



# Functional Elements Specification

## Committee Draft 2.0, 05-January-2006

### Document identifier:

fws-fe-2.0-guidelines-spec-cd-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](mailto:pstan@simtech.a-star.edu.sg))

### Contributor(s):

Andy Tan, Individual ([andytan@intrinix.net](mailto:andytan@intrinix.net))

Shawn Cheng Hua-Shan, XMLBoss ([shawn@xmlboss.net](mailto:shawn@xmlboss.net))

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

Viren Baraiya, Crimson Logic Pte Ltd ([viren@crimsonlogic.com](mailto:viren@crimsonlogic.com))

Jagdeep Talla, Crimson Logic Pte Ltd ([jagdeep@crimsonlogic.com](mailto:jagdeep@crimsonlogic.com))

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

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

V.Ramasamy, SIMTech ([rama@simtech.a-star.edu.sg](mailto:rama@simtech.a-star.edu.sg))

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

Lee Ah Kim, SIMTech ([aklee@simtech.a-star.edu.sg](mailto: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](mailto:fwsifesc@lists.oasis-open.org) list. Others should subscribe to and send comments to the  
43 [fwsicomment@lists.oasis-open.org](mailto:fwsicomment@lists.oasis-open.org) list. To subscribe, send an email message to [fwsicomment-request@lists.oasis-open.org](mailto:fwsicomment-request@lists.oasis-open.org)  
44 with the word "subscribe" as the body of the  
45 message.  
46

47 For information on whether any patents<sup>1</sup> 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-](http://www.oasis-open.org/committees/fws/)  
50 [open.org/committees/fws/](http://www.oasis-open.org/committees/fws/)).

---

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

---

## 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 Functional Element (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 .....	65
85	2.4.4	Interdependency.....	66
86	2.4.5	Related Technologies and Standards .....	66
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> .....	90
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	<i>Terms Used</i> .....	137
148	2.12.3	<i>Key Features</i> .....	138
149	2.12.4	<i>Interdependency</i> .....	139
150	2.12.5	<i>Related Technologies and Standards</i> .....	139
151	2.12.6	<i>Model</i> .....	140
152	2.12.7	<i>Usage Scenarios</i> .....	140
153	2.13	<i>Policy Enforcement Functional Element (new)</i> .....	145
154	2.13.1	<i>Motivation</i> .....	145
155	2.13.2	<i>Terms Used</i> .....	145
156	2.13.3	<i>Key Features</i> .....	145
157	2.13.4	<i>Interdependency</i> .....	146
158	2.13.5	<i>Related Technologies and Standards</i> .....	146
159	2.13.6	<i>Model</i> .....	146
160	2.13.7	<i>Usage Scenarios</i> .....	147
161	2.14	<i>Presentation Transformer Functional Element (Deprecated)</i> .....	149
162	2.15	<i>QoS Functional Element (new)</i> .....	150
163	2.15.1	<i>Motivation</i> .....	150
164	2.15.2	<i>Terms Used</i> .....	150
165	2.15.3	<i>Key Features</i> .....	151
166	2.15.4	<i>Interdependencies</i> .....	151
167	2.15.5	<i>Related Technologies and Standards</i> .....	152
168	2.15.6	<i>Model</i> .....	152
169	2.15.7	<i>Usage Scenarios</i> .....	153
170	2.16	<i>Role and Access Management Functional Element</i> .....	162
171	2.16.1	<i>Motivation</i> .....	162
172	2.16.2	<i>Terms Used</i> .....	162
173	2.16.3	<i>Key Features</i> .....	163
174	2.16.4	<i>Interdependencies</i> .....	164
175	2.16.5	<i>Related Technologies and Standards</i> .....	164
176	2.16.6	<i>Model</i> .....	165
177	2.16.7	<i>Usage Scenario</i> .....	165
178	2.17	<i>Search Functional Element</i> .....	175
179	2.17.1	<i>Motivation</i> .....	175
180	2.17.2	<i>Terms Used</i> .....	175
181	2.17.3	<i>Key Features</i> .....	176
182	2.17.4	<i>Interdependencies</i> .....	176
183	2.17.5	<i>Related Technologies and Standards</i> .....	176
184	2.17.6	<i>Model</i> .....	177
185	2.17.7	<i>Usage Scenario</i> .....	177
186	2.18	<i>Secure SOAP Management Functional Element</i> .....	180
187	2.18.1	<i>Motivation</i> .....	180
188	2.18.2	<i>Terms Used</i> .....	180
189	2.18.3	<i>Key Features</i> .....	181
190	2.18.4	<i>Interdependencies</i> .....	181
191	2.18.5	<i>Related Technologies and Standards</i> .....	181
192	2.18.6	<i>Model</i> .....	182
193	2.18.7	<i>Usage Scenarios</i> .....	182
194	2.19	<i>Sensory Functional Element</i> .....	186
195	2.19.1	<i>Motivation</i> .....	186
196	2.19.2	<i>Terms Used</i> .....	186

