

4514 This use case allows service user to manage the creation/retrieval/modification/deletion of access
4515 level.

4516 **2.16.7.3.2 Flow of Events**

4517 **2.16.7.3.2.1 Basic Flow**

4518 This use case starts when service user wants to manage the access levels.

4519 1: The service user specifies the function it would like to perform (add, update or delete an
4520 access level).

4521 2: Once the service user provides the requested information, one of the sub-flows is executed.

4522 If the service user provides '**Add an Access Level**', then sub-flow 2.1 is executed.

4523 If the service user provides '**Retrieve an Access Level**', then sub-flow 2.2 is activated.

4524 If the service user provides '**Update an Access Level**', then sub-flow 2.3 is activated.

4525 If the service user provides '**Delete an Access Level**', then sub-flow 2.4 is executed.

4526 2.1: Add an Access Level.

4527 2.1.1: The service user specifies the access level information, which includes: name,
4528 description, name of parent access level and group of resources that the access level is
4529 associated with.

4530 2.1.2: The Functional Element connects to the data storage.

4531 2.1.3: The Functional Element check whether the access level and its parent access level
4532 exist in the Functional Element, saves the access level information in the data storage
4533 and the use case ends.

4534 2.2: Retrieve an Access Level.

4535 2.2.1: The service user specifies the access level name to retrieve.

4536 2.2.2: The Functional Element connects to the data storage.

4537 2.2.3: The Functional Element gets access level information from the data storage and
4538 returns to the service user and the use case ends.

4539 2.3: Update an Access Level.

4540 2.3.1: The service user specifies the access level name.

4541 2.3.2: The service user specifies the field(s) and new value(s) to update.

4542 2.3.3: The Functional Element connects to the data storage.

4543 2.3.4: The Functional Element updates the access level information in the data storage
4544 with the value specified in 2.3.2 and the use case ends.

4545 2.4: Delete an Access Level.

4546 2.4.1: The service user specifies the access level name to delete.
4547 2.4.2: The Functional Element connects to the data storage.
4548 2.4.3: The Functional Element removes the record of the access level in the data storage
4549 and the use case ends.

4550 **2.16.7.3.2.2 Alternative Flows**

4551 1: Data Storage Not Available.

4552 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
4553 information is not available, an error message is returned and the use case ends.

4554 2: Access Level Already Exists.

4555 2.1: If in basic flow 2.1.3, the Functional Element checks that the access level already exists
4556 in the data storage, an error message is returned and the use case ends.

4557 3: Access Level Cannot Be Deleted.

4558 3.1: If in basic flow 2.4.3, the other information associated with the Access Level, such as
4559 roles to which the access level is assigned and the parent access level still exists, the access
4560 level information may not be removed. An error message is returned and the use case ends.

4561 4: Parent Access Level Not Exist.

4562 4.1: If in basic flow 2.1.3, the parent access level does not exist, an error message is returned
4563 and the use case ends.

4564 **2.16.7.3.3 Special Requirements**

4565 None

4566 **2.16.7.3.4 Pre-Conditions**

4567 None.

4568 **2.16.7.3.5 Post-Conditions**

4569 None

4570 **2.16.7.4 Manage Role and Access Level Association**

4571 **2.16.7.4.1 Description**

4572 This use case allows service user to assign, update and remove the access level assigned to
4573 role.

4574 **2.16.7.4.2 Flow of Events**

4575 **2.16.7.4.2.1 Basic Flow**

4576 This use case starts when service user wants to manage the relationship between access level
4577 and role.

4578 1: The service user specifies a role and the function he/she would like to perform on the role
4579 (either assign an access level to role, update role access level, or delete role access level).

4580 2: Once the service user provides the requested information, one of the sub-flows is executed.

4581 If the user provides '**Assign an Access Level to Role**', then sub-flow 2.1 is executed.

4582 If the user provides '**Update Access Level for Role**', then sub-flow 2.2 is executed.

4583 If the user provides '**Delete Access Level for Role**', then sub-flow 2.3 is executed.

4584 If the user provides '**Retrieve Access Level for Role**', then sub-flow 2.4 is executed.

4585 If the service user provides '**Retrieve Role for Access Level**', then sub-flow 2.5 is executed.

4586 2.1: Assign an Access Level to Role.

4587 2.1.1: The service user specifies access level that will be assigned to the role.

4588 2.1.2: The Functional Element connects to the data storage.

4589 2.1.3: The Functional Element checks whether the access level has been assigned to the
4590 role. Functional Element saves the access level reference in the role record in the data
4591 storage and the use case ends.

4592 2.2: Update Access Level for Role.

4593 2.2.1: The service user specifies the access level to update and the new access level
4594 information.

4595 2.2.2: The Functional Element connects to the data storage.

4596 2.2.3: The Functional Element updates the access level reference in the role record in the
4597 data storage and the use case ends.

4598 2.3: Delete Access Level to Role.

4599 2.3.1: The service user specifies the access level to delete.

4600 2.3.2: The Functional Element connects to the data storage.

4601 2.3.3: The Functional Element removes the access level reference from the record of the
4602 role in the data storage and the use case ends.

4603 2.4: Retrieve Access Level for Role.

4604 2.4.1: The service user specifies the role to retrieve the access levels associated with it.

4605 2.4.2: The Functional Element connects to the data storage.

4606 2.4.3: The Functional Element retrieves the access level assigned to the role in the data
4607 storage and the use case ends.

4608 2.5: Retrieve Role for Access Level.

4609 2.5.1: The service user specifies the access level to retrieve roles associated to it.

4610 2.5.2: The Functional Element connects to the data storage.

4611 2.5.3: The Functional Element retrieves roles associated to the access level in the data
4612 storage and the use case ends.

4613 **2.16.7.4.2.2 Alternative Flows**

4614 1: Data Storage Not Available.

4615 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
4616 not available, an error message is returned and the use case ends.

4617 2: Access Level Assignment Already Exists.

4618 2.1: If in basic flow 2.1.3, the Functional Element checks that the access level already exists
4619 in the role record in the data storage, an error message is returned and the use case ends.

4620 3: Access Level Assignment Not Exist.

4621 3.1: If in basic flow 2.3.3, the access level assignment does not exist, an error message is
4622 returned and the use case ends.

4623 4: Access Level Not Exist.

4624 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
4625 error message is returned and the use case ends.

4626 5: Role Not Exist.

4627 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
4628 message is returned and the use case ends.

4629 **2.16.7.4.3 Special Requirements**

4630 None.

4631 **2.16.7.4.4 Pre-Conditions**

4632 None.

4633 **2.16.7.4.5 Post-Conditions**

4634 None.

4635 **2.16.7.5 Manage Role Assignment**

4636 **2.16.7.5.1 Description**

4637 The use case allows service user to assign a role to a user, a group, a phase in a lifecycle, to
4638 change or to delete such assignment.

4639 **2.16.7.5.2 Flow of Events**

4640 **2.16.7.5.2.1 Basic Flow**

4641 This use case starts when the service user wants to manage the assignment of a role. This role
4642 can be assigned to a user, group, phase and lifecycle.

4643 1: Service user specifies a role and an operation to perform on the role.

4644 2: Once the service user provides the requested information, one of the sub-flows is executed.

4645 If the user provides '**Assign Role**', then sub-flow 2.1 is executed.

4646 If the user provides '**Retrieve Role**', then sub-flow 2.2 is executed.

4647 If the user provides '**Un-assign Role**', then user sub-flow 2.3 is executed.

- 4648 2.1: Assign Role.
- 4649 2.1.1: The service user specifies a user/group/phase/lifecycle to which the role will be
4650 assigned.
- 4651 2.1.2: Depending of target of the assignment, the Functional Element will check for the
4652 presence of one of the following Functional Elements.
- 4653 User Management Functional Element
- 4654 Group Management Functional Element
- 4655 Phase and Lifecycle Management Functional Element
- 4656 2.1.3: The Functional Element checks whether the role has been assigned to the
4657 intended target
- 4658 2.1.4: The Functional Element saves the relationship between the role and the target and
4659 the use case ends.
- 4660 2.2: Retrieve Role.
- 4661 2.2.1: The service user specifies a user/group/phase/lifecycle to retrieve all roles
4662 assigned
- 4663 2.2.2: Depending of target of the assignment, the Functional Element will check for the
4664 presence of one of the following Functional Elements.
- 4665 User Management Functional Element
- 4666 Group Management Functional Element
- 4667 Phase and Lifecycle Management Functional Element
- 4668 2.2.3: The Functional Element gets the roles that are assigned to the target.
- 4669 2.2.4: The Functional Element returns the results to the service user and the use case
4670 ends.
- 4671 2.3: Un-assign Role.
- 4672 2.3.1: The service user specifies a user/group/phase/lifecycle and the role that is to be
4673 un-assigned.
- 4674 2.3.2: Depending of target of this un-assignment, the Functional Element will check for
4675 the presence of one of the following Functional Elements.
- 4676 User Management Functional Element
- 4677 Group Management Functional Element
- 4678 Phase and Lifecycle Management Functional Element
- 4679 2.3.3: The Functional Element checks if the roles have been assigned to the target in the
4680 first place.
- 4681 2.3.4: The Functional Element removes the role assigned and the use case ends.
- 4682 **2.16.7.5.2.2 Alternative Flows**
- 4683 1: Dependent Functional Element not available.

- 4684 1.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the dependent Functional Elements are not
4685 available, an error message is returned and the use case ends.
- 4686 2: Invalid User/Group/Phase/Lifecycle Account.
- 4687 2.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the dependent Functional Elements are available
4688 but an invalid account is provided, an error message is returned and the use case ends.
- 4689 3: Data Storage Not Available.
- 4690 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
4691 storage, an error message is provided and the use case ends.
- 4692
- 4693 **2.16.7.5.3 Special Requirements**
- 4694 None.
- 4695 **2.16.7.5.4 Pre-Conditions**
- 4696 None.
- 4697 **2.16.7.5.5 Post-Conditions**
- 4698 None.

4699 **2.17 Search Functional Element**

4700 **2.17.1 Motivation**

4701 In a Web Service-enabled implementation, information is distributed across different sites and this
4702 makes searching and collating information difficult. Against this backdrop, this Functional
4703 Element is expected to fulfill the needs identified within an application by covering the following
4704 aspects:

- 4705 • Providing the capability for configuration of different types of data sources for information
4706 search,
- 4707 • Providing the facility to provide a concrete definition of data source classification for
4708 information search,
- 4709 • Providing the ability to define different search scopes for various data source
4710 classification,
- 4711 • Performing information search on those pre-configured different types of data sources
4712 and
- 4713 • Providing the provision to consolidate the return result arising from the search operation.
4714

4715 This Functional Element fulfills the following requirements from the Functional Elements
4716 Requirements Document 02:

- 4717 • Primary Requirements
 - 4718 ○ MANAGEMENT-009,
 - 4719 ○ PROCESS-030 to PROCESS-031, and
 - 4720 ○ PROCESS-034.
- 4721 • Secondary Requirements
 - 4722 ○ None

4723

4724 **2.17.2 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

4725 **2.17.3 Key Features**

4726 Implementations of the Search Functional Element are expected to provide the following key
4727 features:

- 4728 1. The Functional Element MUST provide a mechanism to define and manage Search
4729 Categories.
- 4730 2. The Functional Element MUST provide the capability to configure and store information
4731 about targeted data sources for a particular Search Category.
4732 *Example: Some of the stored information would include Location, Type, Name, Data Fields*
4733 *(of interest to the search) and access control (typically username and password) of the*
4734 *targeted data source.*
- 4735 3. As part of Key Feature (2), the Functional Element MUST also provide the ability to
4736 configure the scope of search and returned results.
- 4737 4. The Functional Element MUST also provide a mechanism to link the Search Categories to
4738 configured target data sources.
- 4739 5. The Functional Element MUST provide the ability to search multiple data sources for a
4740 defined Search Category.
4741 *Example: Some of the common data sources would include RDBMS, XML DB, LDAP*
4742 *servers and flat files like XML files, text files and HTML files*
- 4743 21. The Functional Element MUST provide the ability to perform searches based on a given set
4744 of keyword(s).

4745

4746 In addition, the following key features could be provided to enhance the Functional Element
4747 further:

- 4748 1. The Functional Element MAY also provide the ability to perform conditional and parametric
4749 searches.
- 4750 22. The Functional Element MAY also provide the ability to restrict the scope of a search.
4751 *Example: By providing a particular Search Category or types of data sources for the*
4752 *search.*

4753

4754 **2.17.4 Interdependencies**

4755 None

4756

4757 **2.17.5 Related Technologies and Standards**

4758 None

4759

2.17.6 Model

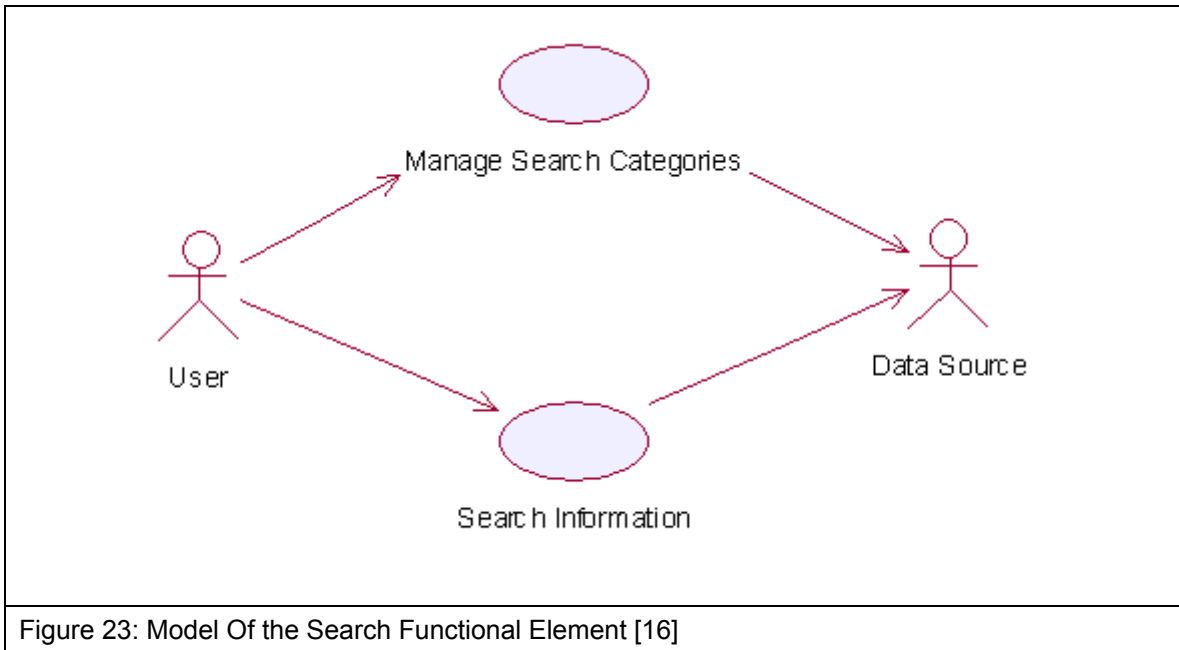


Figure 23: Model Of the Search Functional Element [16]

4760

4761 2.17.7 Usage Scenario

4762 2.17.7.1 Manage Search Categories

4763 2.17.7.1.1 Description

4764 This use case allows the users to manage the different search categories.

4765 2.17.7.1.2 Flow of Events

4766 2.17.7.1.2.1 Basic Flow

4767 This use case starts when the user wishes to manage the different data sources for search to be
4768 performed on it.

4769 1: The users initiates a request to configure data source(s) and type(s) by providing the data
4770 source information and type to be added, removed or retrieved.

4771 2: The Functional Element checks whether the data source configuration file exists.

4772 3: The Functional Element checks the request. Based on the type of request, one of the sub-
4773 flows is executed.

4774 If the request is to '**Create Data Source And Type**', then sub-flow 3.1 is executed.

4775 If the request is to '**View Data Sources And Types**', then sub-flow 3.2 is executed.

4776 If the request is to '**Delete Data Source And Type**', then sub-flow 3.3 is executed.

4777 3.1: Create Data Source and Type.

- 4778 3.1.1: The Functional Element checks whether the same data source and type has been
4779 created.
- 4780 3.1.2: The Functional Element appends the new data source and type in the data source
4781 configuration file specified.
- 4782 3.2: View Data Source and Type.
- 4783 3.2.1: The Functional Element retrieves all the data source and type information from the
4784 data source configuration file.
- 4785 3.2.2: The Functional Element returns the data source(s) and type(s).
- 4786 3.3: Delete Data Source and Type.
- 4787 3.3.1: The Functional Element checks whether the data source and type exist in the data
4788 source configuration based on data source id from the data source configuration file.
- 4789 3.3.2: The Functional Element removes the old data source and type from the data
4790 source configuration file.
- 4791 4: The Functional Element returns a success or failure flag indicating the status of the operation
4792 being performed and use case ends.
- 4793 **2.17.7.1.2.2 Alternative Flows**
- 4794 1: Data Source Configuration File Not Found.
- 4795 1.1: If in basic flow 2, the data source configuration file does not exist, the Functional Element
4796 creates an empty data source configuration file.
- 4797 2: Duplicate Data Source and Type.
- 4798 2.1: If in basic flow 3.1.1, the same data source and type have been configured, the
4799 Functional Element returns an error message and the use case end.
- 4800 3: Data Source and Type Do Not Exist.
- 4801 3.1: If in basic flow 3.2.1 and 3.3.1, a particular data source and type cannot be found in the
4802 specified data source configuration file, the Functional Element returns an error message and
4803 the use case end.
- 4804 **2.17.7.1.3 Special Requirements**
- 4805 None.
- 4806 **2.17.7.1.4 Pre-Conditions**
- 4807 None.
- 4808 **2.17.7.1.5 Post-Conditions**
- 4809 None.

4810 **2.17.7.2 Search Information**

4811 **2.17.7.2.1 Description**

4812 This use case allows any users to perform search on various disparate data sources and types
4813 configured to be searched and returns the matching results.

4814 **2.17.7.2.2 Flow of Events**

4815 **2.17.7.2.2.1 Basic Flow**

4816 This use case starts when users wishes to perform information search on a data source.

4817 1: Users initiates a request to perform information search on a given data source by providing
4818 information to be searched, location of the data source(s) and the data source type(s).

4819 2: The Functional Element checks for the existence of the specified data source(s).

4820 3: The Functional Element validates the data source type(s) against the set of supported data
4821 type(s) configured within the Functional Element that are available for information search.

4822 4: The Functional Element performs information search based on the search parameters given by
4823 the users or the other Functional Elements.

4824 5: The Functional Element returns the result of the information search performed to the users or
4825 other Functional Elements and use case ends.

4826 **2.17.7.2.2.2 Alternative Flows**

4827 1: Data Source(s) Are Not Available.

4828 1.1: In basic flow 2, if the identified data source is not available, the Functional Element
4829 returns an error message and the use case ends.

4830 2: Invalid Configuration Instructions

4831 2.1: In basic flow 2, if the input inform by the user is incomplete, the Functional Element
4832 returns an error message and the use case ends.

4833 3: Invalid Data Source Type.

4834 3.1: In basic flow 3, if the data source type is invalid, the Functional Element returns an error
4835 message and the use case ends.

4836 4: No Matching Result.

4837 4.1: In basic flow 4, if the search results in no matching results, the Functional Element
4838 returns an error message and the use case ends..

4839 **2.17.7.2.3 Special Requirements**

4840 None

4841 **2.17.7.2.4 Pre-Conditions**

4842 None.

4843 **2.17.7.2.5 Post-Conditions**

4844 None.

4845

4846 **2.18 Secure SOAP Management Functional Element**

4847 **2.18.1 Motivation**

4848 In a Web Services implementation, it is envisaged that confidential information is being exchanged
4849 all the time. Against this backdrop, it is imperative that an application in such an environment is
4850 equipped with the capability to guard sensitive information from prying eyes. Secure SOAP
4851 Management fulfills this need by covering the following areas.

- 4852 • The facility of digitally signing SOAP message,
- 4853 • The facility of encrypting SOAP message, and
- 4854 • The capability to generate the original SOAP message after signing or encrypting the
4855 message.

4856

4857 This Functional Element fulfills the following requirements from the Functional Elements
4858 Requirements Document 02:

- 4859 • Primary Requirements
 - 4860 ○ SECURITY-003 (SECURITY-003-3 only),
 - 4861 ○ SECURITY-020 (all), and
 - 4862 ○ SECURITY-022, and
 - 4863 ○ SECURITY-026.
- 4864 • Secondary Requirements
 - 4865 ○ None

4866

4867 **2.18.2 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.

4868

4869 **2.18.3 Key Features**

4870 Implementations of the Secure SOAP Functional Element are expected to provide the following
4871 key features:

- 4872 1. The Functional Element MUST provide the capability to digitally sign SOAP messages
4873 completely or partially using XML-Signature Syntax and Processing, W3C Recommendation
4874 12 February 2002.
- 4875 2. The Functional Element MUST provide the capability to validate a signed SOAP message.
- 4876 3. The Functional Element MUST provide the capability to encrypt SOAP messages
4877 completely or partially using XML-Encryption Syntax and Processing, W3C
4878 Recommendation 10 December 2002.
- 4879 4. The Functional Element MUST provide the capability to decrypt encrypted SOAP messages.
- 4880 23. The Functional Element MUST support PKCS12 compatible digital certificates.
- 4881 5. The Functional Element MUST be able to verify the validity and authenticity of digital
4882 certificates used.

4883

4884 In addition, the following key features could be provided to enhance the Functional Element
4885 further:

- 4886 1. The Functional Element MAY also support PKCS11 compatible tokens.
- 4887 2. The Functional Element MAY also provide log support as part of the audit trails for its
4888 transaction records.

4889

4890 **2.18.4 Interdependencies**

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

4891 **2.18.5 Related Technologies and Standards**

Standards / Specifications	Specific References
Public Key Infrastructure (PKI)	PKI is a system of digital certificates, Certificate Authorities, and other registration authorities that verify and authenticate the validity of each party involved in an Internet transaction In this Functional Element, the private key and public key are generated for the Functional Element to sign and encrypt SOAP messages. The Functional Element uses the session key to encrypt the SOAP message. The digital certificate is attached to the SOAP message after the Functional Element has signed the SOAP message.
XML-Signature Syntax and Processing, W3C Recommendation 12 th Feb 2002 [17]	This specification addresses authentication, non-repudiation and data-integrity issues. In addition, it also specifies the XML syntax and processing rules for creating and representing digital signatures. In this Functional Element, both the digital signature on the SOAP message and validation of the signed SOAP message is done based on this specification.