197	2.19.3	Key Features.....	186
198	2.19.4	Interdependencies.....	187
199	2.19.5	Related Technologies and Standards .....	187
200	2.19.6	Model .....	187
201	2.19.7	Usage Scenarios.....	187
202	2.20	Service Level Management Functional Element (new) .....	190
203	2.20.1	Motivation.....	190
204	2.20.2	Terms Used.....	190
205	2.20.3	Key Features.....	190
206	2.20.4	Interdependencies.....	191
207	2.20.5	Related Technologies and Standards .....	191
208	2.20.6	Model .....	191
209	2.20.7	Usage Scenarios.....	192
210	2.21	Service Level Enforcement Functional Element (new) .....	198
211	2.21.1	Motivation.....	198
212	2.21.2	Terms Used.....	198
213	2.21.3	Key Features.....	198
214	2.21.4	Interdependencies.....	198
215	2.21.5	Related Technologies and Standards .....	199
216	2.21.6	Model .....	199
217	2.21.7	Usage Scenarios.....	199
218	2.22	Service Management Functional Element.....	204
219	2.22.1	Motivation.....	204
220	2.22.2	Terms Used.....	204
221	2.22.3	Key Features.....	204
222	2.22.4	Interdependencies.....	205
223	2.22.5	Related Technologies and Standards .....	205
224	2.22.6	Model .....	206
225	2.22.7	Usage Scenarios.....	206
226	2.23	Service Registry Functional Element.....	212
227	2.23.1	Motivation.....	212
228	2.23.2	Terms Used.....	212
229	2.23.3	Key Features.....	213
230	2.23.4	Interdependencies.....	213
231	2.23.5	Related Technologies and Standards .....	213
232	2.23.6	Model .....	214
233	2.23.7	Usage Scenario.....	214
234	2.24	Service Router Functional Element (new) .....	223
235	2.24.1	Motivation.....	223
236	2.24.2	Terms Used.....	223
237	2.24.3	Key Features.....	224
238	2.24.4	Interdependencies.....	225
239	2.24.5	Related Technologies and Standards .....	225
240	2.24.6	Model .....	226
241	2.24.7	Usage Scenarios.....	226
242	2.25	Service Tester Functional Element (Deprecated).....	234
243	2.26	Transformer Functional Element (new) .....	235
244	2.26.1	Motivation.....	235
245	2.26.2	Terms Used.....	235
246	2.26.3	Key Features.....	236

247	2.26.4	<i>Interdependencies</i> .....	236
248	2.26.5	<i>Related Technologies and Standards</i> .....	237
249	2.26.6	<i>Model</i> .....	237
250	2.26.7	<i>Usage Scenarios</i> .....	237
251	2.27	<i>User Management Functional Element</i> .....	243
252	2.27.1	<i>Motivation</i> .....	243
253	2.27.2	<i>Terms Used</i> .....	243
254	2.27.3	<i>Key Features</i> .....	244
255	2.27.4	<i>Interdependencies</i> .....	245
256	2.27.5	<i>Related Technologies and Standards</i> .....	245
257	2.27.6	<i>Model</i> .....	245
258	2.27.7	<i>Usage Scenarios</i> .....	246
259	2.28	<i>Web Service Aggregator Functional Element</i> .....	253
260	2.28.1	<i>Motivation</i> .....	253
261	2.28.2	<i>Terms Used</i> .....	253
262	2.28.3	<i>Key Features</i> .....	254
263	2.28.4	<i>Interdependencies</i> .....	255
264	2.28.5	<i>Related Technologies and Standards</i> .....	255
265	2.28.6	<i>Model</i> .....	255
266	2.28.7	<i>Usage Scenarios</i> .....	256
267	3	<i>Functional Elements Usage Scenarios</i> .....	261
268	3.1	<i>Service Monitoring</i> .....	262
269	3.2	<i>Securing SOAP Messages</i> .....	263
270	3.3	<i>Decoupled User Access Management</i> .....	264
271	3.4	<i>Single-Sign-On for Distributed Services (Applications)</i> .....	265
272	4	<i>References</i> .....	266
273		<i>Appendix A. Acknowledgments</i> .....	268
274		<i>Appendix B. Revision History</i> .....	269
275		<i>Appendix C. Notices</i> .....	271
276			
277			

---

# 1 Introduction

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

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

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

## 1.1 Document Outline

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

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

- Motivation

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



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

## 1.2 Motivation

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

## 1.3 Terminology

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

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

---

## 2 List of Functional Elements

### 2.1 Data Integrator Functional Element (new)

#### 2.1.1 Motivation

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

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

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

Primary Requirements

1.1 PROCESS-220 to PROCESS-236.

Secondary Requirements

1.2 None

#### 2.1.2 Terms Used

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.

Data Transformation Rule	<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> <ul style="list-style-type: none"> <li>–The first type is the one that applies at the logical data view level and generates instances of data for the whole data view. <ul style="list-style-type: none"> <li>» 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.</li> </ul> </li> <li>–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. <ul style="list-style-type: none"> <li>» 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 .</li> </ul> </li> </ul>
Logical Data View	<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