XML-Encryption Syntax and Processing, W3C Recommendation 10 th Dec 2002 [18]	This specification addresses data privacy by defining a process for encrypting data and representing the result in XML document. In this Functional Element, the encryption and decryption of SOAP messages are done based on this specification.
---	--

4892

4893 **2.18.6 Model**

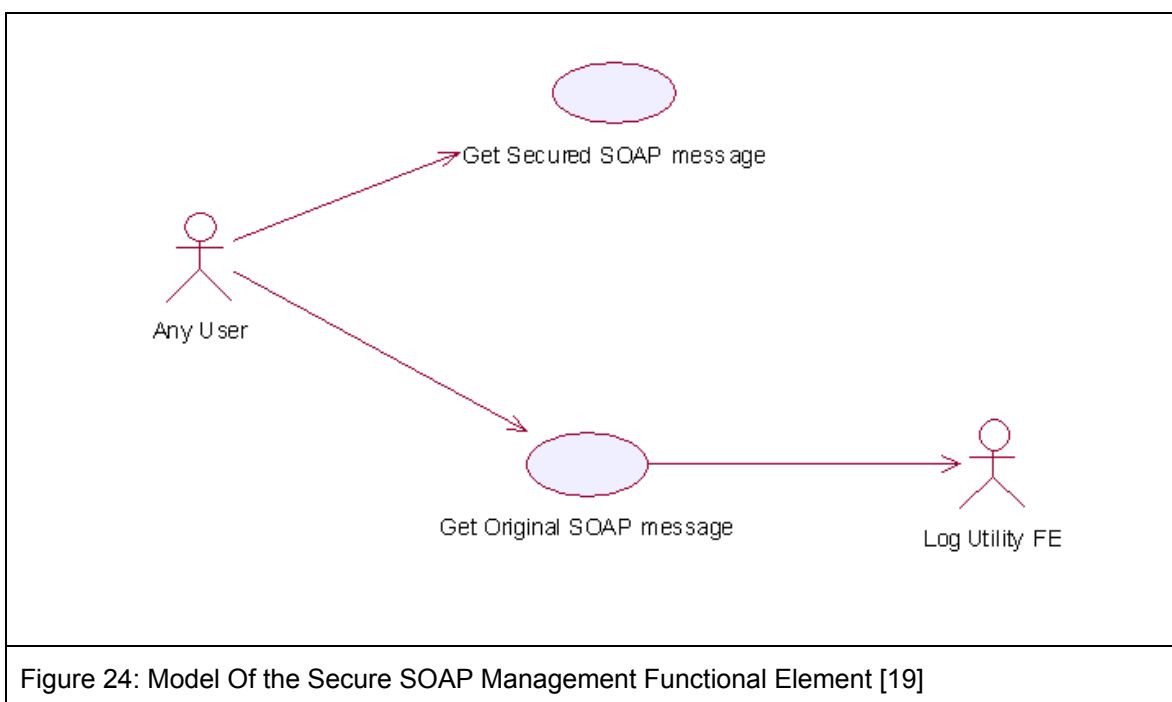


Figure 24: Model Of the Secure SOAP Management Functional Element [19]

4894 **2.18.7 Usage Scenarios**

4895 **2.18.7.1 Get Secured SOAP message**

4896 **2.18.7.1.1 Description**

4897 This Functional Element describes the process to generate secured SOAP message.

4898 **2.18.7.1.2 Flow of Events**

4899 **2.18.7.1.2.1 Basic Flow**

4900 This use case starts when the user wants to secure the SOAP message.

4901 If user wants to 'Sign SOAP message', then basic flow 1 is executed.

4902 If user wants to 'Encrypt and Sign the SOAP message', then basic flow 2 is executed.

4903 1: Sign SOAP Message.

4904 1.1: User sends the SOAP message, digital certificate and specifies the element name that
4905 needs to be signed.

4906 1.2: Functional Element gets the key information from the digital certificate.

4907 *Note: The private key will be used to sign the SOAP message and the public key will be*
4908 *added to the SOAP message after the signing.*

4909 1.3: Functional Element signs the element.

4910 *Note: The digital signature format is expected to be based on XML-Digital Signature Syntax*
4911 *mentioned in section 3.10.5.*

4912 1.4: Functional Element parses the secure SOAP message and regenerates the SOAP
4913 message.

4914 1.5: Functional Element returns the secured SOAP message to user and the use case ends.

4915 2: Encrypt And Sign SOAP Message.

4916 2.1: User sends the SOAP message, digital certificate and specify the element name that
4917 needs to be encrypted.

4918 2.2: User sends the receiver's public key information to Functional Element.

4919 *Note: Receiver's public key will be used to encrypt the session key, which was then used to*
4920 *encrypt the content of the element in the SOAP message.*

4921 2.3: Functional Element gets key information from the user's digital certificate.

4922 *Note: Private key is used to sign the SOAP message and public key is used to add into the*
4923 *SOAP message after the signing.*

4924 2.4: Functional Element generates the session key.

4925 *Note: Session key is used to encrypt the content of the element.*

4926 2.5: Functional Element encrypts the content of element with the session key.

4927 2.6: Functional Element encrypts session key with the receiver's public key.

4928 2.7: Functional Element signs the SOAP message after encryption.

4929 2.8: Functional Element regenerates the SOAP message.

4930 *Note: Functional Element adds the encrypted content of the element, encrypted session key*
4931 *information, the receiver's public key information and the signature to the SOAP message.*

4932 2.9: Functional Element returns the SOAP message and the use case ends.

4933 **2.18.7.1.2.2 Alternative Flows**

4934 1: Cannot Get Key.

4935 1.1: In basic flow 1.2 and 2.3, Functional Element cannot get the key information from the
4936 digital certificate. The Functional Element returns an error message and the use case ends.

4937 2: Cannot Sign

4938 2.1: In basic flow 1.3, Functional Element cannot sign the SOAP message. The Functional
4939 Element returns an error message and the use case ends.

4940 3: Cannot Encrypt

4941 3.1: In basic flow 2.5, Functional Element cannot encrypt the SOAP message. The Functional
4942 Element returns an error message and the use case ends.

4943 **2.18.7.1.3 Special Requirements**

4944 None.

4945 **2.18.7.1.4 Pre-Conditions**

4946 None.

4947 **2.18.7.1.5 Post-Conditions**

4948 None.

4949 **2.18.7.2 Get Original SOAP Message**

4950 **2.18.7.2.1 Description**

4951 This use case allows users to get original SOAP message.

4952 **2.18.7.2.2 Flow of Events**

4953 **2.18.7.2.2.1 Basic Flow**

4954 This use case starts when the user wants to get the original SOAP message.

4955 If the user wants to '**Verify the SOAP message**', then basic flow 1 is executed.

4956 If the user wants to '**Decrypt and Verify the SOAP message**', then basic flow 2 is executed.

4957 1: Verify SOAP Message.

4958 1.1: User sends the SOAP message and sender's digital certificate.

4959 1.2: Functional Element verifies the SOAP message.

4960 *Note: The sender's certificate information will be used to verify the signature.*

4961 1.3: Functional Element gets the original SOAP message, returns to user and the use case
4962 ends.

4963 2: Decrypt And Verify The SOAP Message.

4964 2.1: User sends the SOAP message, user's digital certificate and sender's certificate.

4965 2.2: Functional Element verifies the SOAP message.

4966 *Note: The sender's certificate information will be used to verify the signature.*

4967 2.3: Functional Element gets the user's key information from the user's digital certificate.

4968 *Note: The user's private key will be used to decrypt the session key.*

4969 2.4: Functional Element decrypts the session key.

4970 2.5: Functional Element decrypts the content of the element with the session key.

4971 2.6: Functional Element regenerates the SOAP message.

- 4972 *Note: Functional Element removes the session key information and the digital signature*
4973 *information from the SOAP message and gets the original one.*
- 4974 2.7: Functional Element returns the original SOAP message to user and the use case ends.
- 4975 **2.18.7.2.2 Alternative Flows**
- 4976 1: Verification Fails.
- 4977 1.1: In basic flow 1.3 and 2.3, if verification fails, the Functional Element returns an error
4978 message and the use case ends.
- 4979 2: Decryption of Content Fails.
- 4980 2.1: In basic flow 2.5, the Functional Element cannot decrypt the content of the element. The
4981 Functional Element returns an error message and the use case ends.
- 4982 **2.18.7.2.3 Special Requirements**
- 4983 None
- 4984 **2.18.7.2.4 Pre-Conditions**
- 4985 None.
- 4986 **2.18.7.2.5 Post-Conditions**
- 4987 None.

4988 **2.19 Sensory Functional Element**

4989 **2.19.1 Motivation**

4990 In a Web Service implementation where the presentation capabilities of clients differ, there is a
4991 need to determine the exact ability of the end devices so that the appropriate contents may be
4992 forwarded. The Sensory Functional Element can help to play this role by covering the following
4993 aspects within an application:

- 4994 • Determining the presentation capabilities by inspecting incoming headers, and
- 4995 • Determining the presentation capabilities by extracting MIME information from the
4996 relevant headers.

4997
4998 This Functional Element fulfills the following requirements from the Functional Elements
4999 Requirements Document 02:

- 5000 • Primary Requirements
 - 5001 ○ DELIVERY-001,
 - 5002 ○ DELIVERY-005 to DELIVERY-006, and
 - 5003 ○ DELIVERY-009.
- 5004 • Secondary Requirements
 - 5005 ○ MANAGEMENT-011, and
 - 5006 ○ MANAGEMENT-096.

5007

5008 **2.19.2 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.

5009

5010 **2.19.3 Key Features**

5011 Implementations of the Sensory Functional Element are expected to provide the following key
5012 features:

- 5013 1. The Functional Element MUST intercept HTTP requests from client and determines existing
5014 supportability of the request's MIME type.

5015 24. The Functional Element MUST provide the mechanism to manage MIME types, including the
5016 ability to add, delete and retrieve supported MIME types.

5017

5018 In addition, the following key features could be provided to enhance the Functional Element
5019 further:

5020 1. The Functional Element MAY provide a mechanism to enable Location Based Services
5021 (LBS).

5022 2.19.4 Interdependencies

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

5023 2.19.5 Related Technologies and Standards

5024 None.

5025

5026 2.19.6 Model

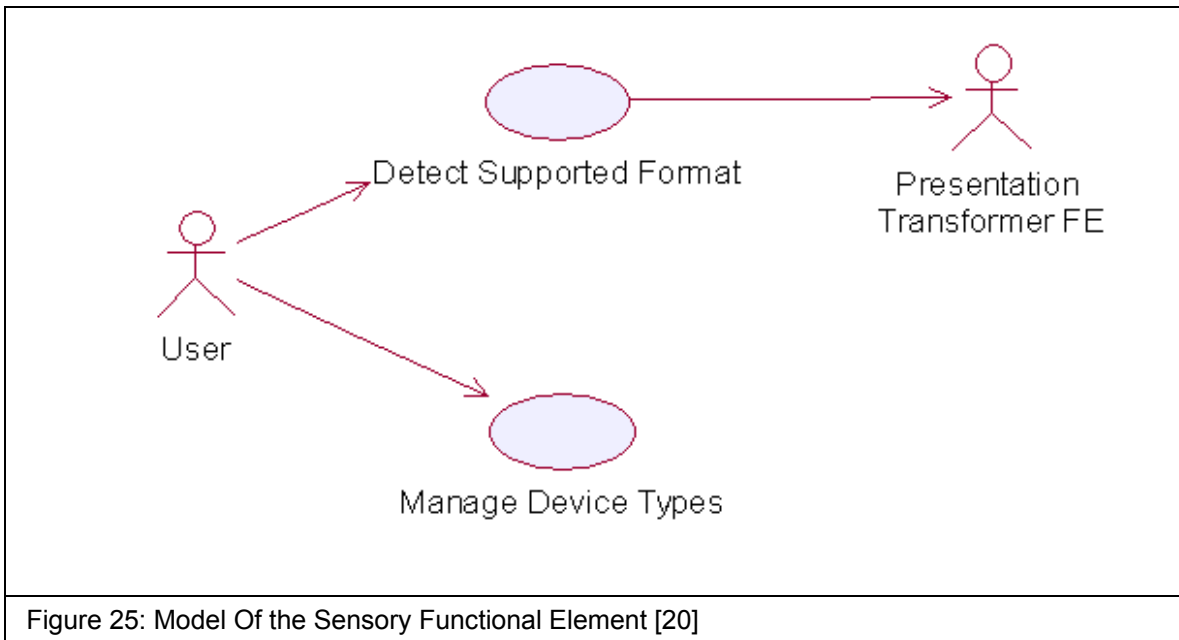


Figure 25: Model Of the Sensory Functional Element [20]

5027 2.19.7 Usage Scenarios

5028 2.19.7.1 Detect Supported Format

5029 2.19.7.1.1 Description

5030 This use case allows the service user (user/other service) to make request and based on that
5031 request it detects service user's device capabilities.

5032 **2.19.7.1.2 Flow of Events**

5033 **2.19.7.1.2.1 Basic Flow**

5034 This use case starts when the service user wishes to use any service provided by the service
5035 provider.

5036 1: The Functional Element receives the request from the service user.

5037 2: The Functional Element extracts MIME name and MIME type from the service user's HTTP
5038 request (even from SOAP request).

5039 3: The Functional Element uses MIME name and MIME TYPE to check with the pre-registered
5040 MIME type.

5041 4: The Functional Element sends device capabilities to service user and ends the use case.

5042 **2.19.7.1.2.2 Alternative Flows**

5043 1: Unsupported Device.

5044 1.1 If in the basic flow 2, the Functional Element is unable to detect the service user' device
5045 capability, the Functional Element returns a error message and the use case ends.

5046 **2.19.7.1.3 Special Requirements**

5047 None

5048 **2.19.7.1.3.1 Supportability**

5049 The edge devices must be able to support the HTTP request.

5050 **2.19.7.1.4 Pre-Conditions**

5051 None.

5052 **2.19.7.1.5 Post-Conditions**

5053 None.

5054 **2.19.7.2 Manage Device Types**

5055 **2.19.7.2.1 Description**

5056 This use case allows the service user to maintain the device (MIME Type information). This
5057 includes adding, changing and deleting device information from the Functional Element.

5058 **2.19.7.2.2 Flow of Events**

5059 **2.19.7.2.2.1 Basic Flow**

5060 This use case starts when the service user wishes to add or delete either device or service
5061 information from the Functional Element.

5062 1: The Functional Element requests that the service user specify the function to perform (either
5063 add, update or delete device or service).

5064 2: Once the service user provides the requested information, one of the sub-flows is executed.
5065 If the service user provides 'Register Device Types', then sub-flow 2.1 is executed.
5066 If the service user provides 'Delete Device Types', then sub-flow 2.2 is executed.
5067 2.1: Register Device Type.
5068 2.1.1: The Functional Element requests that the service user provide the device
5069 information. This includes: MIME Name, MIME Description, Supported MIME type.
5070 2.1.2: Once the service user provides the requested information, the Functional Element
5071 generates and assigns a unique MIME Id number to the device.
5072 2.2: Delete Device Type.
5073 2.2.1: The Functional Element requests that the service user provide the Device ID.
5074 2.2.2: The Functional Element retrieves the existing device information based on the
5075 Device ID.
5076 2.2.3: The service user provides the delete device information and the Functional
5077 Element deletes the device record from the Functional Element.
5078 3: The use case ends when the service user provides the requested information or decided to
5079 end use case.

5080 **2.19.7.2.2 Alternative Flows**

5081 1: Invalid Device Information.
5082 1.1: If in the sub-flow 2.1.2, the requested information provided by the user is invalid, the
5083 Functional Element returns an error message and the use case ends
5084 2: Device Not Found.
5085 2.1 If in the basic flows 2.2.2, the device information with the specified device is not found or
5086 does not exist, the Functional Element returns an error message and the use case ends.

5087 **2.19.7.2.3 Special Requirements**

5088 **2.19.7.2.3.1 Supportability**

5089 Manage Device Types supports the most widespread MIME types used today.

5090 **2.19.7.2.4 Pre-Conditions**

5091 None.

5092 **2.19.7.2.5 Post-Conditions**

5093 If the use case was successful, the device information is added, updated or deleted from the
5094 Functional Element. Otherwise, the Functional Element's state is unchanged.

5095 **2.20 Service Level Management Functional Element (new)**

5096 **2.20.1 Motivation**

5097 The Service Level Management Functional Element enables the management of Service Level
5098 Agreements (SLAs), each of which represents a joint agreement between the service customer
5099 and provider based on a set of service offerings. The service offerings typically expressed as
5100 SLA templates, but still can be customized to cater to various services and customers. The
5101 Service Level Management Functional Element also manages the lifecycle of a SLA which could
5102 be broadly classified into: SLA creation; SLA deployment and provisioning; SLA enforcement and
5103 SLA termination. The last two aspects are covered under Service Level Enforcement Functional
5104 Element.

5105

5106 This Functional Element fulfills the following requirements from the Functional Elements
5107 Requirements Document 02:

- 5108 • Primary Requirements
 - 5109 ○ MANAGEMENT-300.
- 5110 • Secondary Requirements
 - 5111 ○ None.

5112

5113 **2.20.2 Terms Used**

Terms	Description
SLA	Service Level Agreement is a joint agreement between service provider and service customer to define a set of service offerings.

5114

5115 **2.20.3 Key Features**

5116 Implementations of the Service Level Management Functional Element are expected to provide
5117 the following key features:

- 5118 1. The Functional Element MUST provide the ability to create Service Offering and associated
5119 service levels.
- 5120 2. The Functional Element MUST provide the ability to manage defined Service Offerings,
5121 including the ability to retrieve, modify and delete.
- 5122 3. The Functional Element MUST provide the ability to create of a SLA via customer
5123 subscription based on defined Service Offerings.
- 5124 4. The Functional Element MUST provide the ability to generate billing & service level reports
5125 based on defined SLAs.
- 5126 5. The Functional Element MUST provide the ability to notify subscribers of SLA termination.
- 5127 6. The Functional Element MUST provide the ability to delete SLAs upon termination.

5128

5129 In addition, the following key features could be provided to enhance the Functional Element
5130 further:

- 5131 1. The Functional Element MAY provide the ability to customize SLAs. This includes the
5132 capability to:

- 5133 1.1. Alter service offerings parameters.
- 5134 1.2. Add and delete different service offerings into a SLA.
- 5135

5136 **2.20.4 Interdependencies**

Interaction Dependencies	
QoS Management	The Service Level Management Functional Element may make use of the metrics and metering results to model SLAs.
Notification	The Service Level Management Functional Element may make use of the Notification Functional Element to notify subscribers of certain SLAs the happening on the SLAs.

5137

5138 **2.20.5 Related Technologies and Standards**

Standards / Specifications	Specific References
Web Service Level Agreement Project	– Under IBM Emerging Technology Toolkit. Latest update was in 2003. No news on its standardization.

5139

5140 **2.20.6 Model**

5141

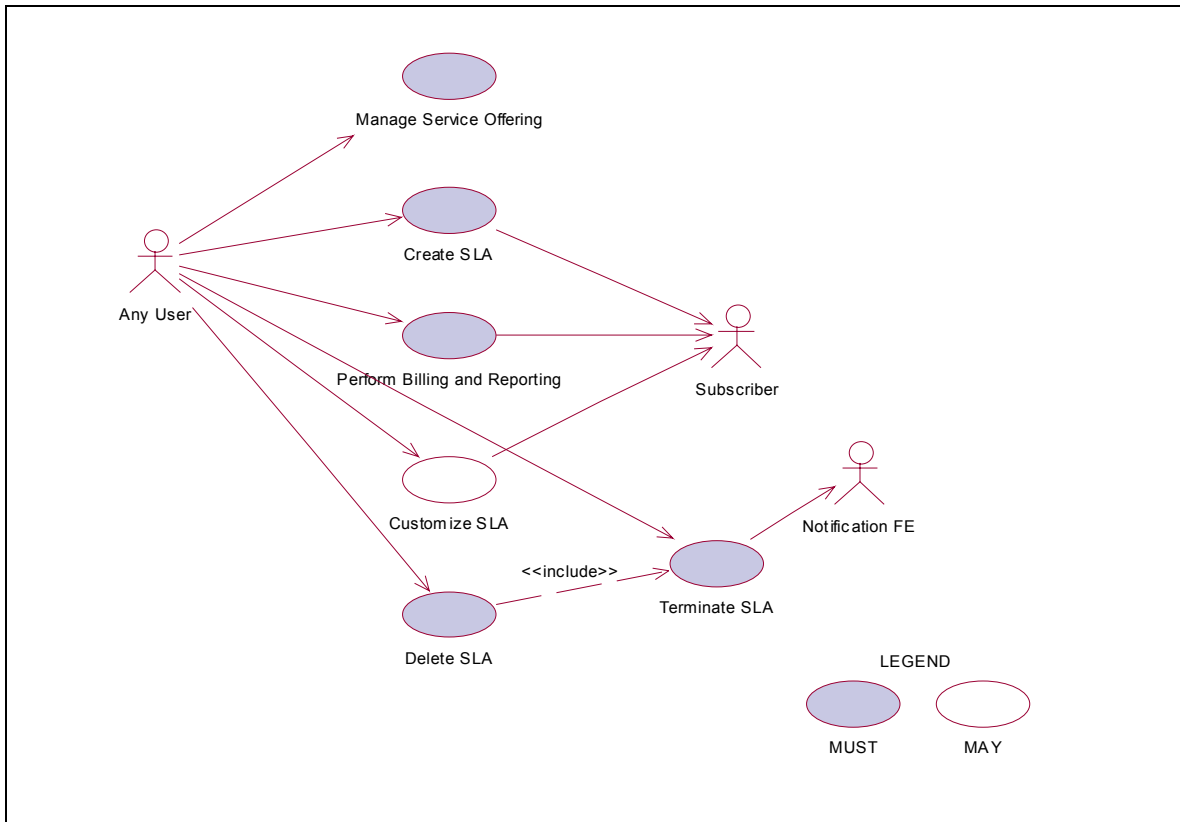


Figure 26: Model of the Service Level Management Functional Element.

5142

5143 2.20.7 Usage Scenarios

5144 2.20.7.1 Manage Service Offering

5145 2.20.7.1.1 Description

5146 This use case allows any user to manage service offering, which enables any user to create,
5147 retrieve, update and delete a service offering.

5148 2.20.7.1.2 Flow of Events

5149 2.20.7.1.2.1 Basic Flow

5150 This use case starts when any user wants to manage service offerings.

5151 1: The user sends Manage Service Offering request to the system together with the specified
5152 operation.

5153 2: On receipt of the request from the user, the functional element will execute one of the sub-
5154 flows. If the service user provides “**Create Service Offering**”, the Create Service Offering sub-
5155 flow (2.1) is executed. If the service user provides “**Update Service Offering**”, the Update
5156 Service Offering sub-flow (2.2) is activated. If the service user provides “**Retrieve Service**
5157 **Offering**”, the Retrieve Service Offering sub-flow (2.3) is activated. If the service user provides
5158 “**Delete Service Offering**”, the Delete Service Offering sub-flow (2.4) is executed.

5159
5160 2.1: Create Service Offering.
5161 2.1.1: The service user specifies details of a service offering.
5162 2.1.2: The system checks the existing service offering.
5163 *2.1.3: The system generates service offering information and adds to the system*
5164 *and the use case ends.*
5165 2.2: Update Service Offering.
5166 2.2.1: The service user specifies the service offering to update.
5167 2.2.2: The system retrieves the existing service offering information.
5168 2.2.3: The service user provides the update service offering information.
5169 *2.2.4: The system updates the service offering with the updated information and*
5170 *ends use case.*
5171 2.3: Retrieve Service Offering.
5172 2.3.1: The service user specifies the service offering to retrieve.
5173 2.3.2: The system retrieves the existing service offering information and ends the use
5174 case.
5175 2.4: Delete Service Offering.
5176 2.4.1: The service user specifies the service offering to delete.
5177 2.4.2: The system retrieves the existing service offering information.
5178 *2.4.3: The system deletes the service offering from the system and the use case*
5179 *ends.*

5180 **2.20.7.1.2.2 Alternative Flows**

5181 1: Invalid Service Offering.
5182 1.1: If in the Basic Flow 2.1.1, system detects any invalid description, system returns
5183 general error message and ends the use case.
5184 2: Service Offering Already Exists.
5185 2.1: If in the Basic Flow 2.1.2, the system checks the existing service offering and finds the
5186 service offering already exists. The system returns an error and ends the use case.
5187 3: Service Offering Not Exist.
5188 3.1: If in the Basic Flow 2.2.2, 2.3.2, 2.4.2, the system checks the existing service
5189 offering and finds the service offering doesn't exist. The system returns an error
5190 and ends the use case.

5191 **2.20.7.1.3 Special Requirements**

5192 **2.20.7.1.4 Pre-Conditions**

5193 None.

5194 **2.20.7.1.5 Post-Conditions**

5195 None.

5196

- 5197 **2.20.7.2 Create SLA**
- 5198 **2.20.7.2.1 Description**
- 5199 This use case allows any user to create Service Level Agreement.
- 5200 **2.20.7.2.2 Flow of Events**
- 5201 **2.20.7.2.2.1 Basic Flow**
- 5202 This use case starts when any user wants to create SLA.
- 5203 1: The user sends a request to create SLA to the Functional Element which includes the
5204 arrangement of the defined service offerings.
- 5205 2: The Functional Element will dispatch the SLA information to the subscribers.
- 5206 3: The subscribers accept the SLA arrangement and the use case ends.
- 5207 **2.20.7.2.3 Alternative Flows**
- 5208 1: Service Offering Not Available.
- 5209 1.1: If in the Basic Flow 1, Functional Element detects the service offering provided by the
5210 user is not available, the Functional Element returns general error message and ends the use
5211 case.
- 5212 2: Subscriber Not Available.
- 5213 2.1: If in the Basic Flow 2, the Functional Element checks that the subscriber is not available,
5214 the Functional Element returns an error and ends the use case.
- 5215 3: Subscriber Don't Agree.
- 5216 3.1: If in the Basic Flow 3, the subscriber does not agree with the arrangement defined in
5217 SLA, the Functional Element returns an error and ends the use case.
- 5218 **2.20.7.2.4 Special Requirements**
- 5219 None.
- 5220 **2.20.7.2.5 Pre-Conditions**
- 5221 None.
- 5222 **2.20.7.2.6 Post-Conditions**
- 5223 If the use case is successful, a SLA is added into the Functional Element.
- 5224
- 5225 **2.20.7.3 Perform Billing and Reporting**
- 5226 This use case allows any user to do billing and reporting of the information related to SLA.