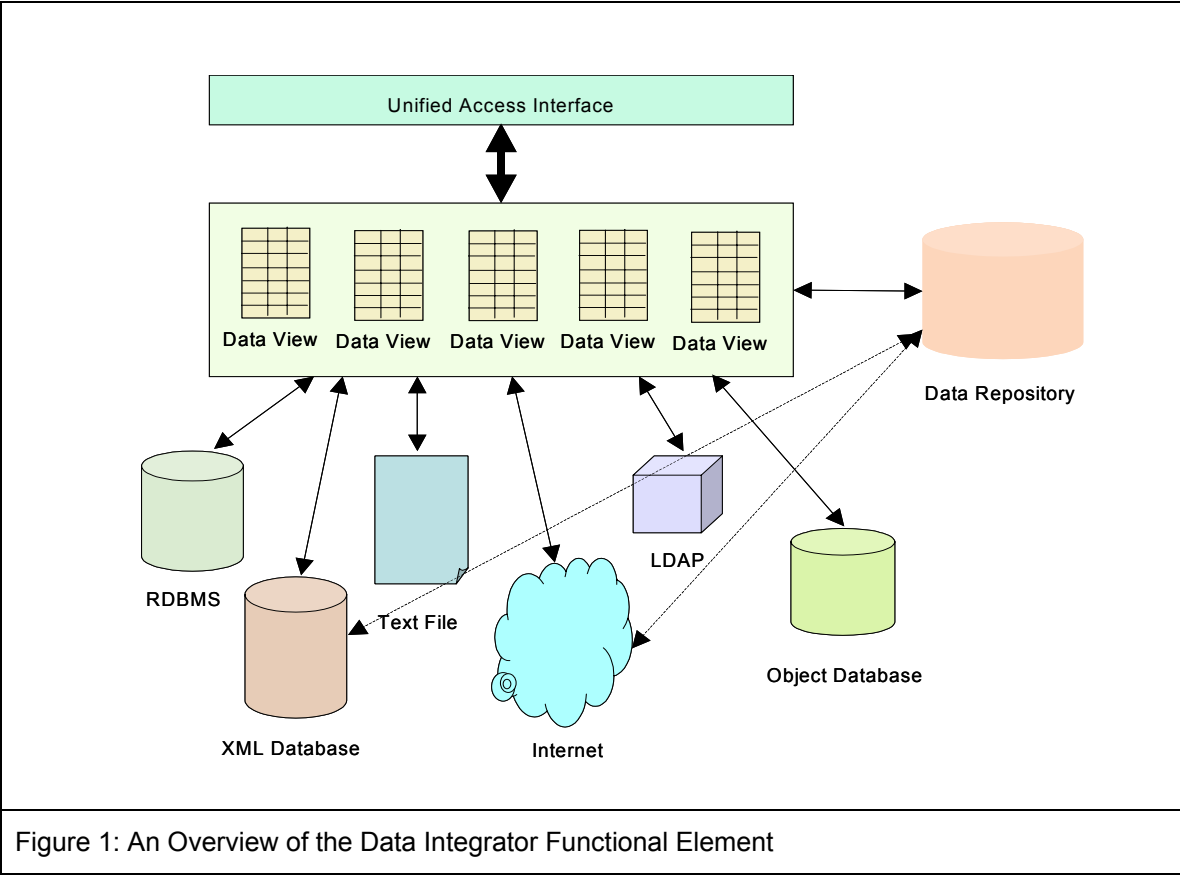


Figure 1: An Overview of the Data Integrator Functional Element

383

384

385

386

387

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

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

### 2.1.3 Key Features

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

1. The Functional Element MUST provide the capability to manage the available data sources. This includes capability to:
  - 1.1. Add new data source to the pool of available data sources.
  - 1.2. Remove data source from the pool of available data sources.
2. The Functional Element MUST provide the capability to define a logical data view based on the pool of available data sources.
3. The Functional Element MUST provide capability to manage the updating and deletion of a logical data view.
4. The Functional Element MUST provide capability to manage the creation, updating and deletion of data transformation rules.
5. The Functional Element MUST provide capability to retrieve data based on the logical data view defined.
6. The Functional Element MUST provide a unified way to query data based on defined logical data views.
7. The Functional Element MUST provide a mechanism to extract data from various data sources and transform the data according to defined transformation rules for a logical data view.