5227 **2.20.7.3.1 Flow of Events**

5228 **2.20.7.3.1.1 Basic Flow**

5229 This use case starts when any user wants to do SLA related billing and report.

5230 1: The user sends a request to conduct billing and reporting by providing information, which
5231 enables to identify the SLA and its service offering and associated subscribers.

5232 2: On receipt of request of performing billing and reporting from the user, the Functional Element
5233 retrieves the billing and report information according to the definition of SLA and internally
5234 recorded information.

5235 3: The Functional Element passes the generated information to the subscribers.

5236 4: The Functional Element passes the response to the user and the use case ends.

5237 **2.20.7.3.1.2 Alternative Flows**

5238 1: Information Not Enough.

5239 1.1: If in the Basic Flow 1, Functional Element detects the information provided by the user is
5240 not enough to form identify the SLA and its associated service offerings and subscribers,
5241 Functional Element returns general error message and ends the use case.

5242 2: No Data Available.

5243 2.1: If in the Basic Flow 2, the Functional Element retrieves the recorded information and
5244 finds it is unavailable or incomplete, the Functional Element returns an error and ends the use
5245 case.

5246 3: Subscriber Not Available.

5247 3.1: If in the Basic Flow 3, the subscriber is not available, the Functional Element returns an
5248 error and ends the use case.

5249 **2.20.7.3.2 Special Requirements**

5250 None.

5251 **2.20.7.3.3 Pre-Conditions**

5252 None.

5253 **2.20.7.3.4 Post-Conditions**

5254 None.

5255

5256 **2.20.7.4 Customize SLA**

5257 **2.20.7.4.1 Description**

5258 This use case allows users to customize a SLA.

5259 **2.20.7.4.1.1 Basic Flow**

5260 This use case starts when any user wants to customize a SLA.

5261 1: The user sends request to customize a SLA by providing the information what will be
5262 customized in a SLA. There are two ways to customize a SLA, to modify the parameters of
5263 service offerings in a SLA and to add or delete service offerings in a SLA.

5264 2: On receipt of a customizing SLA request from the user, the Functional Element checks the
5265 validity of the customized SLA.

5266 3: The Functional Element passes the customized SLA to the subscribers.

5267 4: The subscribers accept the customized SLA.

5268 5: The Functional Element passes the response from the service to the user and the use case
5269 ends.

5270 **2.20.7.4.1.2 Alternative Flows**

5271 1: SLA Not Available.

5272 1.1: If in the Basic Flow 1, the SLA that the user wants to customize does not exist,
5273 Functional Element returns general error message and ends the use case.

5274 2: Information Not Valid.

5275 2.1: If in the Basic Flow 2, Functional Element detects the information provided by the user is
5276 not valid to form a SLA, Functional Element returns general error message and ends the use
5277 case.

5278 3: Subscriber Not Available.

5279 3.1: If in the Basic Flow 3, the subscriber is not available, Functional Element returns general
5280 error message and ends the use case.

5281 4: Subscriber Does Not Accept.

5282 4.1: If in the Basic Flow 4, the subscriber does not accept the customized SLA, Functional
5283 Element returns general error message and ends the use case.

5284 **2.20.7.4.2 Special Requirements**

5285 None.

5286 **2.20.7.4.3 Pre-Conditions**

5287 None.

5288 **2.20.7.4.4 Post-Conditions**

5289 If the use case is successful, a customized SLA is added into the functional element.
5290

5291 **2.20.7.5 Terminate SLA**

5292 This use case enables the user to terminate a SLA.

5293 **2.20.7.5.1 Flow of Events**

5294 **2.20.7.5.1.1 Basic Flow**

5295 This use case starts when the user wants to terminate a SLA.

5296 1: The user sends a request to terminate a SLA to the Functional Element by providing related
5297 information.

5298 2: On receipt of a terminating SLA request from the user, the Functional Element terminates the
5299 operations related to the SLA.

5300 3: The Functional Element notifies the subscribers about the termination of the SLA through
5301 Notification Functional Element.

5302 4: The Functional Element passes the response from the service to the user and the use case
5303 ends.

5304 **2.20.7.5.1.2 Alternative Flows**

5305 1: SLA Not Exist.

5306 1.1: If in the Basic Flow 2, Functional Element detects the SLA that the user wants to
5307 terminate does not exist, Functional Element returns general error message and ends the use
5308 case.

5309 2: Notification FE Not Available.

5310 2.1: If in Basic Flow 3, Functional Element detects the Notification Functional Element is not
5311 available, Functional Element returns general error message and ends the use case.

5312 **2.20.7.5.2 Special Requirements**

5313 None.

5314 **2.20.7.5.3 Pre-Conditions**

5315 None.

5316 **2.20.7.5.4 Post-Conditions**

5317 If the use case is successful, the Functional Element stops all the operations related to the SLA.
5318

5319 **2.20.7.6 Delete SLA**

5320 This use case enables the user to remove a SLA from the Functional Element.

5321 **2.20.7.6.1 Flow of Events**

5322 **2.20.7.6.1.1 Basic Flow**

5323 This use case starts when the user wants to delete a SLA from the Functional Element.

5324 1: The user sends a request to delete a SLA providing related information.

5325 2: On receipt of request of deleting SLA from the user, the Functional Element validates the
5326 provided information and invokes the use case Terminate SLA.

5327 3: The Functional Element deletes the SLA.

5328 4: The Functional Element passes the response from the service to the user and the use case
5329 ends.

5330 **2.20.7.6.1.2 Alternative Flows**

5331 1: SLA Does Not Exist.

5332 1.1: If in the Basic Flow 2, Functional Element detects the SLA that the user wants to delete
5333 does not exist, Functional Element returns general error message and ends the use case.

5334 2: Terminate SLA Error.

5335 2.1: If in the Basic Flow 2, use case Terminate SLA returns error, Functional Element returns
5336 general error message and ends the use case.

5337 **2.20.7.6.2 Special Requirements**

5338 None.

5339 **2.20.7.6.3 Pre-Conditions**

5340 None.

5341 **2.20.7.6.4 Post-Conditions**

5342 If the use case is successful, a SLA is deleted from the Functional Element.

5343 **2.21 Service Level Enforcement Functional Element (new)**

5344 **2.21.1 Motivation**

5345 The Service Level Enforcement Functional Element enables monitoring the compliance of SLA
5346 and enforcing SLA through load management.

5347

5348 This Functional Element fulfills the following requirements from the Functional Elements
5349 Requirements Document 02:

- 5350 • Primary Requirements
 - 5351 ○ MANAGEMENT-301 and
 - 5352 ○ MANAGEMENT-302.
- 5353 • Secondary Requirements
 - 5354 ○ None.

5355

5356 **2.21.2 Terms Used**

Terms	Description
SLA	Service Level Agreement is a joint agreement between service provider and service customer to define a set of service offerings.

5357

5358 **2.21.3 Key Features**

5359 Implementations of the Service Level Enforcement Functional Element are expected to provide
5360 the following key features:

- 5361 1. The Functional Element MUST provide the ability to monitor SLA compliance based on
5362 measured data.
- 5363 2. The Functional Element MUST provide the ability to detect any violation of SLA.
- 5364 3. The Functional Element MUST provide the ability to enforce a SLA via through load
5365 management.

5366

5367 In addition, the following key features could be provided to enhance the Functional Element
5368 further:

- 5369 1. The Functional Element MAY provide the ability to manage load. This include the capability
5370 to:
 - 5371 1.1. Control admission of service.
 - 5372 1.2. Prioritize requests.

5373

5374 **2.21.4 Interdependencies**

Interaction Dependencies

QoS Management	The Service Level Enforcement Functional Element may make use the metrics and metering results to monitor compliance of SLA.
----------------	--

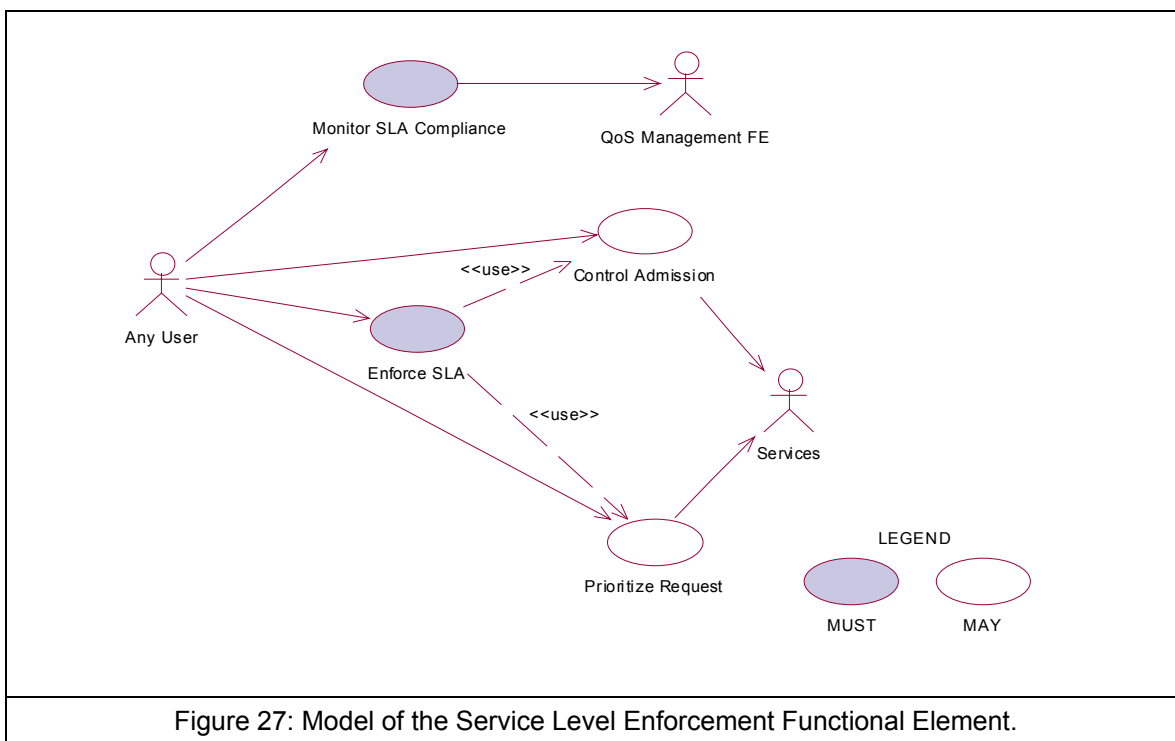
5375

5376 **2.21.5 Related Technologies and Standards**

Standards / Specifications	Specific References
Web Service Level Agreement Project	– Under IBM Emerging Technology Toolkit. Latest update was in 2003. No news on its standardization.

5377 **2.21.6 Model**

5378



5379

5380 **2.21.7 Usage Scenarios**

5381 **2.21.7.1 Monitor SLA Compliance**

5382 **2.21.7.1.1 Description**

5383 This use case allows any user to monitor and check the SLA is compliant or not at the run time.

5384 **2.21.7.1.2 Flow of Events**

5385 **2.21.7.1.2.1 Basic Flow**

5386 This use case starts when any user wants to monitor the SLA compliance.

5387 1: The user sends Monitor SLA Compliance request to the Functional Element together with the
5388 specified SLA information.

5389 2: On receipt of the request from the user, the Functional Element will retrieve the SLA
5390 information.

5391

5392 3: The Functional Element extracts the measured data through QoS Management Functional
5393 Element.

5394 4: The Functional Element checks the compliance of SLA.

5395 5: The Functional Element returns response to the user and the use case ends.

5396 **2.21.7.1.2.2 Alternative Flows**

5397 1: SLA Not Exist.

5398 1.1: If in the Basic Flow 2, the Functional Element detects that the SLA to monitor does not
5399 exist, system returns general error message and ends the use case.

5400 2: Measured Data Not Available.

5401 2.1: If in the Basic Flow 3, the Functional Element retrieves measured data through QoS
5402 Management Functional Element and the latter is not ready, the Functional Element returns
5403 an error and ends the use case.

5404 3: SLA Not Compliant.

5405 3.1: If in the Basic Flow 4, the Functional Element checks the measured data against SLA
5406 and the violation exists, the Functional Element returns an error and ends the use case.

5407 **2.21.7.1.3 Special Requirements**

5408 **2.21.7.1.4 Pre-Conditions**

5409 None

5410 **2.21.7.1.5 Post-Conditions**

5411 None

5412

5413 **2.21.7.2 Control Admission**

5414 **2.21.7.2.1 Description**

5415 As a means of manage load to enforce SLA, the use case allows any user to control admission
5416 toward services.

5417 **2.21.7.2.2 Flow of Events**

5418 **2.21.7.2.2.1 Basic Flow**

5419 This use case starts when any user wants to control admission toward services.

5420 1: The user sends request to control admission to certain services to the Functional Element
5421 which includes the option of admission and the targeted services.

5422 2: The Functional Element will manage the control of admission to the services at run time.

5423 3: The Functional Element returns response to the user and the use case ends.

5424 **2.21.7.2.3 Alternative Flows**

5425 1: Service Not Available.

5426 1.1: If in the Basic Flow 1, Functional Element detects the targeted service provided by the
5427 user is not available, Functional Element returns general error message and ends the use
5428 case.

5429 2: Control Admission Failed.

5430 2.1: If in the Basic Flow 2, the Functional Element fails to control admission to the services at
5431 run time, Functional Element returns an error and ends the use case.

5432 **2.21.7.2.4 Special Requirements**

5433 None.

5434 **2.21.7.2.5 Pre-Conditions**

5435 The services are manageable to the user.

5436 **2.21.7.2.6 Post-Conditions**

5437 If the use case is successful, the load of the monitored services is changed thus the SLA is
5438 enforced through load management.

5439

5440 **2.21.7.3 Prioritize Request**

5441 As a means of load management to enable SLA enforcement, the use case allows any user to
5442 prioritize request to the targeted services according to the requirements of SLA.

5443 **2.21.7.3.1 Flow of Events**

5444 **2.21.7.3.1.1 Basic Flow**

5445 This use case starts when any user wants to prioritize various requests to targeted services.

5446 1: The user sends request to prioritize request to the Functional Element, which include
5447 information of the targeted services, the priority of the request and so on.

5448 2: On receipt of the request from the user, the Functional Element controls the processing of the
5449 request according to the priority given at the run time.

5450 3: The Functional Element passes the response to the user and the use case ends.

5451 **2.21.7.3.1.2 Alternative Flows**

5452 1: Services Not Exist.

5453 1.1: If in the Basic Flow 1, Functional Element detects the targeted service provided by the
5454 user does not exist, Functional Element returns general error message and ends the use
5455 case.

5456 2: Prioritize Request Fails.

5457 2.1: If in the Basic Flow 2, the Functional Element fails to control the requests of the services
5458 according to the priority given the user, the Functional Element returns an error and ends the
5459 use case.

5460 **2.21.7.3.2 Special Requirements**

5461 None.

5462 **2.21.7.3.3 Pre-Conditions**

5463 The services are manageable to the user.

5464 **2.21.7.3.4 Post-Conditions**

5465 If the use case is successful, the load of the monitored services is changed thus the SLA is
5466 enforced through load management.

5467

5468 **2.21.7.4 Enforce SLA**

5469 **2.21.7.4.1 Description**

5470 This use case allows users to enforce a SLA in a run time environment.

5471 **2.21.7.4.1.1 Basic Flow**

5472 This use case starts when any user wants to enforce a SLA in the run time environment.

5473 1: The user sends a request to enforce a SLA to the Functional Element by providing the SLA
5474 and its associated services and the option of the means of enforcement through load
5475 management.

5476 2: On receipt of the request from the user, the Functional Element checks the SLA and decides
5477 the means of enforcement, i.e. by taking advantage of load management.

5478 3: The Functional Element dispatches its request of load management and invokes use case
5479 Control Admission or use case Prioritize Request.

5480 4: The Functional Element returns the response to the user and the use case ends.

5481 **2.21.7.4.1.2 Alternative Flows**

5482 1: SLA Not Available.

5483 1.1: If in the Basic Flow 1, the SLA that the user wants to enforce does not exist, Functional
5484 Element returns general error message and ends the use case.

5485 2: Services Not Exist.

5486 2.1: If in the Basic Flow 1, Functional Element detects the services that the user wants to
5487 enforce SLA do not exist, Functional Element returns general error message and ends the
5488 use case.

5489 3: Control Admission Not Working.

5490 3.1: If in the Basic Flow 3, Functional Element fails to invoke use case control admission,
5491 Functional Element returns general error message and ends the use case.

5492 4: Prioritize Request Not Working.

5493 4.1: If in the Basic Flow 3, Functional Element fails to invoke use case Prioritize Request,
5494 Functional Element returns general error message and ends the use case.

5495 **2.21.7.4.2 Special Requirements**

5496 None.

5497 **2.21.7.4.3 Pre-Conditions**

5498 The services targeted are manageable.

5499 **2.21.7.4.4 Post-Conditions**

5500 None.

5501

5502 **2.22 Service Management Functional Element**

5503 **2.22.1 Motivation**

5504 The ability to monitor Web Services invocation is crucial towards the adoption of this technology
5505 from the security and performance standpoints. A security framework should incorporate an
5506 authentication and authorisation mechanism together with an audit trail. These twin
5507 considerations will serve to discourage resource misuse and in addition, will help to promote the
5508 “pay-as-you-use” concept. Service throughput on the server end is another important parameter
5509 that must be monitored. Administrators of services, which are sluggish, should be notified
5510 immediately via any electronic means.

5511

5512 This Functional Element fulfills the following requirements from the Functional Elements
5513 Requirements Document 02:

- 5514 • Primary Requirements
 - 5515 ○ MANAGEMENT-090, and
 - 5516 ○ MANAGEMENT-093 to MANAGEMENT-096.
- 5517 • Secondary Requirements
 - 5518 ○ None

5519 **2.22.2 Terms Used**

Terms	Description
Management Domain	Management Domain refers to the set of servers that needs to be monitored. This domain is typically under the control of one agency and administered by a known administrator.
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

5520

5521 **2.22.3 Key Features**

5522 Implementations of the Service Management Functional Element are expected to provide the
5523 following key features:

- 5524 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)
- 5525 25. The Functional Element MUST provide the capability to discover services that are under the
5526 Management Domain.
- 5527 26. The Functional Element MUST provide the capability to configure Performance Parameters
5528 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.

5529 27. The Functional Element MUST provide a means to log Performance Parameters.

5530

5531 In addition, the following key feature could be provided to enhance the Functional Element
 5532 further:

5533 1. The Functional Element MAY provide the capability to configure additional attributes that is
 5534 tagged along with a particular Web Service.

Example: The access permission for invoking the service.

5535 2. The Functional Element MAY provide verification services to block unauthorized Web
 5536 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.

5537 **2.22.4 Interdependencies**

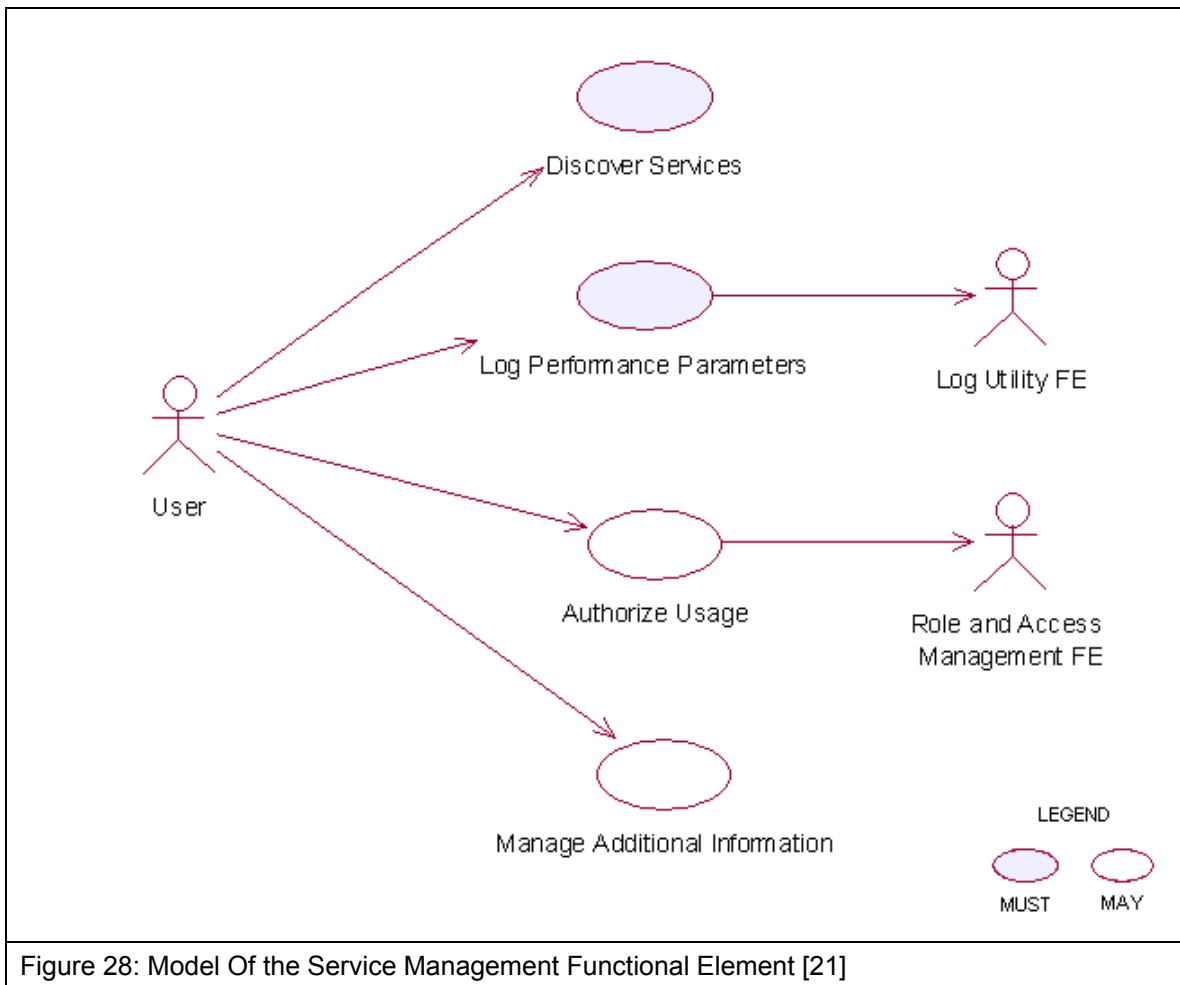
Direct Dependency	
Log Utility Functional Element	The Log Utility Functional Element helps to log the Performance Parameter into the appropriate data sources

5538

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.

5539 **2.22.5 Related Technologies and Standards**

5540 None

2.22.6 Model5542 **2.22.7 Usage Scenarios**5543 **2.22.7.1 Discover Services**5544 **2.22.7.1.1 Description**

5545 This use case describes the scenario surrounding the automatic discovery of services hosted in
 5546 the Management Domain.

5547 **2.22.7.1.2 Flow of Events**5548 **2.22.7.1.2.1 Basic Flow**

5549 The use case begins when the user wants to retrieve a list of services URLs from the
 5550 Management Domain.

5551 1: The user sends a request to retrieve the list of services URLs from the Management Domain.

5552 2: The Functional Element reads from a configuration file to so as to determine the exact
 5553 boundaries of the Management Domain.

5554 3: The Functional Element retrieves from each of the servers as stated in the configuration file a
5555 list of service URLs that it is hosting

5556 4: The Functional Element returns the list of service URLs back to the user and the use case
5557 ends.

5558 **2.22.7.1.2.2 Alternative Flows**

5559 1: Configuration File Does Not Exist

5560 1.1: In basic flow 2, the Functional Element fails to read boundaries from the configuration
5561 file. The Functional Element in turn return an error message and the use case end.

5562 2: Fail To Communicate With the Server

5563 2.1: In basic flow 3, the Functional Element fails to communicate with the servers hosting the
5564 services. The Functional Element in turn return an error message and the use case end.

5565 **2.22.7.1.3 Special Requirements**

5566 The protocol of communicating with a server hosting the services is not standardized. Each
5567 server may offer different mechanism for retrieving the list of services hosted and as such, the
5568 extensibility this approach is severely limited.

5569 **2.22.7.1.4 Pre-Conditions**

5570 None.

5571 **2.22.7.1.5 Post-Conditions**

5572 None

5573

5574 **2.22.7.2 Log Performance Parameters**

5575 **2.22.7.2.1 Description**

5576 This use case allows the user to log the performance parameters of all the Web Services that is
5577 being hosted by an application that contains the Service Management Functional Element.

5578 **2.22.7.2.2 Flow of Events**

5579 **2.22.7.2.2.1 Basic Flow**

5580 The use case begins when the user wants to log the performance parameters of all the Web
5581 Services that is being hosted by an application that contains the Service Management Functional
5582 Element.

5583 1: The user sends a request to log the performance parameters of all the Web Services hosted.

5584 2: The Functional Element reads from a configuration file the performance parameter to be
5585 logged.

5586 3: The Functional Element extracts the performance parameters for the incoming message and
5587 stores them into the data store

5588 4: The Functional Element next extracts the performance parameters for the outgoing message
5589 and stores them into the data store

5590 5: The Functional Element stores the necessary information into the data store.

5591 **2.22.7.2.2.2 Alternative Flows**

5592 1: No Performance Parameter Found.

5593 1.1: In basic flow 2, the Functional Element discovers that the performance parameter to be
5594 logged is not configured. The Functional Element returns an error message and the use case
5595 ends.

5596 2: Data Store Not Available.

5597 2.1: In basic flow 5, the Functional Element detects that the data store is not available. The
5598 Functional Element returns an error message and the use case ends.

5599 **2.22.7.2.3 Special Requirements**

5600 None.

5601 **2.22.7.2.4 Pre-Conditions**

5602 None.

5603 **2.22.7.2.5 Post-Conditions**

5604 None.

5605

5606 **2.22.7.3 Authorize Usage**

5607 **2.22.7.3.1 Description**

5608 This use case describes the authentication process for invoking a Web Service that is being
5609 hosted by an application that contains the Service Management Functional Element.

5610 **2.22.7.3.2 Flow of Events**

5611 **2.22.7.3.2.1 Basic Flow**

5612 The use case starts when a user accesses a service.

5613 1: The user sends a request to invoke a particular Web Service.

5614 2: The Functional Element extracts the following information from the incoming message

5615 2.1: The username attribute that resides in the header of the incoming message

5616 3: The Functional Element extracts the access privilege associated with the service from the data
5617 store

5618 4: The Functional Element uses the Role and Access Management Functional Element to retrieve
5619 the role of the user.

5620 5: The Functional Element looks up the data store to determine if the user is authorized to access
5621 the service

5622 6: The Functional Element allows the request to be process and the use case ends.

5623 **2.22.7.3.2.2 Alternative Flow**

5624 1: Username header not found.

5625 1.1: In basic flow 2, the username attribute is not found in the header.

5626 1.2: The Functional Element denies access to the requested Web Service and returns an
5627 error message.

5628 2: Web Service access privilege not set.

5629 2.1: In basic flow 3, the Functional Element could not find the access privilege for the Web
5630 Service.

5631 2.2: The Functional Element denies access to the requested Web Service and returns an
5632 error message.

5633 3: Role and Access Management Functional Element not available

5634 3.1: In basic flow 4, the Functional Element could not find the Role and Access Management
5635 Functional Element.

5636 3.2: The Functional Element denies access to the requested Web Service and returns an
5637 error message.

5638 4: User not authorize

5639 4.1: In basic flow 5, the Functional Element looks up the data source and determines that the
5640 user does not have the required privilege to access the service.

5641 4.2: The Functional Element denies access to the requested Web Service and returns an
5642 error message.

5643 **2.22.7.3.3 Special Requirements**

5644 None.

5645 **2.22.7.3.4 Pre-Conditions**

5646 None.

5647 **2.22.7.3.5 Post-Conditions**

5648 None.

5649

5650 **2.22.7.4 Manage Additional Information**

5651 **2.22.7.4.1 Description**

5652 This use case helps to maintain the following attributes of a Web Service that is useful in
5653 determining if a particular user has the privilege to invoke it.

5654 Service Name. This is the name of the service to monitor

5655 Access level. This refers to the access level of the Web Services hosted

5656 Role Names. If a user's role matches any of the roles contained here, then he/she has the
5657 privilege to access the Web Service.

5658 **2.22.7.4.2 Flow of Events**

5659 **2.22.7.4.2.1 Basic Flow**

5660 This use case starts when user wants to manage services.

5661 1: The user specifies the additional information that he wants to create/update/delete/retrieve.

5662 2: Once the user provides the requested information, one of the sub-flows is executed.

5663 If the user provides '**Create Service Parameter**', then sub-flow 2.1 is executed.

5664 If the user provides '**Update Service Parameter**', then sub-flow 2.2 is executed.

5665 If the user provides '**Delete Service Parameter**', then sub-flow 2.3 is executed.

5666 If the user provides '**Retrieve Service Parameter**', then sub-flow 2.4 is executed.

5667 2.1: Create Service Parameter.

5668 2.1.1: The user specifies the service to create with the appropriate additional information.

5669 2.1.2: The Functional Element connects to the data store.

5670 2.1.3: The Functional Element saves the new service in the data store and the use case
5671 ends.

5672 2.2: Update Service Parameter.

5673 2.2.1: The user specifies the service to update with the appropriate additional information.

5674 2.2.2: The Functional Element connects to the data store.

5675 2.2.3: The Functional Element updates the service in the data store and the use case
5676 ends.

5677 2.3: Delete Service Parameter.

5678 2.3.1: The user specifies the service to delete.

5679 2.3.2: The Functional Element connects to the data store.

5680 2.3.3: The Functional Element deletes the service in the data store and the use case
5681 ends.

5682 2.4: Retrieve Service Parameter.

5683 2.4.1: The user specifies the service to retrieve.

5684 2.4.2: The Functional Element connects to the data store.

5685 2.4.3: The Functional Element retrieves the service from the data store and the use case
5686 ends.

5687 **2.22.7.4.2.2 Alternative Flows**

5688 1: Data Store Not Available.

5689 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
5690 is returned and the use case ends.

5691 **2.22.7.4.3 Special Requirements**

5692 None.

5693 **2.22.7.4.4 Pre-Conditions**

5694 None.

5695 **2.22.7.4.5 Post-Conditions**

5696 None.

5697 **2.23 Service Registry Functional Element**

5698 **2.23.1 Motivation**

5699 In a Web Service-enabled implementation, there exist the needs to maintain a central repository
5700 of all the services that are available. This facilitates service lookups as well as management of
5701 Web Services within the application that contains the Functional Element. In order to achieve
5702 these expectations, the Functional Element will cover the following aspects.

- 5703 • Simplify management of information in a XML registry server like UDDI and ebXML, and
- 5704 • Simplify information publish and query from a XML registry server like UDDI and ebXML.

5705

5706 This Functional Element fulfills the following requirements from the Functional Elements
5707 Requirements Document 02:

- 5708 • Primary Requirements
 - 5709 ○ PROCESS-031 to PROCESS-032,
 - 5710 ○ PROCESS-035, and
 - 5711 ○ MANAGEMENT-097 to MANAGEMENT-100
- 5712 • Secondary Requirements
 - 5713 ○ PROCESS-014.

5714

5715 **2.23.2 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

5716 **2.23.3 Key Features**

5717 Implementations of the Service Registry Functional Element are expected to provide the following
5718 key features:

- 5719 1. The Functional Element MUST provide the capability to facilitate the management of the
5720 following information in a UDDI or an ebXML compliant registry server.
- 5721 1.1. Organisation
 - 5722 1.2. Classification / Taxonomy
 - 5723 1.3. Web Service
 - 5724 1.4. tModel
 - 5725 1.5. Service Binding
- 5726 The management of this information includes registering, updating, deleting and searching.
- 5727 2. As part of Key Feature (1), the Functional Element MUST provide the ability to perform the
5728 operations specified across multiple registry servers.
- 5729 3. The Functional Element MUST provide a mechanism to enable single step publishing of
5730 services into registry servers
5731

5732 **2.23.4 Interdependencies**

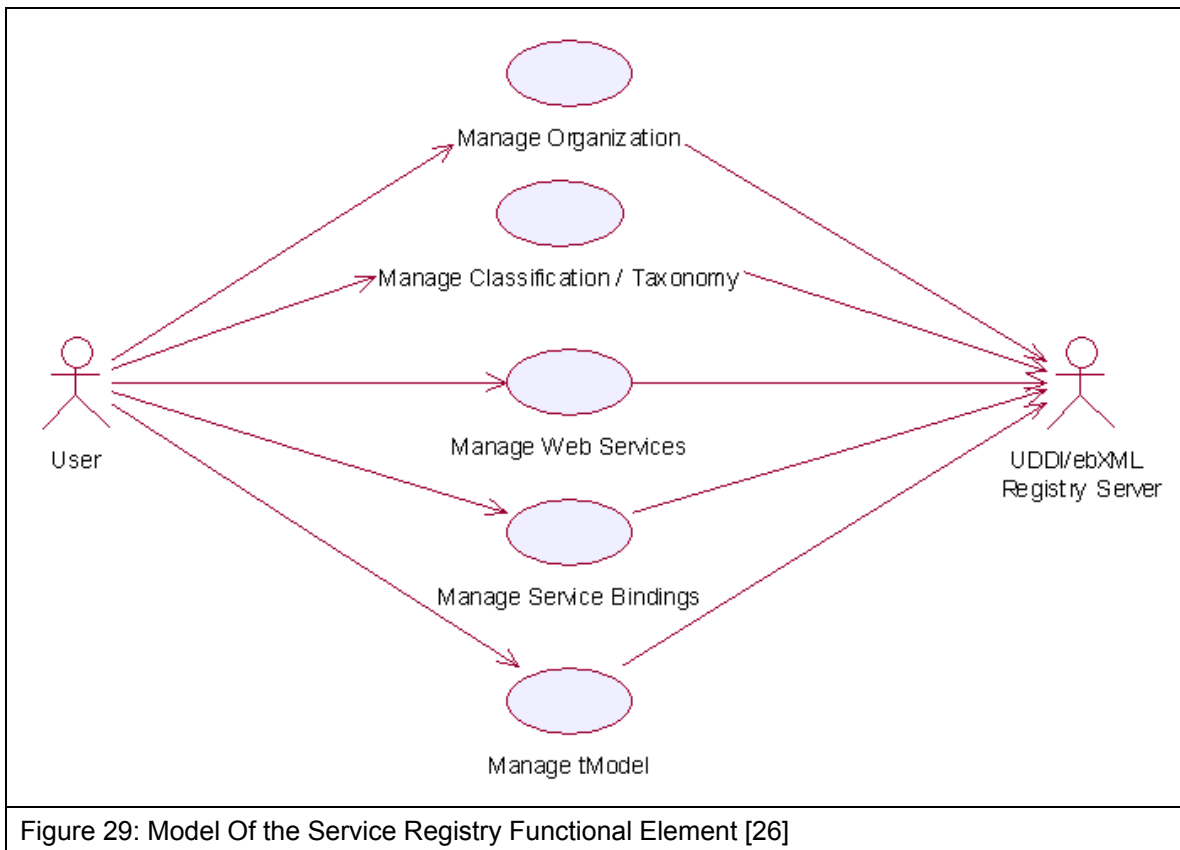
5733 None
5734

5735 **2.23.5 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. [22] 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. [23]
ebXML Registry Information Model (RIM) Specification v2.0 [24]	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 [25]	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.

5736

5737

2.23.6 Model5738 **2.23.7 Usage Scenario**5739 **2.23.7.1 Manage Classification / Taxonomy**5740 **2.23.7.1.1 Description**

5741 This use case allows any users to create, remove and view classification/taxonomy in the
 5742 registry.

5743 **2.23.7.1.2 Flow of Events**5744 **2.23.7.1.2.1 Basic Flow**

5745 This use case starts when the users of registry server wishes to create, remove or view the
 5746 classification/taxonomy in the registry server.

5747

5748 1: User initiates a request type to the Functional Element stating whether to create, remove or
 5749 view classification/taxonomy.

5750 2: The Functional Element checks whether the registry server exists.

5751 3: The Functional Element checks the request. Based on the type of request, one of the sub-
 5752 flows is executed.

5753 If the request is to '**Create Classification/Taxonomy**', then sub-flow 3.1 is executed.

5754 If the request is to '**View Classification/Taxonomy**', then sub-flow 3.2 is executed.

5755 If the request is to '**Remove Classification/Taxonomy**', then sub-flow 3.3 is executed.

5756 3.1: Create Classification/Taxonomy.

5757 3.1.1: Other Functional Element provides username, password and registry server URL
5758 to the Functional Element for authentication.

5759 3.1.2: The Functional Element checks for the user validity in the identified registry server.

5760 3.1.3: Other Functional Element provides classification/taxonomy information to be
5761 created in the registry server.

5762 3.1.4: The Functional Element checks for the duplicate classification/taxonomy name.

5763 3.1.5: The Functional Element creates the classification/taxonomy information in the
5764 private (default) or the public UDDI registry server according to the URL provided by
5765 other Functional Element, if it does not exist.

5766 3.2: View Classification/Taxonomy.

5767 3.2.1: The Functional Element retrieves all the classification/taxonomy from the identified
5768 registry server, which may be private (default) or public.

5769 3.2.2: The Functional Element returns the classification/taxonomy information from the
5770 identified registry server to other Functional Element.

5771 3.3: Remove Classification/Taxonomy.

5772 3.3.1: Other Functional Element provides username, password and registry server URL
5773 to the Functional Element for authentication.

5774 3.3.2: The Functional Element checks for the user validity in the identified registry server.

5775 3.3.3: Other Functional Element provides classification/taxonomy key (i.e. UUID) to be
5776 removed from the identified registry server.

5777 3.3.4: The Functional Element removes the classification/taxonomy information from the
5778 private (default) or the public UDDI registry server according to the URL provided by the
5779 user.

5780 4: The Functional Element returns the status of the operation and the use case ends.

5781 **2.23.7.1.2.2 Alternative Flows**

5782 1: Registry Server Down.

5783 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element
5784 returns an error message and the use case ends.

5785 2: Invalid Username And Password.

5786 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
5787 Element returns an error message and the use case ends.

5788 3: Classification/Taxonomy Key Not Found.

5789 3.1: In the basic flow 3.3.3, if the classification/taxonomy key cannot be found in the
5790 specified registry server, the Functional Element returns an error message and the use
5791 case ends.

5792 4: Duplicate Classification/Taxonomy.
5793 4.1: In the basic flow 3.1.4, If the same classification/taxonomy name has been defined in
5794 the registry server, the Functional Element returns an error message and the use case
5795 ends.

5796 **2.23.7.1.3 Special Requirements**
5797 None

5798 **2.23.7.1.4 Pre-Conditions**
5799 In order to manage the classification/taxonomy in the registry server, users must be registered
5800 with the registry server. Username and password will be given when a user registers with a
5801 registry server.

5802 **2.23.7.1.5 Post-Conditions**
5803 None.

5804 **2.23.7.2 Manage Web Services**

5805 **2.23.7.2.1 Description**
5806 This use case allows any users to register, remove and view Web Services in the private (default)
5807 as well as the public UDDI Registry Server.

5808 **2.23.7.2.2 Flow of Events**

5809 **2.23.7.2.2.1 Basic Flow**
5810 This use case starts when the users of registry server wishes to create, remove and view Web
5811 Services.

5812 1: User initiates a request type to the Functional Element stating whether to create, remove or
5813 view Web Services in the identified private or public registry server.

5814 2: The Functional Element checks whether the registry server exists.

5815 3: The Functional Element checks the request. Based on the type of request, one of the sub-
5816 flows is executed.

5817 If the request is to '**Create Web Service**', then sub-flow 3.1 is executed.

5818 If the request is to '**View Web Services**', then sub-flow 3.2 is executed.

5819 If the request is to '**Remove Web Service**', then sub-flow 3.3 is executed.

5820 3.1: Create Web Service.

5821 3.1.1: User provides username, password and registry server URL to the Functional
5822 Element for authentication.

5823 3.1.2: The Functional Element checks for the user validity in the identified registry server.

5824 3.1.3: Other Functional Element provides Web Service information to be created in the
5825 registry server.

5826 3.1.4: The Functional Element creates the Web Service information in the private
5827 (default) or the public UDDI registry server according to the URL provided by other
5828 Functional Element.

5829 3.2: View Web Services.

5830 3.2.1: The Functional Element retrieves all the Web Services from the identified registry
5831 server for specific stated conditions like service name search, business name search,
5832 etc.

5833 3.2.2: The Functional Element displays the Web Services information search results from
5834 the identified registry server to other Functional Element.

5835 3.3: Remove Web Service

5836 3.3.1 User provides username, password and registry server URL to the Functional
5837 Element for authentication.

5838 3.3.2: The Functional Element checks for the user validity in the identified registry server.

5839 3.3.3: Other Functional Element provides Web Service key (i.e. UUID) to be removed
5840 from the identified registry server.

5841 3.3.4: The Functional Element removes the Web Service information from the private
5842 (default) or the public UDDI registry server according to the URL provided by other
5843 Functional Element.

5844 4: The Functional Element returns the results of the operation and the use case ends.

5845 **2.23.7.2.2.2 Alternative Flows**

5846 1: Registry Server Down.

5847 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element
5848 returns an error message and the use case ends.

5849 2: Invalid Username And Password.

5850 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
5851 Element returns an error message and the use case ends.

5852 3: Web Service Key Not Found.

5853 3.1: In the basic flow 3.3.3, if the Web Service key cannot be found in the specified registry
5854 server, the Functional Element returns an error message and the use case ends.

5855 **2.23.7.2.3 Special Requirements**

5856 **2.23.7.2.4 Pre-Conditions**

5857 In order to manage Web Services in the registry server, the users must be registered with the
5858 registry server. Username and password will be given when a user registers with a registry
5859 server.

5860 **2.23.7.2.5 Post-Conditions**

5861 None.

5862 **2.23.7.3 Manage Organization**

5863 **2.23.7.3.1 Description**

5864 This use case allows any users to create, remove and view organization in the registry.

5865 **2.23.7.3.2 Flow of Events**

5866 **2.23.7.3.2.1 Basic Flow**

5867 This use case starts when the users of registry server wishes to create, remove or view
5868 Organization.

5869 1: User initiates a request type to the Functional Element stating whether to create, remove or
5870 view Organization.

5871 2: The Functional Element checks whether the registry server exists.

5872 3: The Functional Element checks the request. Based on the type of request, one of the sub-
5873 flows is executed.

5874 If the request is to '**Create Organization**', then sub-flow 3.1 is executed.

5875 If the request is to '**View Organizations**', then sub-flow 3.2 is executed.

5876 If the request is to '**Remove Organization**', then sub-flow 3.3 is executed.

5877 3.1: Create Organization.

5878 3.1.1: Other Functional Element provides username, password and registry server URL
5879 to the Functional Element for authentication.

5880 3.1.2: The Functional Element checks for the user validity in the identified registry server.

5881 3.1.3: Other Functional Element provides organization information to be created in the
5882 registry server.

5883 3.1.4: The Functional Element checks for the duplicate organization name.

5884 3.1.5: The Functional Element creates the organization information in the private (default)
5885 or the public UDDI registry server according to the URL provided by other Functional
5886 Element, if it does not exist.

5887 3.2: View Organizations.

5888 3.2.1: The Functional Element retrieves all the organizations from the identified registry
5889 server for specific stated conditions like organization name, key, etc.

5890 3.2.2: The Functional Element returns the organization information from the identified
5891 registry server to other Functional Element.

5892 3.3: Remove Organization.

5893 3.3.1: Other Functional Element provides username, password and registry server URL
5894 to the Functional Element for authentication.

5895 3.3.2: The Functional Element checks for the user validity in the identified registry server.

5896 3.3.3: Other Functional Element provides Organization key (i.e. UUID) to be removed
5897 from the identified registry server.

5898 3.3.4: The Functional Element removes the Organization information from the private
5899 (default) or the public UDDI registry server according to the URL provided by the user.

5900 4: The Functional Element returns the status of the operation and the use case ends.

5901 **2.23.7.3.2 Alternative Flows**

5902 1: Registry Server Down.

5903 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element
5904 returns an error message and the use case ends.

5905 2: Invalid Username And Password.

5906 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
5907 Element returns an error message and the use case ends.

5908 3: Organization Key Not Found.

5909 3.1: In the basic flow 3.3.3, if the Organization key cannot be found in the specified registry
5910 server, the Functional Element returns an error message and the use case ends.

5911 4: Duplicate Organization.

5912 4.1: In the basic flow 3.1.4, If the same Organization name has been defined in the registry
5913 server the Functional Element returns an error message and the use case ends.

5914 **2.23.7.3.3 Special Requirements**

5915 None

5916 **2.23.7.3.4 Pre-Conditions**

5917 In order to manage Organization in the registry server, users must be registered with the registry
5918 server. Username and password will be given when a user registers with a registry server.

5919 **2.23.7.3.5 Post-Conditions**

5920 None.

5921 **2.23.7.4 Manage Service Binding**

5922 **2.23.7.4.1 Description**

5923 This use case allows any users to register, remove and view Service Binding in the private
5924 (default) as well as the public UDDI Registry Server.

5925 **2.23.7.4.2 Flow of Events**

5926 **2.23.7.4.2.1 Basic Flow**

5927 This use case starts when the users of registry server wishes to create, remove and view Service
5928 Binding.

5929 1: User initiates a request type to the Functional Element stating whether to create, remove or
5930 view Service Binding in the identified private or public registry server.

5931 2: The Functional Element checks whether the registry server exists.

5932 3: The Functional Element checks the request. Based on the type of request, one of the sub-
5933 flows is executed.

- 5934 If the request is to '**Create Service Binding**', then sub-flow 3.1 is executed.
- 5935 If the request is to '**View Service Bindings**', then sub-flow 3.2 is executed.
- 5936 If the request is to '**Remove Service Binding**', then sub-flow 3.3 is executed.
- 5937 3.1: Create Service Binding.
- 5938 3.1.1: User provides username, password and registry server URL to the Functional
5939 Element for authentication.
- 5940 3.1.2: The Functional Element checks for the user validity in the identified registry server.
- 5941 3.1.3: Other Functional Element provides Service Binding information to be created in the
5942 registry server.
- 5943 3.1.4: The Functional Element creates the Service Binding information in the private
5944 (default) or the public UDDI registry server according to the URL provided by other
5945 Functional Element.
- 5946 3.2: View Service Bindings.
- 5947 3.2.1: The Functional Element retrieves all the Service Bindings from the identified
5948 registry server for specific stated conditions like service binding key search, etc.
- 5949 3.2.2: The Functional Element displays the Service Bindings information search results
5950 from the identified registry server to other Functional Element.
- 5951 3.3: Remove Service Binding
- 5952 3.3.1 User provides username, password and registry server URL to the Functional
5953 Element for authentication.
- 5954 3.3.2: The Functional Element checks for the user validity in the identified registry server.
- 5955 3.3.3: Other Functional Element provides Service Binding key (i.e. UUID) to be removed
5956 from the identified registry server.
- 5957 3.3.4: The Functional Element removes the Service Binding information from the private
5958 (default) or the public UDDI registry server according to the URL provided by other
5959 Functional Element.
- 5960 4: The Functional Element returns the results of the operation and the use case ends.
- 5961 **2.23.7.4.2.2 Alternative Flows**
- 5962 1: Registry Server Down.
- 5963 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element returns
5964 an error message and the use case ends.
- 5965 2: Invalid Username And Password.
- 5966 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
5967 Element returns an error message and the use case ends.
- 5968 3: Service Binding Key Not Found.
- 5969 3.1: In the basic flow 3.3.3, if the Service Binding key cannot be found in the specified registry
5970 server, the Functional Element returns an error message and the use case ends.