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

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

### 2.1.4 Interdependencies

None

2.1.5 Related Technologies and Standards

RDBMS, LDAP, XML Database

2.1.6 Model

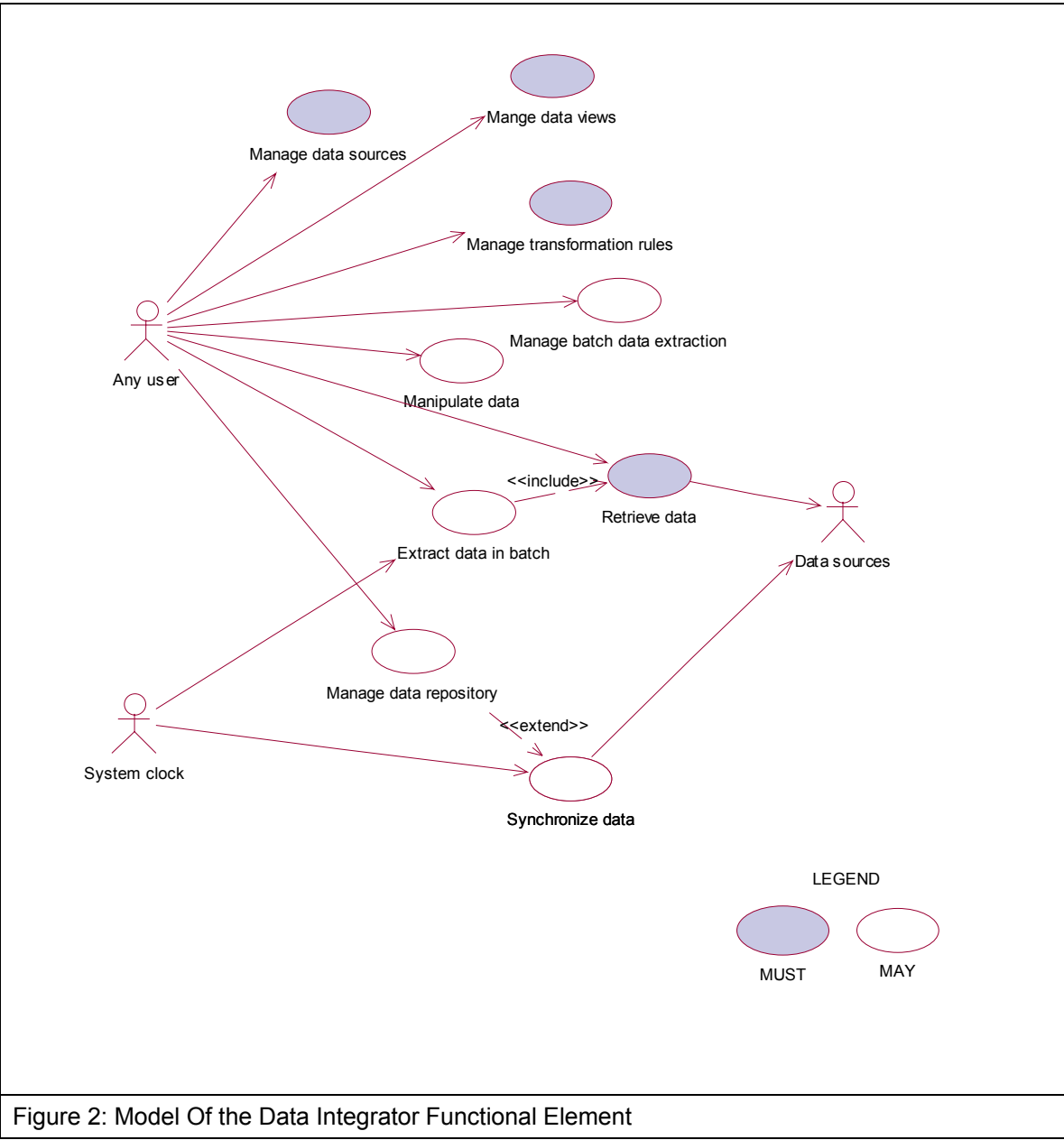


Figure 2: Model Of the Data Integrator Functional Element

## 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 (with the assumption that the user has a valid login Id and password).

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       **2.1.7.2.2.2 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 **2.1.7.3.2.2 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     **2.1.7.5.2.2 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 **2.1.7.6.2.2 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

## 2.2 Error Management Functional Element (new)

### 2.2.1 Motivation

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

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

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

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

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

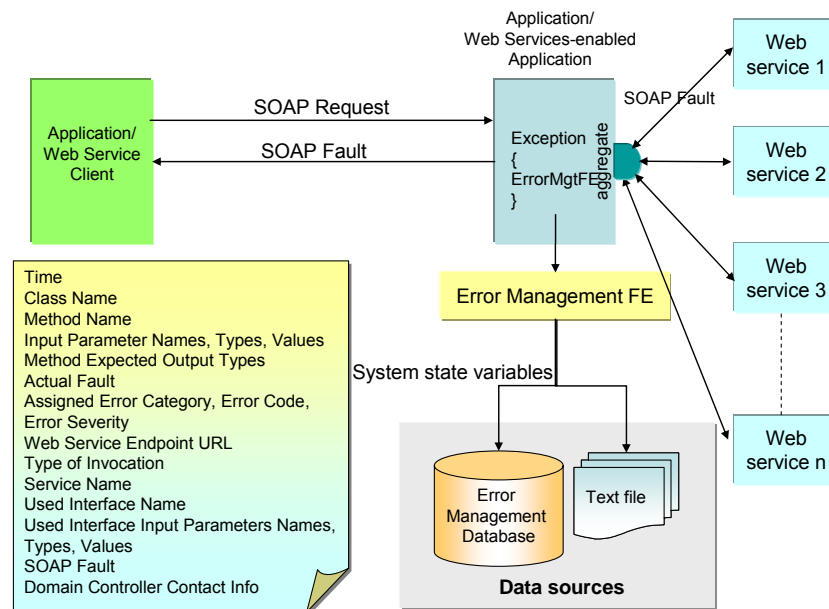


Figure 3: Error Management Functional Element Usage

This Functional Element fulfills the following requirements from the Functional Elements Requirements:

Primary Requirements

- MANAGEMENT-340 to MANAGEMENT-346

## 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.

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

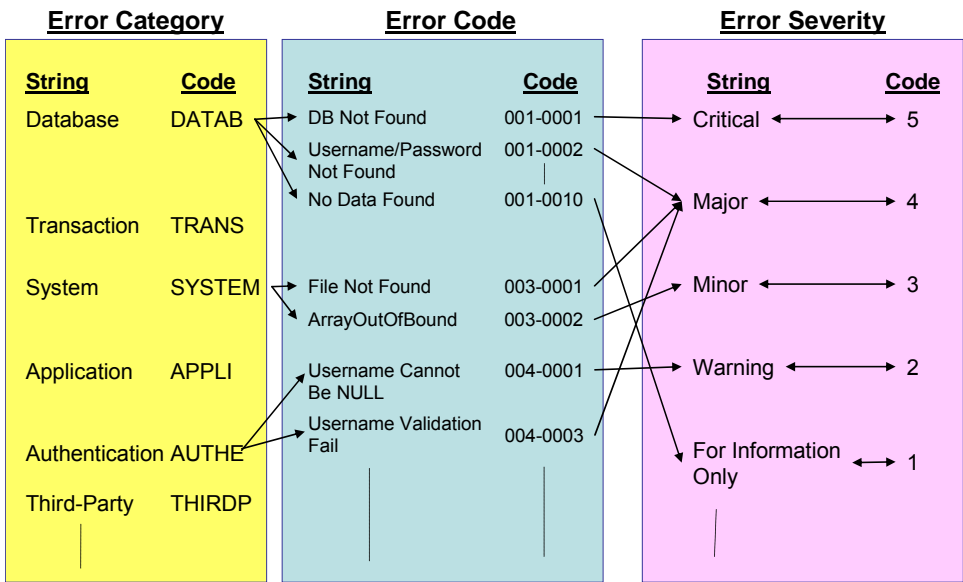


Figure 4: An Example of Error Hierarchy

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

### 2.2.3 Key Features

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

1. The Functional Element MUST provide the ability to create new Error Category
2. The Functional Element MUST provide the ability to modify and delete defined Error Category.

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

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

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

## 2.2.4 Interdependencies

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

## 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	<b>XML Schema Part 0: Primer Second Edition</b> W3C Recommendation 28 October 2004 <b>XML Schema Part 1: Structures Second Edition</b> W3C Recommendation 28 October 2004 <b>XML Schema Part 2: Datatypes Second Edition</b> 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