5971 **2.23.7.4.3 Special Requirements**

5972 **2.23.7.4.4 Pre-Conditions**

5973 In order to manage Service Binding in the registry server, the users must be registered with the
5974 registry server. Username and password will be given when a user registers with a registry
5975 server.

5976 **2.23.7.4.5 Post-Conditions**

5977 None.

5978 **2.23.7.5 Manage tModel**

5979 **2.23.7.5.1 Description**

5980 This use case allows any users to register, remove and view tModel in the private (default) as
5981 well as the public UDDI Registry Server.

5982 **2.23.7.5.2 Flow of Events**

5983 **2.23.7.5.2.1 Basic Flow**

5984 This use case starts when the users of registry server wishes to create, remove and view tModel.

5985 1: User initiates a request type to the Functional Element stating whether to create, remove or
5986 view tModel in the identified private or public registry server.

5987 2: The Functional Element checks whether the registry server exists.

5988 3: The Functional Element checks the request. Based on the type of request, one of the sub-
5989 flows is executed.

5990 If the request is to '**Create tModel**', then sub-flow 3.1 is executed.

5991 If the request is to '**View tModels**', then sub-flow 3.2 is executed.

5992 If the request is to '**Remove tModel**', then sub-flow 3.3 is executed.

5993 3.1: Create tModel.

5994 3.1.1: User provides username, password and registry server URL to the Functional
5995 Element for authentication.

5996 3.1.2: The Functional Element checks for the user validity in the identified registry server.

5997 3.1.3: Other Functional Element provides tModel information to be created in the registry
5998 server.

5999 3.1.4: The Functional Element creates the tModel information in the private (default) or
6000 the public UDDI registry server according to the URL provided by other Functional
6001 Element.

6002 3.2: View tModels.

6003 3.2.1: The Functional Element retrieves all the tModels from the identified registry server
6004 for specific stated conditions like tModel name search, tModel key search, etc.

6005 3.2.2: The Functional Element displays the tModel information search results from the
6006 identified registry server to other Functional Element.

6007 3.3: Remove tModel.

6008 3.3.1 User provides username, password and registry server URL to the Functional
6009 Element for authentication.

6010 3.3.2: The Functional Element checks for the user validity in the identified registry server.

6011 3.3.3: Other Functional Element provides tModel key (i.e. UUID) to be removed from the
6012 identified registry server.

6013 3.3.4: The Functional Element removes the tModel information from the private (default)
6014 or the public UDDI registry server according to the URL provided by other Functional
6015 Element.

6016 4: The Functional Element returns the results of the operation and the use case ends.

6017 **2.23.7.5.2.2 Alternative Flows**

6018 1: Registry Server Down.

6019 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element returns
6020 an error message and the use case ends.

6021 2: Invalid Username And Password.

6022 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional
6023 Element returns an error message and the use case ends.

6024 3: tModel Key Not Found.

6025 3.1: In the basic flow 3.3.3, if the tModel key cannot be found in the specified registry server,
6026 the Functional Element returns an error message and the use case ends.

6027 **2.23.7.5.3 Special Requirements**

6028 **2.23.7.5.4 Pre-Conditions**

6029 In order to manage tModel in the registry server, the users must be registered with the registry
6030 server. Username and password will be given when a user registers with a registry server.

6031 **2.23.7.5.5 Post-Conditions**

6032 None.

6033 **2.24 Service Router Functional Element (new)**

6034 **2.24.1 Motivation**

6035 There have been concerns about enabling direct access to Web Services, especially services
6036 that handle sensitive information and can be potentially exploited. These are security concerns
6037 that must be addressed. Furthermore, there are organisations that are not keen to expose their
6038 Web Services' endpoints directly to potential users, especially in situations where the exact forms
6039 and types of users are hard to predict. This is typical for services that are made publicly available.
6040 Furthermore, these Web Services typically do not have in-built security or authentication
6041 capabilities and need to rely on other services to provide them. For such cases, it is important
6042 that access to these Web Services must be pre- and post-processed, as well as provided a
6043 façade or middle-layer between the invocations by users and the actual invocations to these
6044 services.

6045

6046 This Functional Element aims to fulfill the needs of this space by provide capability for easy and
6047 simple mechanisms for invoking such Web Services by:

- 6048 • Providing a façade to service requesters for services location transparency, services
6049 reliability.
- 6050 • Performing pre- and post- processing before and after web services invocation.

6051

6052 This Functional Element fulfills the following requirements from the Functional Elements
6053 Requirements Document 02:

- 6054 • Primary Requirements
 - 6055 ○ PROCESS-250 to PROCESS-260.
- 6056 • Secondary Requirements
 - 6057 ○ None

6058

6059 **2.24.2 Terms Used**

Terms	Description
Façade	Façade is exterior face or interface of a system, which hides the implementation details of the system.
Functional handler	Functional handler is a software component that performs certain business processing on the parameters passed.

6060

6061 Figure 30 depicts the basic concepts of how the participating entities collaborate together in the
6062 Service Router Functional Element. All the invocations from service client come to the Service
6063 router which servers as façade. The Service Router routes the invocation the actual web
6064 services. Functional handlers could be incorporated in the Functional Element. The functional
6065 handlers can be invoked before or after the actual web services are invoked.

6066

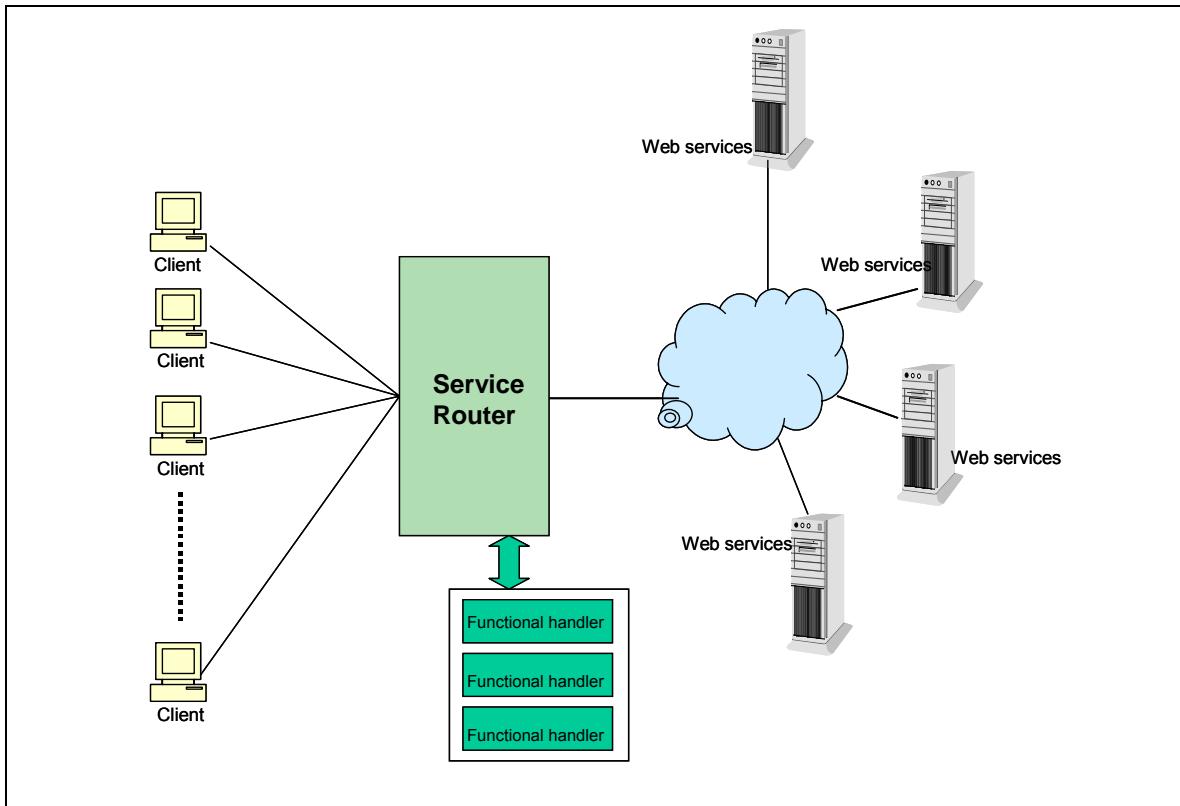


Figure 30: An Overview of the Service Router Functional Element

6067

6068 2.24.3 Key Features

6069 Implementations of the Service Router Functional Element are expected to provide the following
6070 key features:

- 6071 1. The Functional Element **MUST** provide mechanism as façade for web services invocations.
6072 This mechanism has the following capabilities:
 - 6073 1.1. Provide a single access point for web service invocation.
 - 6074 1.2. Provide the location transparency of actual web services.
- 6075 2. The Functional Element **MUST** provide capability to route web services invocation on
6076 behalf of service requesters to the specified actual web services.
- 6077 3. The Functional Element **MUST** provide capability to manage web services invocation in the
6078 aspects of invocation time-out, transaction management.
- 6079 4. The Functional Element **MUST** provide capability to manage the registration of web
6080 services that are going to be invoked.
- 6081 5. The Functional Element **MUST** provide capability to deploy registered web services
6082 automatically into the façade.
- 6083 6. The Functional Element **MUST** provide mechanism to incorporate functional handlers.
- 6084 7. The Functional Element **MUST** provide capability to perform processing by invoking
6085 functional handlers defined for a web services invocation before the web services is really
6086 invoked.
- 6087 8. The Functional Element **MUST** provide capability to perform processing by invoking
6088 functional handlers for a web services invocation after the web services is invoked.

- 6089 9. The Functional Element MUST provide capability to manage functional handlers.
6090 10. The Functional Element MUST provide capability to manage the parameter mappings
6091 between two adjacent functional handlers and parameter mapping between functional
6092 handler and web services.

6093

6094 In addition, the following key features could be provided to enhance the Functional Element
6095 further:

6096 1. The Functional Element MAY provide capability to invoke the alternative web services if the
6097 actual web services that is targeted to invoke is not available. The Functional Element MAY
6098 provide the capability to define a sequence of functional handlers for a web services for a
6099 web services invocation.

6100 1. The Functional Element MAY provide capability to enable the invocation of functional
6101 handlers in pre-defined sequence for a web for a web services invocation.

6102

6103 **2.24.4 Interdependencies**

6104 None.

6105

6106 **2.24.5 Related Technologies and Standards**

6107 None.

6108

6109 **2.24.6 Model**

6110

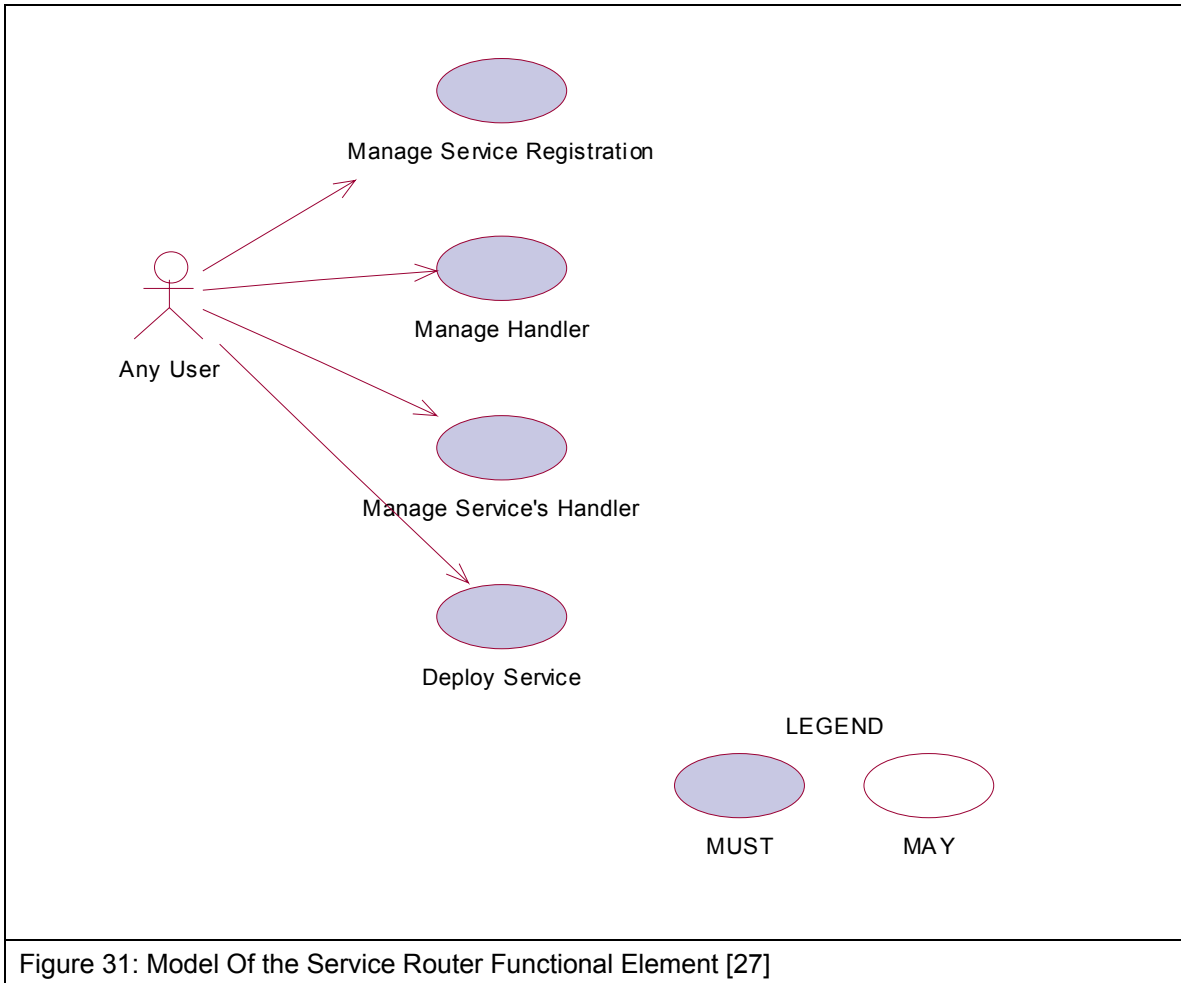


Figure 31: Model Of the Service Router Functional Element [27]

6111

6112 **2.24.7 Usage Scenarios**

6113 **2.24.7.1 Manage Service Registration**

6114 **2.24.7.1.1 Description**

6115 This use case allows the user to register, remove and view web services from or to the service
6116 router.

- 6117 • Register Web Service

6118 Web services details are registered to the service router.

- 6119 • Delete Web Service

6120 Web services are removed from the service router.

- 6121 • View Web Service
- 6122 View the registration information of a web service.
- 6123 **2.24.7.1.2 Flow of Events**
- 6124 **2.24.7.1.2.1 Basic Flow**
- 6125 This use case starts when the user of service router wishes to register, remove and view web
- 6126 services registration.
- 6127 1: The user initiates a request type to the Functional Element stating whether to register, remove
- 6128 or view web services registration in the service router.
- 6129 2: The Functional Element checks the request. Based on the type of request, one of the sub-
- 6130 flows is executed. If the request is to register a new web service in the service router, system
- 6131 executes 'Register Web Service'. If the request is to view web services from the service router,
- 6132 system executes 'View Web Services'. If the request is to remove a web service from the service
- 6133 router, system executes 'Remove Web Service'.
- 6134 2.1: Register Web Service.
- 6135 2.1.1: The user provides the WSDL of a web service.
- 6136 2.1.2: The user provides other web service information to be kept in the service
- 6137 router.
- 6138 2.1.3: The Functional Element retrieves web service information from the WSDL and
- 6139 keeps them into the registry.
- 6140 2.2: View Web Services.
- 6141 2.2.1: The Functional Element retrieves the service from the registry with the specific
- 6142 service name.
- 6143 2.2.2: The Functional Element returns the web services information results to the
- 6144 user.
- 6145 2.3: Remove Web Service
- 6146 2.3.1: The user provides web service name to be removed from the identified
- 6147 registry server.
- 6148 2.3.2: The Functional Element removes the web service information from the
- 6149 registry.
- 6150 3: The Functional Element responses the status of the operation whether it is successful or failure
- 6151 to the user and the use case ends.
- 6152 **2.24.7.1.2.2 Alternative Flows**
- 6153 1: WSDL error.
- 6154 1.1: In the Basic Flow 2.1.1, if the WSDL could not be retrieved, "WSDL error" will be sent
- 6155 back.
- 6156 2: Service does not exist

6157 2.1: In the Basic Flow 2.2.1 and 2.3.1, if the service name does not exist, "Service does not
6158 exist" error will be sent back.

6159 **2.24.7.1.3 Special Requirements**

6160 None.

6161 **2.24.7.1.4 Pre-Conditions**

6162 None.

6163 **2.24.7.1.5 Post-Conditions**

6164 None.

6165

6166 **2.24.7.2 Manage Handler**

6167 **2.24.7.2.1 Description**

6168 This use case allows any user to add, remove and view handler to the service router.

6169 **2.24.7.2.2 Flow of Events**

6170 **2.24.7.2.2.1 Basic Flow**

6171 This use case starts when the user of registry server wishes to add, remove or view web service
6172 handlers.

6173 1: The user initiates a request type to the Functional Element stating whether to add, remove or
6174 view web service handlers.

6175 2: The Functional Element checks the request. Based on the type of request, one of the sub-
6176 flows is executed. If the request is to add a new web service handler to the router, system
6177 executes 'Add Service Handler'. If the request is to view web service handlers, system executes
6178 'View Service Handlers'. If the request is to remove a handler from the router, system executes
6179 'Remove Service Handler'.

6180 2.1: Add Service Handler.

6181 2.1.1: The user provides handler name and location to The Functional Element.

6182 2.1.2: The service adds the information to the registry.

6183 2.2: View Service Handlers.

6184 2.2.1: The Functional Element receives a handler name from the user.

6185 2.2.2: The Functional Element returns the information of the handler to the user.

6186 2.3: Remove Service Handler.

6187 2.3.1: The user provides handler name to be removed from the service router.

6188 2.3.2: The Functional Element removes the service handler from the registry.

6189 3: The Functional Element responses the status of the operation whether it is successful or failure
6190 to the user and the use case ends.

6191 **2.24.7.2.2 Alternative Flows**

6192 1: Handler name error.

6193 1.1: In the Basic Flow 2.2.1 and 2.3.1, if the handler name does not exist, system displays an
6194 error message and exits the use case.

6195

6196 **2.24.7.2.3 Special Requirements**

6197 None.

6198 **2.24.7.2.4 Pre-Conditions**

6199 None.

6200 **2.24.7.2.5 Post-Conditions**

6201 None.

6202

6203 **2.24.7.3 Manage Service's Handler**

6204 **2.24.7.3.1 Description**

6205 This use case allows the user to add, remove and view handlers to the services registered in the
6206 service router.

6207 • Add a handler to a service

6208 New handler is added to a registered service.

6209 • Remove a handler to a service

6210 Existing handler is removed from a registered service.

6211 • View service's handler

6212 Existing handlers of a service could be viewed by the user.

6213 **2.24.7.3.2 Flow of Events**

6214 **2.24.7.3.2.1 Basic Flow**

6215 This use case starts when the user of service router wishes to add, remove or view handlers to a
6216 service.

6217 1: The user initiates a request type to the Functional Element stating whether to add, remove or
6218 view handlers to a service.

6219 2: The Functional Element checks the request. Based on the type of request, one of the sub-
6220 flows is executed. If the request is to add a new web service handler to a registered web service,
6221 system executes 'Add Service Handler'. If the request is to view web service handlers, system
6222 executes 'View Service Handlers'. If the request is to remove a handler from a service, system
6223 executes 'Remove Service Handler'.

- 6224 2.1: Add Service Handler.
- 6225 2.1.1: The user provides handler name, service name and parameter mappings to The
6226 Functional Element.
- 6227 2.1.2: The service adds the information to the registry.
- 6228 2.2: View Service Handlers.
- 6229 2.2.1: The Functional Element receives the service name from the user.
- 6230 2.2.2: The Functional Element retrieves all the handlers and return to the user.
- 6231 2.3: Remove Service Handler.
- 6232 2.3.1: The user provides handler name and service name to be removed from the
6233 service router.
- 6234 2.3.2: The Functional Element removes the service handler from the registry.
- 6235 3: The Functional Element responses the status of the operation whether it is successful or failure
6236 to the user and the use case ends.
- 6237 **2.24.7.3.2 Alternative Flows**
- 6238 1: Handler name or service name does not exist.
- 6239 1.1: In the Basic Flow 2.1.1, 2.2.1 and 2.3.1, if the service name or the handler name does
6240 not exist, system displays an error message and exits the use case.
- 6241 **2.24.7.3.3 Special Requirements**
- 6242 None.
- 6243 **2.24.7.3.4 Pre-Conditions**
- 6244 None.
- 6245 **2.24.7.3.5 Post-Conditions**
- 6246 None.
6247
- 6248 **2.24.7.4 Deploy Service**
- 6249 **2.24.7.4.1 Description**
- 6250 This use case allows the user to deploy registered services to an application server.
- 6251 • Add server information to The Functional Element
- 6252 New server is added to a registered service.
- 6253 • Remove server information to The Functional Element
- 6254 Existing server is removed from a registered service.
- 6255 • View server information

6256 Existing server information could be viewed by the user.

6257 • Deploy service

6258 Deploy a registered service to a server.

6259 .

6260 **2.24.7.4.2 Flow of Events**

6261 **2.24.7.4.2.1 Basic Flow**

6262 This use case starts when the user of service router wishes to add, remove, view server
6263 information or deploy a web service to a server.

6264 1: The user initiates a request type to the Functional Element stating whether to add, remove or
6265 view server's information or deploy service.

6266 2: The Functional Element checks the request. Based on the type of request, one of the sub-
6267 flows is executed. If the request is to add a server to the router, system executes 'Add Server'. If
6268 the request is to view server information, system executes 'View Server'. If the request is to
6269 remove a server from the router, system executes 'Remove Server'. If the request is to deploy a
6270 service to a server, system executes 'Deploy Service'.

6271 2.1: Add Server.

6272 2.1.1: The user provides server name and location of the server.

6273 2.1.2: The service adds the information to the registry.

6274 2.2: View Server.

6275 2.2.1: The Functional Element receives the server name from the user.

6276 2.2.2: The Functional Element retrieves the information and return to the user.

6277 2.3: Remove Server.

6278 2.3.1: The user provides the server name from the service router.

6279 2.3.2: The Functional Element removes the server from the registry.

6280 2.4: Deploy Service.

6281 2.4.1: The user provides the server name and service name from the service router.

6282 2.4.2: The Functional Element generate code package the service and deploy it to
6283 the server.

6284 3: The Functional Element responses the status of the operation whether it is successful or failure
6285 to the user and the use case ends..

6286 **2.24.7.4.2.2 Alternative Flows**

6287 1: Service name or server name does not exist.

6288 1.1: In the Basic Flow 2.2.1, 2.3.1 and 2.4.1, if the service name or the server name does not
6289 exist, system displays an error message and exits the use case.

6290

6291 **2.24.7.4.3 Special Requirements**

6292 None.

6293 **2.24.7.4.4 Pre-Conditions**

6294 None.

6295 **2.24.7.4.5 Post-Conditions**

6296 None.

6297

6298 **2.25 Service Tester Functional Element**

6299 **[Deprecated]**

6300 This Functional Element has been deprecated in this version. Please refer to its replacement,
6301 2.15 Quality of Service (QoS) Functional Element (new) for further details.

6302 **2.26 Transformer Functional Element (new)**

6303 **2.26.1 Motivation**

6304 Different applications support different format of files or message. Sometimes same information
6305 needs to be represented in different format in different use cases. This element tries to provide a
6306 framework to facilitate transformation between files or messages.

6307

6308 This Functional Element fulfills the following requirements from the Functional Elements
6309 Requirements Document 02:

- 6310 • Primary Requirements
 - 6311 ○ DELIVERY-150,
 - 6312 ○ DELIVERY-151,
 - 6313 ○ DELIVERY-152,
 - 6314 ○ DELIVERY-153,
 - 6315 ○ DELIVERY-155, and
 - 6316 ○ DELIVERY-157.
- 6317 • Secondary Requirements
 - 6318 ○ None.

6319

6320 **2.26.2 Terms Used**

Terms	Description
API Handlers	Binary components which are deployed at the same location as the element. This component provides a set of APIs for the element to invoke to transform files or messages.
Web Services Handler	A web service which are used by the element to invoke to transform files or messages.
WSDL	Web Services Description Language
XSLT	Extensible Stylesheet Language Transformation

6321

6322 Figure 32 depicts the basic concepts of 2 steps approach of Transformer Functional Element.
6323 Step 1 begins when the user (service requester) requests to define supported message, file
6324 types, XSLT templates and process handlers. The Function Element persists these definitions
6325 the return the results. Step 2 begins when the user requests for file or message transformation.
6326 The user provides messages or files to be transformed. The Functional Element will do the
6327 transformation and returns the result to the user.

6328

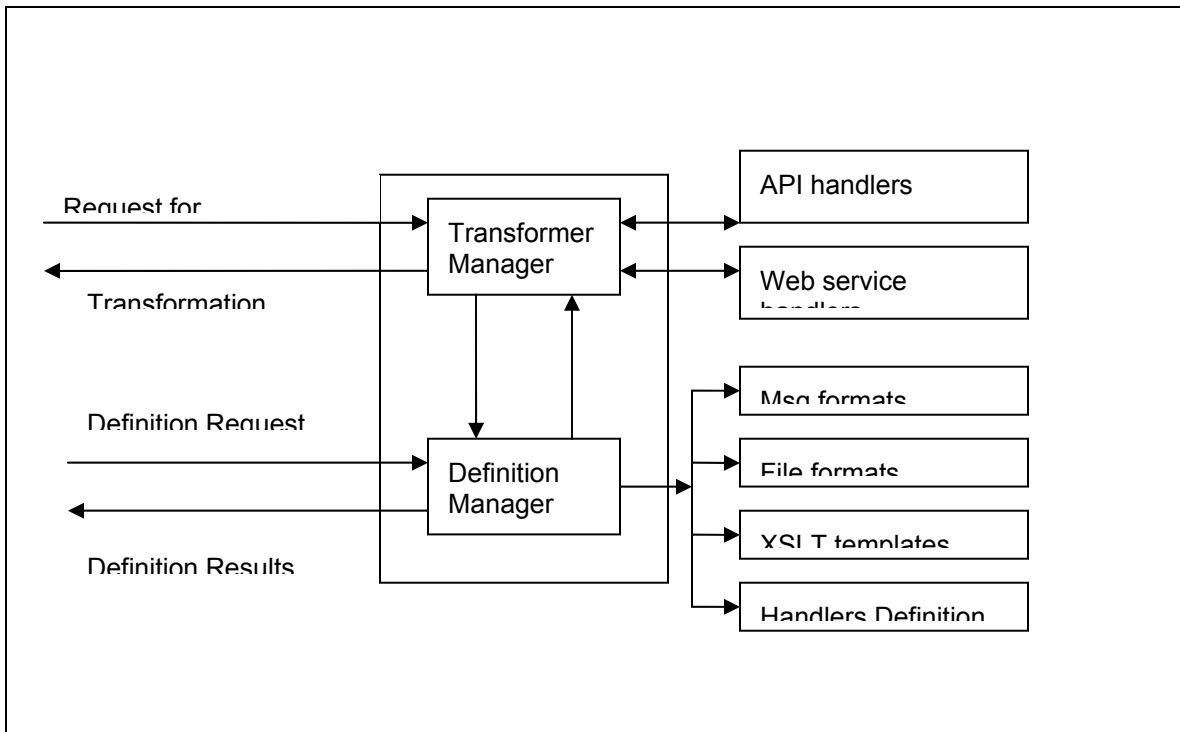


Figure 32: An Overview of Transformer Functional Element

6330

6331 2.26.3 Key Features

6332 Implementations of the Transformer Functional Element are expected to provide the following key
6333 features:

- 6334 1. The Functional Element MUST provide the capability to manage supported files and
6335 messages.
- 6336 2. The Functional Element MUST provide the capability to manage XSLT templates.
- 6337 3. The Functional Element MUST provide the capability to manage handlers for transformation.
- 6338 4. The Functional Element MUST provide the handler to transform SOAP, WSDL messages.

6339

6340 In addition, the following key features could be provided to enhance the Functional Element
6341 further:

- 6342 1. The Functional Element MAY provide the capability to chain handlers.
- 6343 2. The Functional Element MAY provide the capability to measure the performance of handlers.
- 6344 3. The Functional Element MAY provide the capability to select the efficient handlers to do the
6345 transformation.

6346

6347 2.26.4 Interdependencies

Direct Dependency

Log Utility Functional Element	The Log Utility Functional Element is used to record the data.
--------------------------------	--

6348

6349 2.26.5 Related Technologies and Standards

Specifications	Description
SOAP 1.2	The ability to parse the SOAP message.
WSDL 1.1	The ability to parse the WSDL.

6350

6351 2.26.6 Model

6352

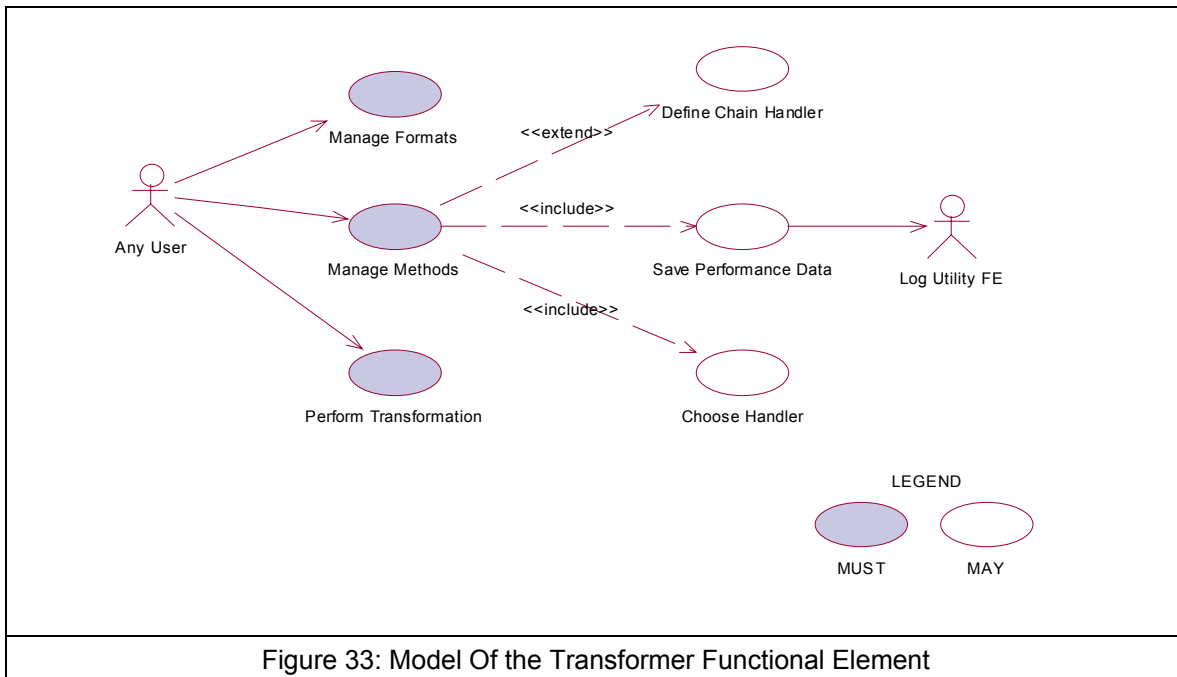


Figure 33: Model Of the Transformer Functional Element

6353

6354 2.26.7 Usage Scenarios

6355 2.26.7.1 Manage Formats

6356 2.26.7.1.1 Description

6357 This use case allows the user to manage file or message formats supported by this element.

6358 2.26.7.1.2 Flow of Events

6359 2.26.7.1.2.1 Basic Flow

6360 This use case starts when the user wants to manage file or message formats.

6361 1: The user provides the management operation to the functional element.

6362 2: Based on the operation one of the following sub-flow is executed. If the operation is “add-
6363 format” sub-flow 2.1 is executed. If the operation is “delete-format” sub-flow 2.2 is executed. If
6364 the operation is “query-format” sub-flow 2.3 is executed.

6365 2.1: Add format

6366 2.1.1: The system gets the format name, file extension name.

6367 2.1.2: The system save this information.

6368 2.2: Delete format

6369 2.2.1: The system gets the format name.

6370 2.2.2: The system deletes format information.

6371 2.3: Query format:

6372 2.3.1: The system gets the format name.

6373 3: The Functional Element responses the status of the operation whether it is successful or failure
6374 to the user and the use case ends.

6375 **2.26.7.1.2.2 Alternative Flows**

6376 1: Format Name Already Registered.

6377 1.1 In Basic Flow 2.1.2, if the format name already registered, the system will assign error
6378 message to the result message.

6379 2: Format Name Does Not Exist

6380 2.1 In Basic Flow 2.2.2, if the format name does not exist, the system will assign error
6381 message to the result message.

6382 **2.26.7.1.3 Special Requirements**

6383 None.

6384 **2.26.7.1.4 Pre-Conditions**

6385 None.

6386 **2.26.7.1.5 Post-Conditions**

6387 None.

6388

6389

6390 **2.26.7.2 Manage Methods**

6391 **2.26.7.2.1 Description**

6392 This use case allows the user to manage the methods that are used to do the transformation.

6393 **2.26.7.2.2 Flow of Events**

6394 **2.26.7.2.2.1 Basic Flow**

6395 This use case starts when a user wants to manage the methods that are used to do the
6396 transformation.

6397 1. The user provides the management operation and data.

6398 2. Based on the operation it specified, one of the following sub-flows is expected. If the operation
6399 is 'Add Method', then sub-flow 2.1 is executed. If the operation is 'Delete Method', then sub-flow
6400 2.2 is executed. If the operation is "Query Method", then sub-flow 2.3 is executed.

6401 2.1: Add Method.

6402 2.1.1: The user sets the file method name, type (API or Web Service), Input file format
6403 location and Output file format location, or user submits the WDSL of a known web
6404 service.

6405 2.1.2: The system save this information.

6406 2.2: Delete Method.

6407 2.2.1: The user sets the method name.

6408 2.2.2: The system deletes this information

6409 2.3: Query Method.

6410 2.3.1: The user sets the method name, or input format, or output format.

6411 3: The Functional Element responses the status of the operation whether it is successful or failure
6412 to the user and the use case ends.

6413 **2.26.7.2.2.2 Alternative Flows**

6414 1: Method Name Already Registered.

6415 1.1 In Basic Flow 2.1.2, if the format name already registered, the system will assign error
6416 message to the result message.

6417 2: Method Name Does Not Exist.

6418 2.1 In Basic Flow 2.2.2, if the format name does not exist, the system will assign error
6419 message to the result message.

6420 **2.26.7.2.3 Special Requirements**

6421 None.

6422 **2.26.7.2.4 Pre-Conditions**

6423 None.

6424 **2.26.7.2.5 Post-Conditions**

6425 None.

6426

6427

6428 **2.26.7.3 Perform Transformation**

6429 **2.26.7.3.1 Description**

6430 This use case allows the user to transform a file from one format to another format.

6431 **2.26.7.3.2 Flow of Events**

6432 **2.26.7.3.2.1 Basic Flow**

6433 This use case starts when a user wants to transform a file from one format to another format.

6434 1: The user set the file name to be transformed and the destination format.

6435 2: The system checks all the methods which use this file as input.

6436 3: The system checks all the methods which use the destination format as output.

6437 4: Select one method based on the performance data recorded before.

6438 5: Invoke the methods and save the performance data.

6439 6: Return the results and the use case ends.

6440 **2.26.7.3.2.2 Alternative Flows**

6441 1: If in Basic Flow 4 there is there is no method to do the transformation, the system return error
6442 message to the user and this use case ends.

6443 **2.26.7.3.3 Special Requirements**

6444 None.

6445 **2.26.7.3.4 Pre-Conditions**

6446 None.

6447 **2.26.7.3.5 Post-Conditions**

6448 None.

6449

6450

6451 **2.26.7.4 Define Chain Handler**

6452 **2.26.7.4.1 Description**

6453 This use case allows the user to create new handler based on the existing handler if a
6454 transformation could be done directly but could be done indirectly through a chain of existing
6455 handler.

6456 **2.26.7.4.2 Flow of Events**

6457 **2.26.7.4.2.1 Basic Flow**

6458 1: User sets the chain handler name and the handlers involved in this chain.

6459 2: The system gets the input format name of the first handler and the output format name of the
6460 last handler.

6461 3: The system save this information.

6462 4: Return the results to the user and end the use case.

6463 **2.26.7.4.2.2 Alternative Flows**

6464 1: If the handler name could not be found in Basic Flow 2, system returns the results to the user
6465 and the use case ends.

6466 **2.26.7.4.3 Special Requirements**

6467 None.

6468 **2.26.7.4.4 Pre-Conditions**

6469 None.

6470 **2.26.7.4.5 Post-Conditions**

6471 None.

6472

6473

6474 **2.26.7.5 Choose Handler**

6475 **2.26.7.5.1 Description**

6476 This use case allows the system to choose a handler for transformation.

6477 **2.26.7.5.2 Flow of Events**

6478 **2.26.7.5.2.1 Basic Flow**

6479 This use case starts when the transform use case needs a handler to do the transformation.

6480 1. The system checks the handlers that match the input and out put format.

6481 2: The system returns the name of the handler to the transform use case and ends this use case.

6482 **2.26.7.5.2.2 Alternate Flow**

6483 1: In Basic Flow 1, if there are more handlers available and performance data are available, then
6484 the system select the handler with the best performance data. Otherwise select any one.

6485 2: In Basic Flow 1, if the handler is a XSLT template, return the template name to the transform.

6486 **2.26.7.5.3 Special Requirements**

6487 None.

6488 **2.26.7.5.4 Pre-Conditions**

6489 None.

6490 **2.26.7.5.5 Post-Conditions**

6491 None.

6492

6493

6494 **2.26.7.6 Save Performance Data**

6495 **2.26.7.6.1 Description**

6496 This use case saves performance data of each handler.

6497 **2.26.7.6.2 Flow of Events**

6498 **2.26.7.6.2.1 Basic Flow**

6499 This use case starts when user wants to measure the performance of the handlers.

6500 1: It starts time counting.

6501 2: Collection CPU information, DISK access information and Network traffic information.

6502 3: Waiting for the termination of the handler.

6503 4: Save this information and end the use case.

6504 **2.26.7.6.2.2 Alternative Flows**

6505 1: In Basic Flow 3, If the log file is not available, the Functional Element returns an error and the
6506 user case ends.

6507 **2.26.7.6.3 Special Requirements**

6508 None.

6509 **2.26.7.6.4 Pre-Conditions**

6510 None.

6511 **2.26.7.6.5 Post-Conditions**

6512 None.

6513

6514

6515 **2.27 User Management Functional Element**

6516 **2.27.1 Motivation**

6517 The User Management Functional Element is expected to be an integral part of the user access
6518 management (UAM) functionalities that is expected to be needed by a Web Service-enabled
6519 implementation. This FE is expected to fulfill the needs arising out of managing resources within
6520 an application, with a user-centric viewpoint. As such it will cover aspects that include:

- 6521 • Basic user accounts management facilities,
- 6522 • Ability to extend dynamically from the basic set of account information,
- 6523 • Capability for configurable policies governing account management,
- 6524 • Providing log trails for user activities, and
- 6525 • Management of user authentication means, either directly or indirectly.

6526
6527 This Functional Element fulfills the following requirements from the Functional Elements
6528 Requirements Document 02:

- 6529 • Primary Requirements
 - 6530 ○ MANAGEMENT-001 to MANAGEMENT-003,
 - 6531 ○ MANAGEMENT-005,
 - 6532 ○ MANAGEMENT-008,
 - 6533 ○ MANAGEMENT-012, and
 - 6534 ○ SECURITY-002 (all).
- 6535 • Secondary Requirements
 - 6536 ○ SECURITY-001.

6537

6538 **2.27.2 Terms Used**

Terms	Description
Namespace	Namespace is use to segregate the instantiation of the application across different application domains. If a company has two separate standalone application, for example, an email application and an equipment booking application, then these two are considered as separate application domains.
User	A user is loosely defined to include both human and virtual users. Virtual users could include service users and application (or machine) users that are utilising other services in a SOA environment.

User Access Management / UAM	<p>User Access Management or UAM refer to the concept of managing users in a holistic manner, considering all aspect which includes:</p> <p>Defining a set of basic user information that should be stored in any enterprise application.</p> <p>Providing a means to extend this basic set of user information when needed.</p> <p>Simplifying management by grouping related users together through certain criteria.</p> <p>Having the flexibility of adopting both coarse/fine grain access controls.</p>
User Repository	User Repository is where the user information is stored. It can be a database or a flat file.

6539

6540 2.27.3 Key Features

6541 Implementations of the User Management Functional Element are expected to provide the
6542 following key features:

- 6543 1. The Functional Element MUST provide a User Repository.
- 6544 2. The Functional Element MUST be able to control access to such a User Repository.
- 6545 4. The Functional Element MUST provide a basic User structure with a set of pre-defined
6546 attributes.
- 6547 5. The Functional Element MUST provide the capability to extend on the basic User structure
6548 dynamically.
- 6549 6. As part of Key Feature (4), this dynamic extension MUST be definable and configurable at
6550 runtime implementation of the Functional Element.
- 6551 7. The Functional Element MUST provide the capability to manage the creation and deletion of
6552 instances of Users based on defined structure.
- 6553 8. The Functional Element MUST provide the capability to manage all the information (attribute
6554 values) stored in such Users. This includes the capability to:
 - 6555 2.1. Retrieve and update attribute's values belonging to a User,
 - 6556 2.2. Generate a random password,
 - 6557 2.3. Encrypt sensitive user information, and
 - 6558 2.4. Authenticate a user.
- 6559 9. As part of Key Feature (7.4), the authentication of a User MUST be achieved at least through
6560 the use of a password.
- 6561 10. The Functional Element MUST provide a mechanism for managing Users across different
6562 application domains.

6563 *Example: Namespace control mechanism*

6564

6565 In addition, the following key features could be provided to enhance the Functional Element
6566 further:

- 6567 1. The Functional Element MAY provide a mechanism to control the username format.
6568 *Example: Usernames must be at least 8 characters long.*
- 6569 2. The Functional Element MAY provide additional security mechanisms to enhance the
6570 security of sensitive information like user passwords.

- 6571 *Example: Passwords are stored in security tokens, or a more secure encryption algorithms*
6572 *for passwords.*
- 6573 11. If Key Feature (2) is provided, the Functional Element MAY also provide a selection of
6574 selectable encryption algorithms.
- 6575 3. The Functional Element MAY provide additional security policies to ensure that systems are
6576 not compromised.
- 6577 *Example: Passwords must be changed every 30 days.*
- 6578 12. If Key Feature (4) is provided, the Functional Element MAY also provide a facility to notify
6579 users before the password expires.
6580

6581 **2.27.4 Interdependencies**

Interaction Dependencies	
Group Management Functional Element	The Group Management Functional Element may be used to provide useful aggregation of the users.
Phase and Lifecycle Management Functional Element	The Phase and Lifecycle Management Functional Element may be used to maintain the relationships between various phases of a project lifecycle and the group who is working on it.
Role and Access Management Functional Element	The Role and Access Management Functional Element may be used to manage the user's access rights by virtue of it's association with a group, phase or even the complete lifecycle of the project.

6582

6583 **2.27.5 Related Technologies and Standards**

6584 None

6585 **2.27.6 Model**

6586

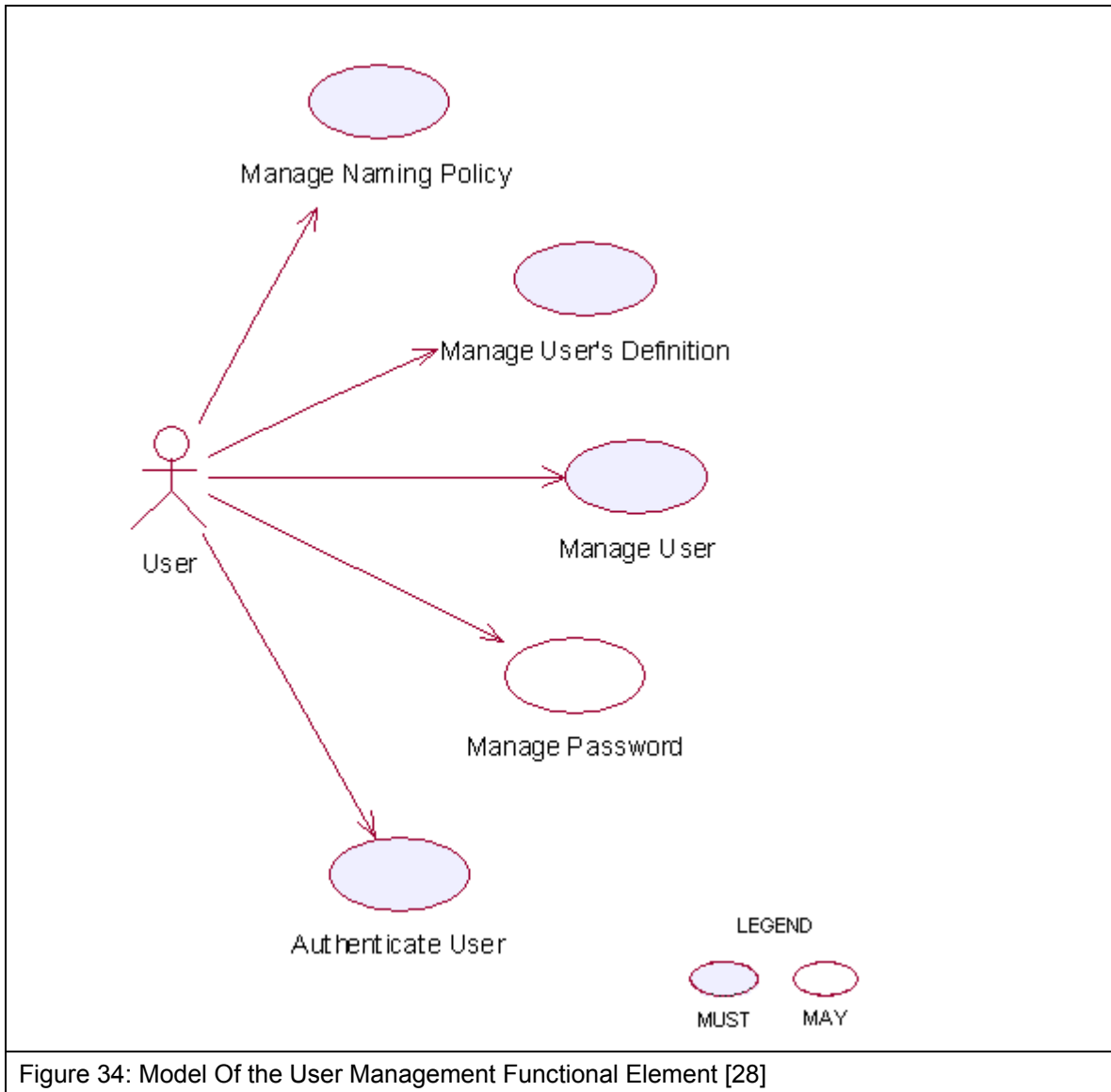


Figure 34: Model Of the User Management Functional Element [28]

6587 **2.27.7 Usage Scenarios**

6588 **2.27.7.1 Manage Naming Policy**

6589 **2.27.7.1.1 Description**

6590 This use case allows any user to manage naming policy when creating/updating user accounts.
 6591 The service user may create, update, retrieve and delete a naming policy.

6592 **2.27.7.1.2 Flow of Events**

6593 **2.27.7.1.2.1 Basic Flow**

6594 This use case starts when any user wants to manage naming policy for creating/updating user
 6595 account.

6596 1: The user sends Manage Naming Policy request to the Functional Element together with the
6597 specified operation.

6598 2: Functional Element gets the operation. Based on the operation, one of the sub-flows is
6599 executed.

6600 If the service user provides '**Create Naming Policy**', then sub-flow 2.1 is executed.

6601 If the service user provides '**Update Naming Policy**', then sub-flow 2.2 is executed.

6602 If the service user provides '**Delete Naming Policy**', then sub-flow 2.3 is executed.

6603 2.1: Create Naming Policy.

6604 2.1.1: The service user specifies namespace, name and description of the policy to
6605 create, for example, the policy name may be name length, the policy description may be
6606 "=7".

6607 2.1.2: The Functional Element checks the existing naming policy.

6608 2.1.3: The Functional Element generates naming policy information and adds to the
6609 Functional Element and the use case ends.

6610 2.2: Update Naming Policy.

6611 2.2.1: The service user specifies the policy to update.

6612 2.2.2: The Functional Element retrieves the existing naming policy information.

6613 2.2.3: The service user provides the update naming policy information according to the
6614 policy name used in creating a naming policy.

6615 2.2.4: The Functional Element updates the naming policy with the updated information
6616 and ends use case.

6617 2.3: Retrieve Naming Policy.

6618 2.3.1: The service user specifies the policy to retrieve.

6619 2.3.2: The Functional Element retrieves the existing naming policy information and ends
6620 the use case.

6621 2.4: Delete Naming Policy.

6622 2.4.1: The service user specifies the policy to delete.

6623 2.4.2: The Functional Element retrieves the existing naming policy information.

6624 2.4.3: The Functional Element deletes the naming policy from the Functional Element
6625 and the use case ends.

6626 **2.27.7.1.2.2 Alternative Flows**

6627 1: Invalid Policy.

6628 1.1: If in the basic flow 2.1.1, Functional Element detects any invalid description, Functional
6629 Element returns general error message and ends the use case.

6630 2: Naming Policy already exists.

6631 2.1: If in the basic flow 2.1.2, the Functional Element checks the existing naming policy and
6632 finds the naming policy already exists. The Functional Element returns an error and ends the
6633 use case.

6634 **2.27.7.1.3 Special Requirements**

6635 **2.27.7.1.4 Pre-Conditions**

6636 None.

6637 **2.27.7.1.5 Post-Conditions**

6638 If the use case was successful, the naming policy information is added to the Functional Element.
6639 To do any creating and updating of User information after the naming policy is added must satisfy
6640 the naming policies defined. If unsuccessful, the Functional Element's state is unchanged.

6641 **2.27.7.2 Manage User Definition**

6642 **2.27.7.2.1 Description**

6643 The use case allows any user to manage user definition when more basic user definition can not
6644 satisfied in creating/updating user accounts. The service user may create, update, retrieve and
6645 delete a user definition.

6646 **2.27.7.2.2 Flow of Events**

6647 **2.27.7.2.2.1 Basic Flow**

6648 This use case starts when any user wants to manage user definition for creating/updating user
6649 account.

6650 1: The user sends Manage User Definition request to the Functional Element together with the
6651 specified operation.

6652 2: Functional Element gets the operation. Based on the operation, one of the sub-flows is
6653 executed.

6654 If the service user provides '**Create User Definition**', then sub-flow 2.1 is executed.

6655 If the service user provides '**Update User Definition**', then sub-flow 2.2 is executed.

6656 If the service user provides '**Delete User Definition**', then sub-flow 2.3 is executed.

6657 2.1: Create User Definition.

6658 2.1.1: The service user specifies namespace, name and description of the user definition
6659 fields to create.

6660 2.1.2: The Functional Element checks the existing user definition fields (including basic
6661 ones).

6662 2.1.3: The Functional Element generates user definition information and adds to the
6663 Functional Element and the use case ends.

6664 2.2: Update User Definition.

6665 2.2.1: The service user specifies the user definition field to update.

6666 2.2.2: The Functional Element retrieves the existing user definition information.

6667 2.2.3: The service user provides the update user definition information.

6668 2.2.4: The Functional Element updates the user definition with the updated information
6669 and ends use case.

6670 2.3: Retrieve User Definition.

6671 2.3.1: The service user specifies the user definition to retrieve.

6672 2.3.2: The Functional Element retrieves the existing user definition information and ends
6673 the use case.

6674 2.4: Delete User Definition.

6675 2.4.1: The service user specifies the user definition to delete.

6676 2.4.2: The Functional Element retrieves the existing user definition information.

6677 2.4.3: The Functional Element deletes the user definition from the Functional Element
6678 and the use case ends.

6679 **2.27.7.2.3 Alternative Flows**

6680 1: Invalid User Definition.

6681 1.1: If in basic flow 2.1.1, Functional Element detects any invalid description, Functional
6682 Element returns general error message and ends the use case.

6683 2: User Definition already exists.

6684 2.1: If in basic flow 2.1.2, the Functional Element checks the existing user definition and finds
6685 the user definition already exists. The Functional Element returns an error and ends the use
6686 case.

6687 3: User Definition not exists.

6688 3.1: If in basic flow 2.2.2, 2.3.2 and 2.4.2, the Functional Element checks the existing user
6689 definition and finds the user definition does not exist. The Functional Element returns an
6690 error and ends the use case.

6691 **2.27.7.2.4 Special Requirements**

6692 None

6693 **2.27.7.2.5 Pre-Conditions**

6694 None.

6695 **2.27.7.2.6 Post-Conditions**

6696 If the use case was successful, the user definition information is added to the Functional Element.
6697 Thereafter, when creating and updating User, the User information must satisfy the user definition
6698 defined earlier. If the use case fails, the Functional Element's state is unchanged.

6699 **2.27.7.3 Manage User**

6700 This use case describes the management of a user, namely the creation, deletion, retrieval and
6701 update of the user.

6702 **2.27.7.3.1 Flow of Events**

6703 **2.27.7.3.1.1 Basic Flow**

6704 This use case starts when the user wants to manage a user.

6705 If user wants to **'Create User**, then basic flow 1 is executed.

6706 If user wants to **'Retrieve User**, then basic flow 2 is executed.

6707 If user wants to **'Update User**, then basic flow 3 is executed.

6708 If user wants to **'Delete User**, then basic flow 4 is executed.

6709 1: Create User.

6710 1.1: User provides the information that is necessary for creating a user.

6711 1.2: The Functional Element validates the user information provided against the naming
6712 policy.

6713 1.3: The Functional Element validates the user information provided against the user's
6714 definition.

6715 1.4: Functional Element creates the user and the use case ends.

6716 2: Retrieve User.

6717 2.1: User provides the necessary information for retrieving the complete user's attributes.

6718 2.2: The Functional Element returns the user's information and the use case ends.

6719 3: Update User.

6720 3.1: User provides the necessary information for updating the group's attributes.

6721 3.2: The Functional Element validates the user's information provided against the naming
6722 policy.

6723 3.3: The Functional Element validates the user information provided against the user's
6724 definition.

6725 3.4: The Functional Element updates the user and the use case ends.

6726 4: Delete User.

6727 4.1: User provides the necessary information for deleting a user group.

6728 4.2: Functional Element deletes the user and the use case ends.

6729 **2.27.7.3.1.2 Alternative Flows**

6730 1: User Exist.

6731 1.1: In basic flow 1.4, if the Functional Element detects an identical user, the Functional
6732 Element returns an error message and the use case ends.

- 6733 2: User Does Not Exist.
- 6734 1.1: In basic flow 2.2, 3.4 and 4.2, if the Functional Element cannot find a user that matches
6735 the user's criteria, the Functional Element returns an error message and the use case ends.
- 6736 **2.27.7.3.2 Special Requirements**
- 6737 None.
- 6738 **2.27.7.3.3 Pre-Conditions**
- 6739 None.
- 6740 **2.27.7.3.4 Post-Conditions**
- 6741 None.
- 6742 **2.27.7.4 Authenticate User**
- 6743 **2.27.7.4.1 Description**
- 6744 This use case allows users to authenticate a user.
- 6745 **2.27.7.4.2 Flow of Events**
- 6746 **2.27.7.4.2.1 Basic Flow**
- 6747 This use case starts when users wish to authenticate a user.
- 6748 1: Users provide user name and password to Functional Element.
- 6749 2: The Functional Element checks the user name and password.
- 6750 3: The Functional Element returns the result to users and the use case ends.
- 6751 **2.27.7.4.2.2 Alternative Flows**
- 6752 None.
- 6753 **2.27.7.4.3 Special Requirements**
- 6754 None.
- 6755 **2.27.7.4.4 Pre-Conditions**
- 6756 None.
- 6757 **2.27.7.4.5 Post-Conditions**
- 6758 None.
- 6759 **2.27.7.5 Manage Password**
- 6760 This use case describes the management of password in this Functional Element.

6761 **2.27.7.5.1 Flow of Events**

6762 **2.27.7.5.1.1 Basic Flow**

6763 This use case starts when the user wants to obtain an encrypted password. This can be
6764 achieved via one of the following basic flow.

6765 If user wants to '**Generate Password**', then basic flow 1 is executed.

6766 If user wants to '**Encrypt Password**', then basic flow 2 is executed.

6767 1: Generate Password

6768 1.1: The user specifies the option of format of password among available options in the
6769 Functional Element.

6770 1.2: The Functional Element generates clear text password based on the format specified by
6771 the service user.

6772 1.3: The Functional Element includes "Encrypt Password" use case to encrypt the clear text
6773 password.

6774 1.4: The Functional Element returns the clear text password and encrypted password to user
6775 and the use case ends.

6776 2: Encrypt Password

6777 1.1: The user provides clear text password to Functional Element.

6778 1.2: The user specifies the encryption algorithm to be used.

6779 1.3: The Functional Element encrypts the clear text password.

6780 1.4: The Functional Element returns the encrypted password to user and the use case ends.

6781 **2.27.7.5.1.2 Alternative Flows**

6782 None.

6783 **2.27.7.5.2 Special Requirements**

6784 None.

6785 **2.27.7.5.3 Pre-Conditions**

6786 None.

6787 **2.27.7.5.4 Post-Conditions**

6788 None.

6789 **2.28 Web Service Aggregator Functional Element**

6790 **2.28.1 Motivation**

6791 In any Web Service-enabled application, it is expected that complex business functions have to
6792 be realized via aggregation of multiple Web Services. This Functional Element is expected to
6793 fulfill the needs arising out of Web Services composition. As such it will cover aspects that
6794 include:

- 6795 • Facilitating the composition of Web Services, and
- 6796 • Testing of aggregated Web Services.

6797

6798 This Functional Element fulfills the following requirements from the Functional Elements
6799 Requirements Document 02:

- 6800 • Primary Requirements
 - 6801 ○ PROCESS-010 to PROCESS-014.
- 6802 • Secondary Requirements
 - 6803 ○ PROCESS-131

6804

6805 **2.28.2 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.

6806

6807 The following diagram shows the meaning of the terms in the context of Web Services
6808 aggregation.

6809

6810

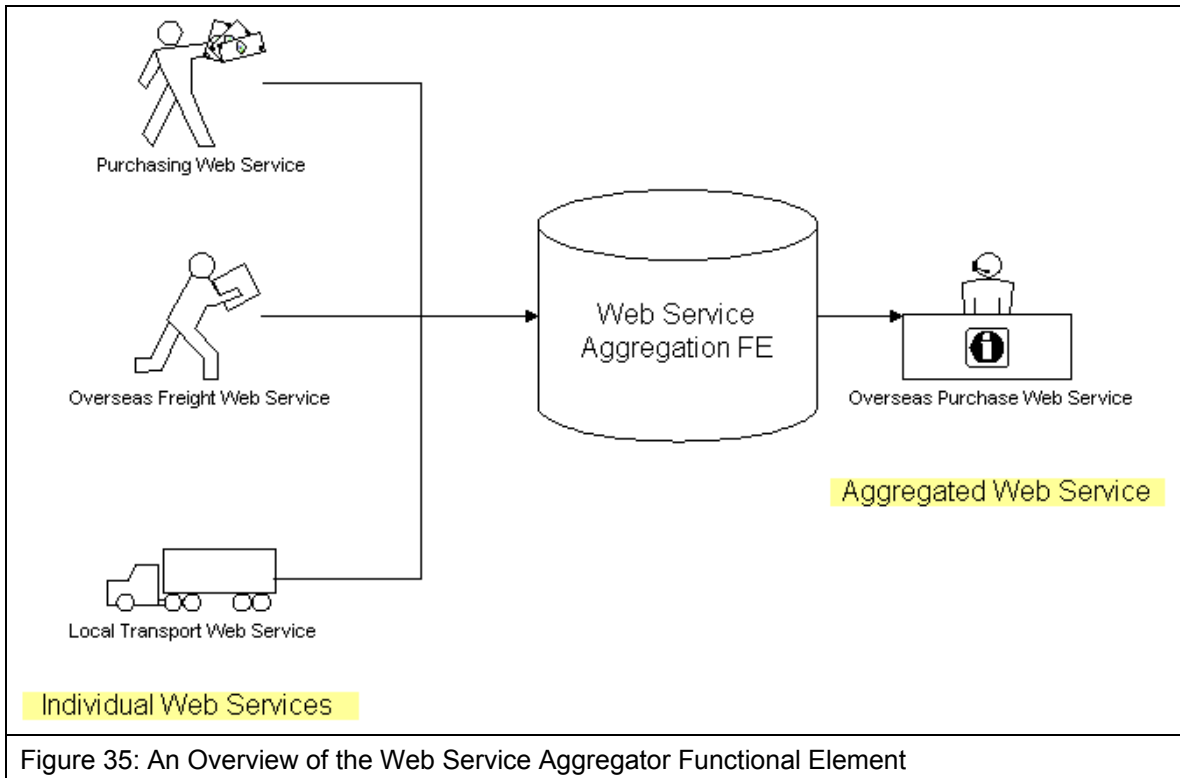


Figure 35: An Overview of the Web Service Aggregator Functional Element

6811

6812 2.28.3 Key Features

6813 Implementations of the Web Service Aggregator Functional Element are expected to provide the
6814 following key features:

- 6815 1. The Functional Element MUST provide a mechanism for composing any number of Web
6816 Services into single Web Service according to specified Composition Rule(s).
- 6817 2. Individual web services can reside at any location, but it is expected to be accessible.
- 6818 3. As part of Key Feature (1), the WSDL of a web service used for composition MUST be
6819 available.
- 6820 4. The Functional Element MUST support the definition, modification and removal of
6821 Composition Rules.
- 6822 5. The Functional Element MUST encapsulate the composition logic used into an interpretable
6823 XML-based script based on a particular standard*.

6824 *Example: BPEL or WSCI. The TC will have to decide on which standard to use*

6825

6826 In addition, the following key features could be provided to enhance the Functional Element
6827 further:

- 6828 1. The Functional Element MAY provide the capability to transform the interpretable XML-based
6829 script into an executable program.
- 6830 2. If Key Feature (1) is provided, then the Functional Element MAY also have the following
6831 capabilities:
 - 6832 2.1 The ability to test the functionality of the aggregated Web Service,
 - 6833 2.2 A WSDL to describe the aggregated Web Service, and
 - 6834 2.3 The capability to publish the aggregated Web Service into an UDDI-compliant registry

6835 **2.28.4 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

6836

6837 **2.28.5 Related Technologies and Standards**

Specifications	Specific References
Business Process Execution Language for Web Services version 2.0 [29]	Web Services Business Process Execution Language Version 2.0, Committee Draft, 01 September 2005

6838

6839 **2.28.6 Model**

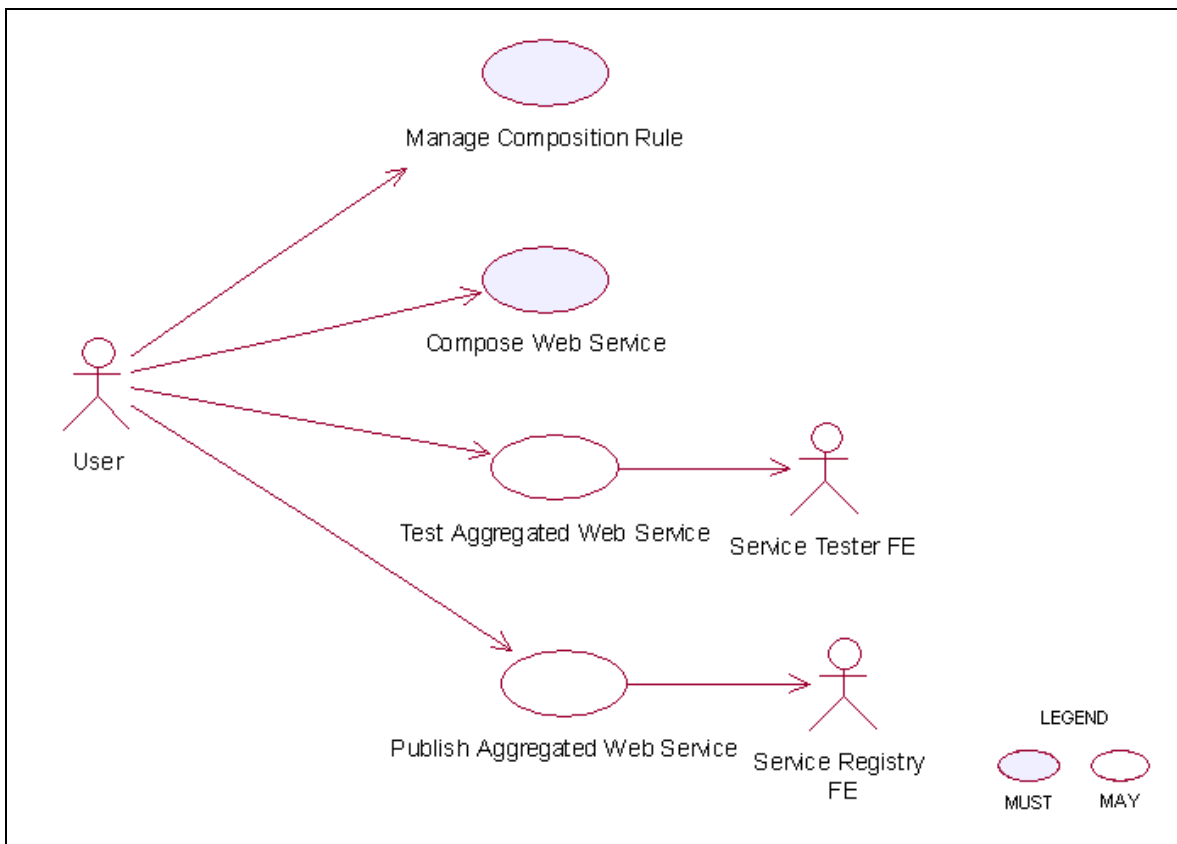


Figure 36: Model Of the Web Service Aggregation Functional Element [30]

6840

- 6841 **2.28.7 Usage Scenarios**
- 6842 **2.28.7.1 Manage composition rule**
- 6843 **2.28.7.1.1 Description**
- 6844 This use case allows the user to manage the composition rule used for Web Services
6845 aggregation.
- 6846 **2.28.7.1.2 Flow of Events**
- 6847 **2.28.7.1.2.1 Basic Flow**
- 6848 The use case begins when the user wants to manage a composition rule.
- 6849 1: The user sends a request to the Functional Element together with the composition rule and
6850 operation.
- 6851 2: Based on the operation it specified, one of the following sub-flows is executed:
- 6852 If the operation is '**Define a rule**', then sub-flow 2.1 is executed.
- 6853 If the operation is '**Update a rule**', then sub-flow 2.2 is executed.
- 6854 If the operation is '**Retrieve a rule**', then sub-flow 2.3 is executed.
- 6855 If the operation is '**Remove a rule**', then sub-flow 2.4 is executed.
- 6856 2.1: Define Rule.
- 6857 2.1.1: The Functional Element gets the composition rule, i.e. names of all Web Service,
6858 the sequence specification, parameters mapping between Web Services.
- 6859 2.1.2: The Functional Element verifies the correctness of composition rule.
- 6860 2.1.3: The Functional Element saves the composition rule to persistent mechanism.
- 6861 2.2: Update Rule.
- 6862 2.2.1: The Functional Element gets the name of composition rule.
- 6863 2.2.2: The Functional Element retrieves the composition rule definition from persistent
6864 mechanism.
- 6865 2.2.3: The Functional Element verifies the correctness of composition rule.
- 6866 2.2.4: The Functional Element updates the composition rule.
- 6867 2.3: Retrieve Rule.
- 6868 2.3.1: The Functional Element gets the name of composition rule.
- 6869 2.3.2: The Functional Element retrieves the definition of composition rule.
- 6870 2.3.3: The Functional Element returns the definition of rule.
- 6871 2.4: Remove Rule.
- 6872 2.4.1: The Functional Element gets the name of composition rule.

6873 2.4.2: The Functional Element checks whether the rule exists.

6874 2.4.3: The Functional Element removes the rule.

6875 3: The Functional Element returns the results to indicate the success or failure of this operation to
6876 the user and the use case ends.

6877 **2.28.7.1.2.2 Alternative Flows**

6878 1: Composition Rule Already Created.

6879 1.1: If in the basic flow 2.1.2, the same rule already created, Functional Element will return an
6880 error message to the user and the use case ends.

6881 2: Composition Rule Not Exist.

6882 2.1: If in the basic flow 2.2, 2.3, and 2.4 the specified rule does not exist, Functional Element
6883 will return an error message to the user and the use case ends.

6884 3: Persistency Mechanism Error.

6885 3.1: If in the basic flow 2.1, 2.2, 2.3, and 2.4, the Functional Element cannot perform data
6886 persistency, Functional Element will return an error message to the user and the use case
6887 ends.

6888 **2.28.7.1.3 Special Requirements**

6889 None.

6890 **2.28.7.1.4 Pre-Conditions**

6891 None.

6892 **2.28.7.1.5 Post-Conditions**

6893 None.

6894 **2.28.7.2 Compose Web Services**

6895 **2.28.7.2.1 Description**

6896 This use case will allow users to aggregate several simpler services into a higher-level service.

6897 **2.28.7.2.2 Flow of Events**

6898 **2.28.7.2.2.1 Basic Flow**

6899 This use case begins when any user wants to compose a Web Service.

6900 1: The user passes in a list of parameters for composition, including URLs of the WSDL,
6901 composition rules.

6902 2: Functional Element checks the signature of the Web Services to be composed via accessing
6903 WSDL.

6904 3: Functional Element generates interpretable XML-based script to encapsulate the composition
6905 logic.

6906 4: Functional Element returns the generated script and the use case ends.

6907 **2.28.7.2.2 Alternative Flows**

6908 1: Functional Element generates executable program and WSDL.

6909 1.1: At basic flow 3, Functional Element may transform the interpretable XML-based script
6910 into an executable program, if the user requested.

6911 1.2: At basic flow 3, Functional Element may generate WSDL for the executable program, if
6912 the user requested.

6913 1.3: Functional Element returns the code of executable program and WSDL file

6914 2: Functional Element detects ambiguity in Web Services signature.

6915 2.1: At basic flow 2, Functional Element encounters an ambiguity in the Web Services
6916 signature which it cannot resolve.

6917 2.2: Functional Element returns an error message that there is a composition error.

6918 3: Functional Element detects error in Web Services composition.

6919 3.1: At basic flow 3, Functional Element encounters an error in the Web Services
6920 composition.

6921 3.2: Functional Element returns an error message that there is a composition error.

6922 **2.28.7.2.3 Special Requirements**

6923 None.

6924 **2.28.7.2.4 Pre-Conditions**

6925 The composition rule for this Web Services aggregation must be pre-defined.

6926 **2.28.7.2.5 Post-Conditions**

6927 The generated program is ready for deployment in any Web Services container.

6928

6929 **2.28.7.3 Test Aggregated Web Services**

6930 **2.28.7.3.1 Description**

6931 This use case will allow users to test the functionality of aggregate web service.

6932 **2.28.7.3.2 Flow of Events**

6933 **2.28.7.3.2.1 Basic Flow**

6934 This use case begins when any user wants to test aggregated web service.

6935 1: The user passes in a list of parameters for testing, including URLs of the WSDL, values of
6936 parameters for invocation.

6937 2: Functional Element invokes the aggregated web service with parameters.

6938 3: Functional Element compares the returned parameter with the expected values.

6939 4: Functional Element returns the result of comparison and the use case ends.

6940 **2.28.7.3.2 Alternative Flows**

6941 1: Functional Element cannot invoke the aggregated web service.

6942 1.1: At basic flow 2, Functional Element encounters problems of invoking the aggregated web
6943 services.

6944 1.2: Functional Element returns an error message that indicates the invocation error.

6945 **2.28.7.3.3 Special Requirements**

6946 None.

6947 **2.28.7.3.4 Pre-Conditions**

6948 The executable program must be generated and deployed in web services hosting environment
6949 and ready for invocation.

6950 **2.28.7.3.5 Post-Conditions**

6951 None.

6952 **2.28.7.4 Publish Aggregated Web Services**

6953 **2.28.7.4.1 Description**

6954 This use case will allow users to publish the aggregated web services into UDDI registry.

6955 **2.28.7.4.2 Flow of Events**

6956 **2.28.7.4.2.1 Basic Flow**

6957 This use case begins when any user wants to publish the aggregated web services into UDDI
6958 registry.

6959 1: The user passes in a list of parameters for publishing, including URLs of the WSDL of
6960 aggregated web services, URL of UDDI and parameters of business and services description.

6961 2: Functional Element checks the availability of UDDI.

6962 3: Functional Element publishes services description of aggregated web services into UUDI.

6963 4: Functional Element returns the publish result and the use case ends.

6964 **2.28.7.4.2.2 Alternative Flows**

6965 1: UDDI registry server is not available

6966 1.1: At basic flow 2, Functional Element cannot connect to UDDI registry if UDDI registry
6967 server is not available.

6968 1.2: Functional Element returns the error message that UDDI connection cannot be built.

6969 2: Functional Element detects error in Web Services publishing.

6970 2.1: At basic flow 3, Functional Element encounters an error in the publishing Web Services.

6971 2.2: Functional Element returns an error message that there is a publishing error.

6972 **2.28.7.4.3 Special Requirements**

6973 None.

6974 **2.28.7.4.4 Pre-Conditions**

6975 The WSDL of the aggregated web services must exist.

6976 **2.28.7.4.5 Post-Conditions**

6977 None

6978

3 Functional Elements Usage Scenarios

6979

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:

6980

6981

- A service monitoring solution for the management of services in a SOA model

6982

- Enabling security through the Secure SOAP Functional Element

6983

- Decoupled User Access Management with support for multi-domain capabilities in a web service environment

6984

6985

- Single-Sign On for Distributed Services (Applications)

6986

6987

6988

6989 **3.1 Service Monitoring**

6990 In a SOA environment, management of services includes the capability to monitor services within
 6991 the management domain. These includes:

- 6992 • Monitoring the performance of services invoked
- 6993 • Generating audit trails of services invoked
- 6994 • Monitoring and testing the availability of services on the remote machine (server)

6995 A basic solution can be realised through the aggregation of two Functional Element, namely
 6996 Service Management and QoS, as shown in Figure 19. This solution can be improved with
 6997 notification capabilities, using the Notification Engine, be it to a remote client, a system
 6998 administrator or an end user of a particular service. Further enhancement can be added with a
 6999 Rule Engine that will have the cognitive ability to make decisions. An example of this
 7000 enhancement would be the ability to decide when should notifications or alerts be sent and in
 7001 what form.

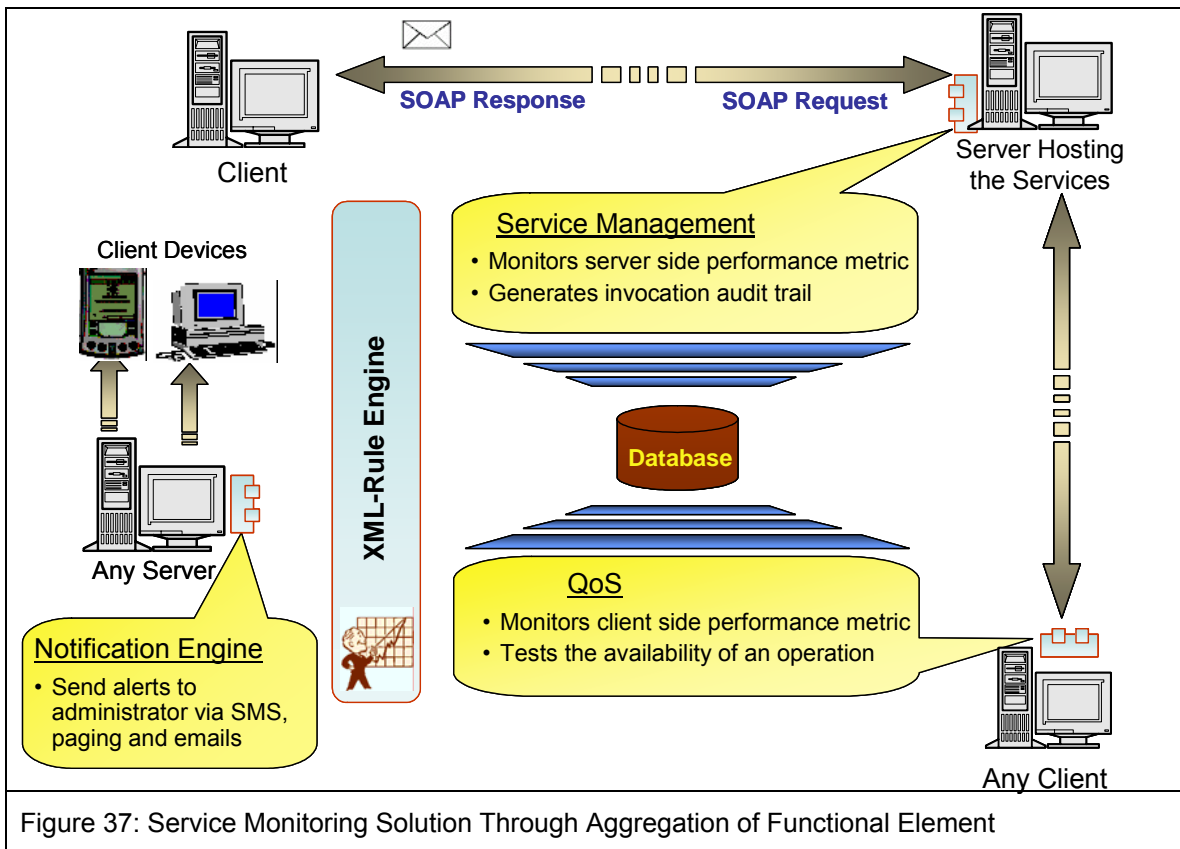


Figure 37: Service Monitoring Solution Through Aggregation of Functional Element

7002

7003 **3.2 Securing SOAP Messages**

7004 SOAP in its pure form does not have any built in security as it is meant to be a simple and
7005 lightweight protocol. As such, where security is needed, additional capabilities must be provided.
7006 Presently, standards like XML Encryption and XML Signature are available. Making use of these
7007 standards, the Secure Soap Functional Element, when deployed on both the sending and
7008 receiving parties, will be able to provide encryption and signing of messages as illustrated in
7009 Figure 20.
7010

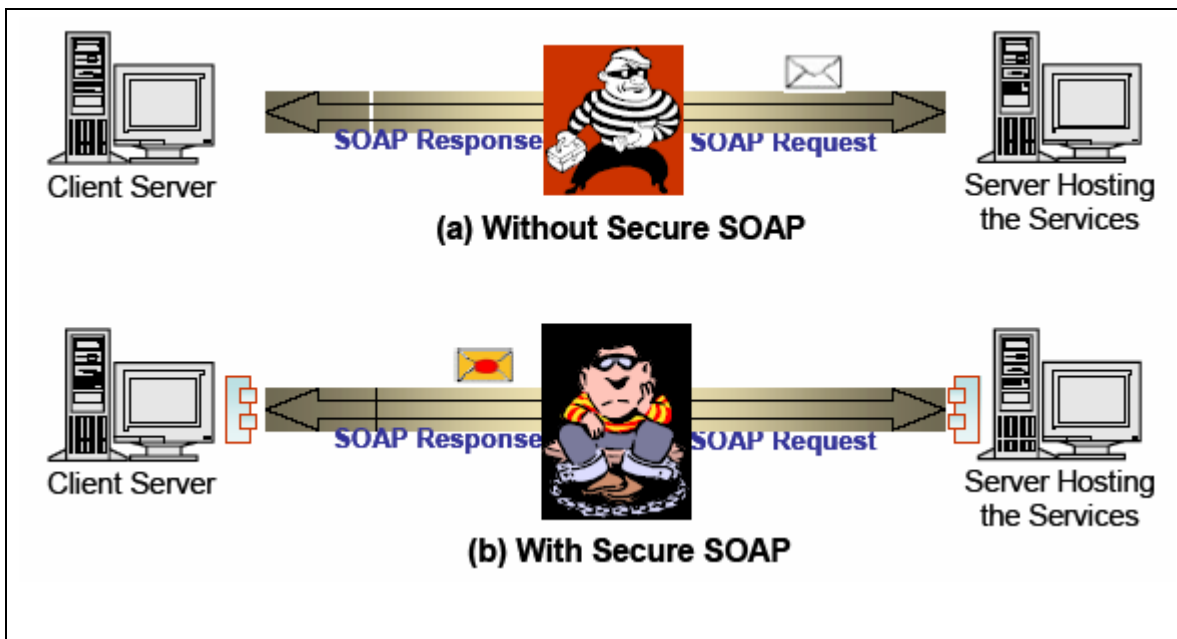


Figure 38: Securing SOAP Messages Using Secure SOAP Functional Element

7011

7012

3.3 Decoupled User Access Management

7013

User Access Management (UAM) has been implemented in many forms and in a wide variety of ways, from the most basic to the most complex. At the most simple form, the functionality would include username and password support. On the end of the scale, it would include functionalities like distributed access management, replication capabilities and fine-grain controls just to name a few.

7018

In this specification, the goal is to provide a set of Functional Element that can be used as building blocks for UAM, and can be extended when the need arises. It is provided as a decoupled building blocks consisting of four Functional Elements, namely User Management, Group Management, Role & Access Management and Phase & Lifecycle Management, as illustrated in Figure 21. These Functional Elements can be used in a variety of combinatorial forms, and some of these examples include:

7024

- User Management only, or

7025

- User Management and Group Management, or

7026

- User Management and Role & Access Management, or

7027

- User Management, Group Management and Role & Access Management, or

7028

- All the four Functional Elements in tandem

7029

On the same token, any of the Functional Elements can be replaced with similar functionality third party web services. As these services are designed to be in a web service environment, each of them also supports the concept of namespace. This namespace provision enables each of the Functional Elements to be used as web services that can be accessed by multiple organisations or to cater for users from different domains. With this, access control for example, can be defined for multiple domains without corruption or interferences problems.

7035

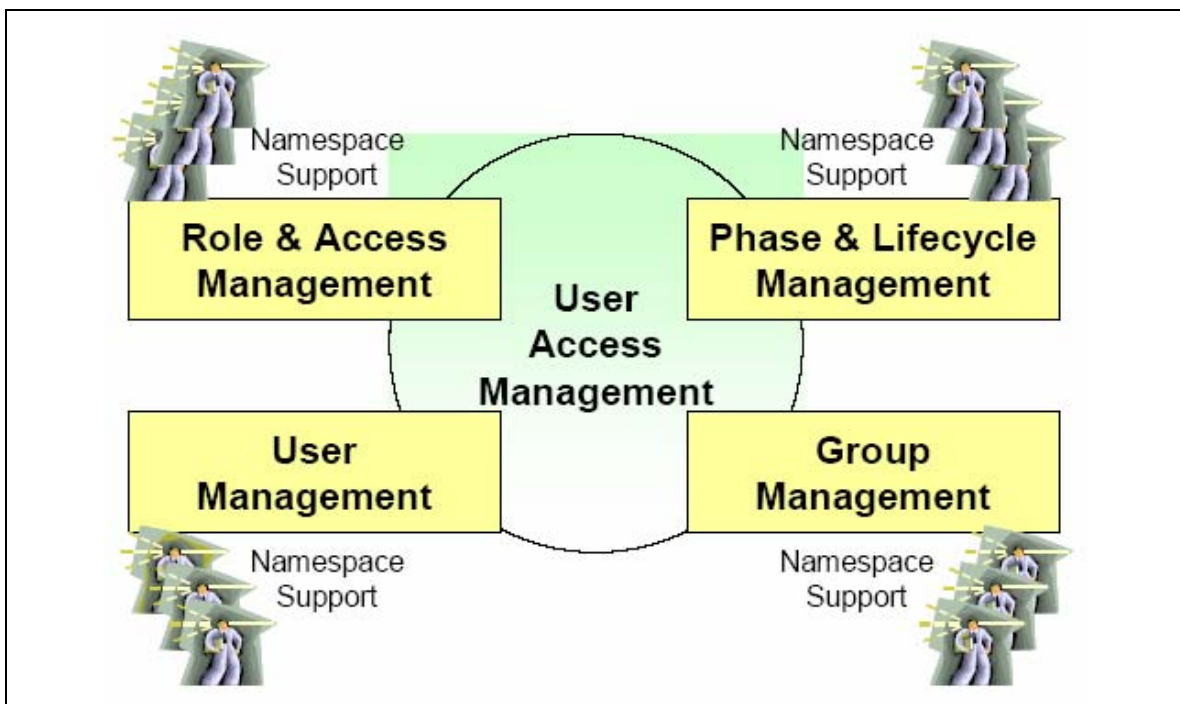


Figure 39: User Access Management via Functional Element

7036
7037

3.4 Single-Sign-On for Distributed Services (Applications)

7038 In a SOA world, it is very likely that services for a composite application can be potentially made
7039 up of multiple 3rd party services from different application domains. It is also very likely that each
7040 of these domains will require authentication of the user separately. However, it is not user friendly
7041 to enforce re-authentication as the user moves from one domain to another. Using the Identity
7042 Management Functional Element, with the potential combination of Secure SOAP Functional
7043 Element and other user access management Functional Elements like User Management, a
7044 solution for such an environment can be put together to enable Single-Sign-On. In this scenario of
7045 use, a Circle of Trust between different application domains can be established using the Identity
7046 Management Functional Element, and the exchanges between these domains can be secured
7047 using the Secure SOAP Functional Element. Access and authentication to individual domains
7048 remain the purview of the distributed applications, and can potentially also leveraged on the
7049 Decoupled User Access Management scenario detailed in section 3.3.
7050

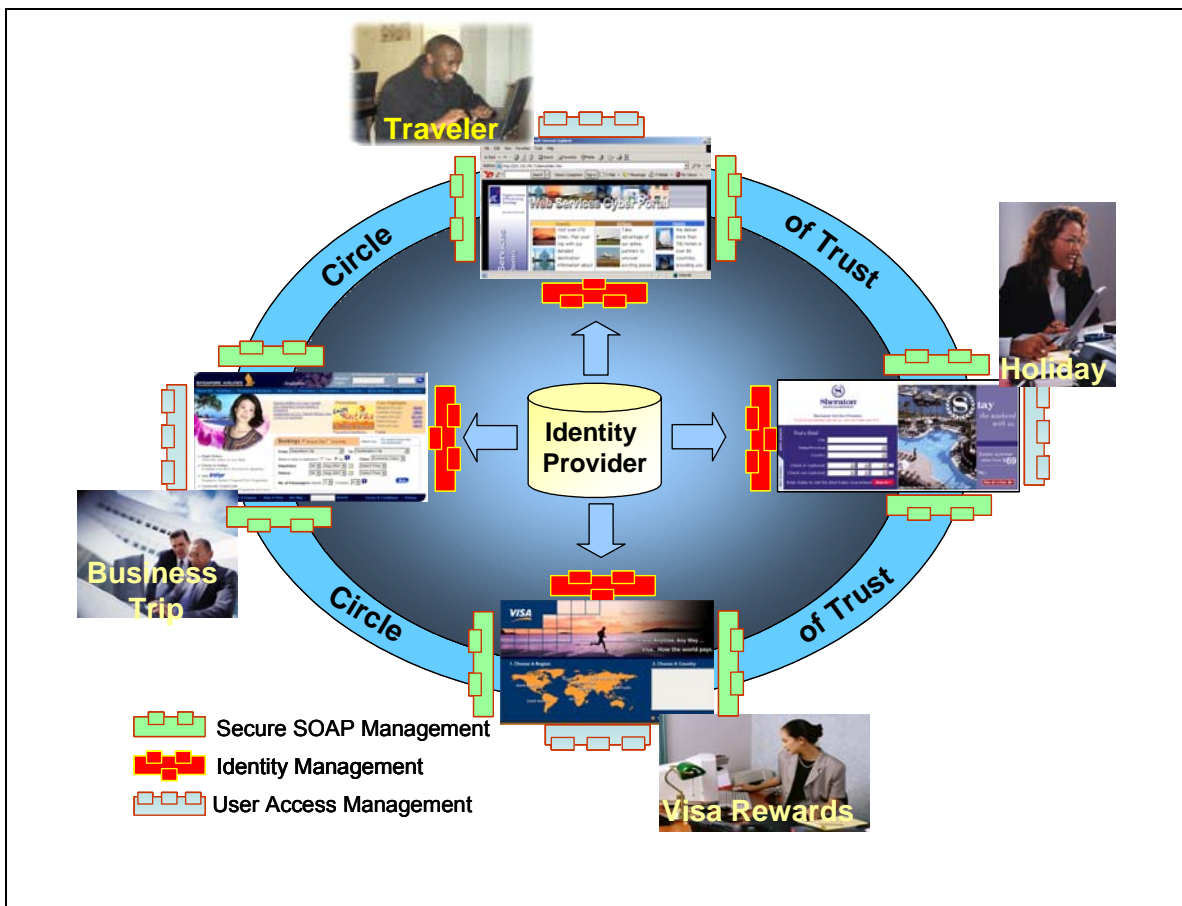


Figure 40: User Access Management via Functional Element

7051
7052
7053

4 References

1. FWSI TC, OASIS, **Web Service Implementation Methodology - Working Draft 0.1**, <http://www.oasis-open.org/apps/org/workgroup/fws/documents.php>, September 2004.
2. FWSI TC, OASIS, **Functional Elements Requirements - Approved Document 02**, <http://www.oasis-open.org/apps/org/workgroup/fws/documents.php>, October 2005.
3. S. Bradner, **Key words for use in RFCs to Indicate Requirement Levels**, 809, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
4. Cheng, Y.S., **WSRA Use Case Specifications - Event Handler**, version 1.0, JSSL of Singapore Institute of Manufacturing Technology, November 2003.
5. Wu, Y.Z., **WSRA Use Case Specifications – Group Management**, version 1.4, JSSL of Singapore Institute of Manufacturing Technology, September 2003.
6. OASIS Web Services Security TC, **Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)**, <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>, March 2004
7. OASIS, **Security Assertion Markup Language (SAML) v1.0**, <http://www.oasis-open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip>, September 2002.
8. Liberty Alliance, **ID-FF 1.2 Specifications**, version 1.2, http://www.projectliberty.org/specs/index.html#ID-FF_Specs.
9. Liberty Alliance, **ID-WSF 1.0 Specifications**, version 1.0, http://www.projectliberty.org/specs/index.html#ID-WSF_Specs.
10. **Web Services Federation Language (WS-Federation)**, <http://www-106.ibm.com/developerworks/webservices/library/ws-fed/>, July 2003.
11. Chan, L.P., **WSRA Use Case Specifications – Identity Management**, version 0.3, JSSL of Singapore Institute of Manufacturing Technology, December 2003.
12. Yin, Z.L., **WSRA Use Case Specifications – Log Utility**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
13. Limbu, D.K., **WSRA Use Case Specifications - Notification Engine**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
14. Wu, Y.Z., **WSRA Use Case Specifications - Phase & LC Management**, version 1.3, JSSL of Singapore Institute of Manufacturing Technology, October 2003.
15. Xu, X.J., **WSRA Use Case Specifications - Role & Access Management**, version 1.3, JSSL of Singapore Institute of Manufacturing Technology, September 2003.

-
16. Ramasamy, V., **WSRA Use Case Specifications - Search**, version 1.3, JSSL of Singapore Institute of Manufacturing Technology, June 2004.
 17. W3C, **XML-Signature Syntax and Processing**, W3C Recommendation, <http://www.w3.org/TR/xmlsig-core/>, February 2002.
 18. W3C, **XML-Encryption Syntax and Processing**, W3C Recommendation, <http://www.w3.org/TR/xmlenc-core/>, August 2002.
 19. Wu, Y.Z., **WSRA Use Case Specifications - Secure SOAP Management Private**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002
 20. Limbu, D.K., **WSRA Use Case Specifications - Sensory Engine**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
 21. Cheng, H.K., **WSRA Use Case Specifications - Service Management**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
 22. 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.
 23. 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.
 24. 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.
 25. 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.
 26. Ramasamy, V., **WSRA Use Case Specifications - Service Registry**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
 27. Yin Z.L., **WSRA Use Case Specifications – Service Router**, version 1.0, Web Services Programme of Singapore Institute of Manufacturing Technology, October 2004.
 28. Xu, X.J., **WSRA Use Case Specifications – User Management**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
 29. OASIS Web Services Business Process Execution Language TC, **Web Services Business Process Execution Language Specification**, Version 2.0, Committee Draft, <http://www.oasis-open.org/apps/org/workgroup/wsbpel/documents.php>, September 2002.
 30. Cheng, H.K., **WSRA Use Case Specifications – Web Service Aggregator**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.

7053 **Appendix A. Acknowledgments**

7054 Special thanks to the following individuals who contributed significantly towards this specification:

- 7055 • Ang Chai Hong
- 7056 • Chan Lai Peng
- 7057 • Cheng Yushi
- 7058 • Dilip Kumar Limbu
- 7059 • V. Ramasamy
- 7060 • Wu Yingzi
- 7061 • Xu Xingjian, and
- 7062 • Yin Zunliang.

7063 The committee would also like to express its appreciation for the encouragement and guidance
7064 provided by Jamie Clark throughout the course of the TC work.

7065

7066 The committee would also like to record its heartfelt appreciation to IBM Rational (Singapore) Pte.
7067 Ltd. for kindly agreeing to allow the use of the Rational Tools towards the creation of the Use
7068 Case Model used in this document.

7069

7070

7073

Appendix B. Revision History

7074

The following revision of this document represents the major milestones achieved.

7075

Rev	Date	By Whom	What
FWSI-FESC-specifications-01.doc	01-Jul-2004	Huang Kheng Cheng Puay Siew Tan	First Draft
FWSI-FESC-specifications-02.doc	18-Oct-2004	Huang Kheng Cheng Puay Siew Tan	Second Draft
fws-fe-1.0-guidelines-spec-wd-03.doc	25-Nov- 2004	Huang Kheng Cheng	Second Draft (Voted version)
fws-fe-1.0-guidelines-spec-cs-01.doc	04-Mar-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.
fws-fe-2.0-guidelines-spec-wd-01.doc	28-Oct-2005	Puay Siew Tan	<p>New working draft for Version 2.0 of the FE Specs:</p> <ul style="list-style-type: none"> • Deprecated 2 FEs, namely Presentation Transformer and Service Tester • Replaced the deprecated FEs with Transformer and Quality of Service (QoS) FEs respectively • Added 10 new FEs identified for version 2.0 • Minor changes to the following FEs: <ul style="list-style-type: none"> ○ Phase & Lifecycle Management ○ Secure SOAP Management ○ Sensory ○ Service Management ○ Service Registry ○ Web Service Aggregator • Usage Scenarios (added 1 more usage scenario for SSO)

7076

7077

Appendix C. Notices

7078 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
7079 that might be claimed to pertain to the implementation or use of the technology described in this
7080 document or the extent to which any license under such rights might or might not be available;
7081 neither does it represent that it has made any effort to identify any such rights. Information on
7082 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
7083 website. Copies of claims of rights made available for publication and any assurances of licenses
7084 to be made available, or the result of an attempt made to obtain a general license or permission
7085 for the use of such proprietary rights by implementors or users of this specification, can be
7086 obtained from the OASIS Executive Director.

7087 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
7088 applications, or other proprietary rights which may cover technology that may be required to
7089 implement this specification. Please address the information to the OASIS Executive Director.

7090 Copyright © OASIS Open 2004. All Rights Reserved.

7091 This document and translations of it may be copied and furnished to others, and derivative works
7092 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
7093 published and distributed, in whole or in part, without restriction of any kind, provided that the
7094 above copyright notice and this paragraph are included on all such copies and derivative works.
7095 However, this document itself does not be modified in any way, such as by removing the
7096 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
7097 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
7098 Property Rights document must be followed, or as required to translate it into languages other
7099 than English.

7100 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
7101 successors or assigns.

7102 This document and the information contained herein is provided on an "AS IS" basis and OASIS
7103 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
7104 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
7105 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
7106 PARTICULAR PURPOSE.