933 **2.2.6 Model**

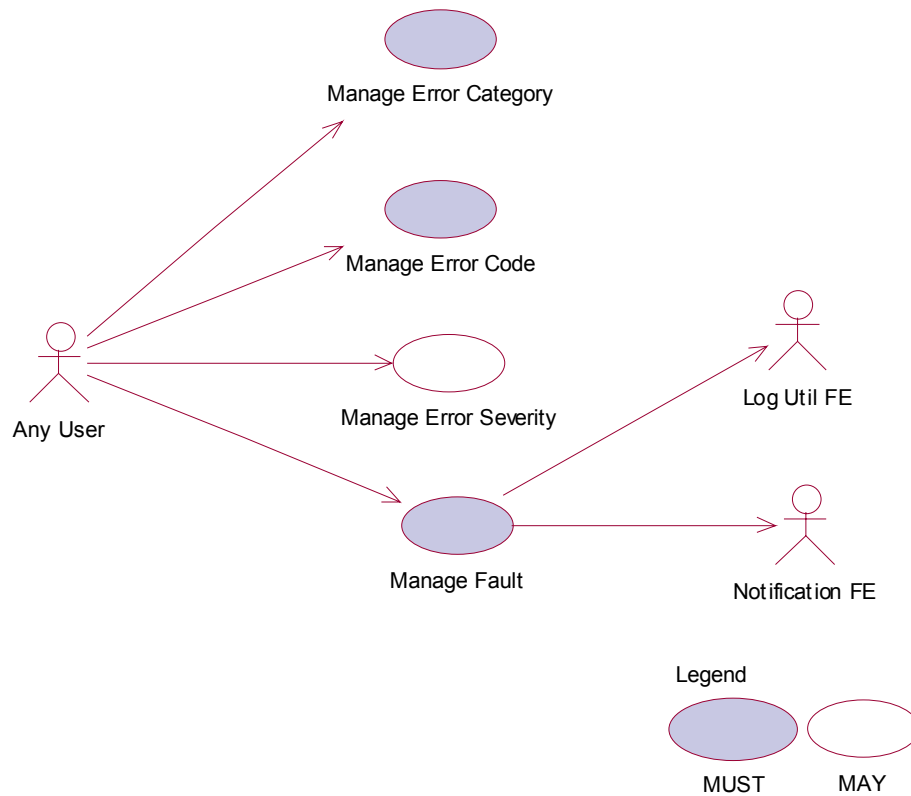


Figure 5: Model Of the Error Management Functional Element

934

935 **2.2.7 Usage Scenarios**

936 **2.2.7.1 Manage Error Category**

937 **2.2.7.1.1 Description**

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

939 **2.2.7.1.2 Flow of Events**

940 **2.2.7.1.2.1 Basic Flow**

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

942 1: The user sends a request to manipulate an Error Category.

943 2: Based on the operation it specifies, one of the following sub-flows is executed:

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

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

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

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

948 2.1: Create Error Category.

949 2.1.1: The Functional Element gets category definition.

950 2.1.2: The Functional Element checks whether the category exists.

951 2.1.3: The Functional Element creates the category and save it in the error database.

952 2.2: Retrieve Error Category.

953 2.2.1: The Functional Element gets the Error Category name.

954 2.2.2: The Functional Element checks whether the category exists.

955 2.2.3: The Functional Element retrieves the Error Category's information from the Error

956 Management Data sources.

957 2.3: Update Error Category.

958 2.3.1: The Functional Element gets the Error Category name.

959 2.3.2: The Functional Element checks whether the Error Category exists.

960 2.3.3: The Functional Element updates the category definition and save it in the Error

961 Management Data sources.

962 2.4: Delete Error Category.

963 2.4.1: The Functional Element gets the Error Category name.

964 2.4.2: The Functional Element checks whether the Error Category name exists.

965 2.4.3: The Functional Element checks whether the Error Code associated to the Error

966 Category name exists.

967 • If Error Codes associated to the Error Category name exists, then basic sub-flow

968 2.4.4 is executed.

969 • If Error Codes associated to the Error Category name does not exist, then the

970 basic sub-flow 2.4.7 is executed.

971 2.4.4: Error Codes associated to the Error Category name exists.

972 • If the Error Severity associated to the respective Error Codes exists, the basic

973 sub-flow 2.4.5 is executed.

974 • If the Error Severity associated to the respective Error Code does not exist, then

975 the basic sub-slow 2.4.6 is executed.

976 2.4.5: The Error Severity Exist.

977 2.4.5.1: The Functional Element removes the error severities associated to the

978 respective Error Code from sub-flow 2.4.4.

979 2.4.6: The Error Severity Does Not Exist.



980 2.4.6.1 The Functional Element removes the respective Error Codes from sub-flow  
981 2.4.3 from the Error Management Data sources.

982 2.4.7: The Error Codes Associated to the Error Category Name Does Not Exist.

983 2.4.7.1: The Functional Element removes the respective Error Codes associated to  
984 the Error Category name (from sub-flow 2.4.3) from the Error Management Data  
985 sources.

986 2.4.8: The Functional Element removes the Error Category name from the Error  
987 Management Data sources.

988 3: The Functional Element returns the results of the operation to the end user and the use case  
989 ends.

990 **2.2.7.1.2.2 Alternative Flows**

991 1: Error Category Already Exists.

992 1.1: If in the basic flow 2.1.2, the error category is already defined, the Functional Element  
993 writes the system state variables into the Error Management Data Sources using Log Utility  
994 Functional Element and notifies the system domain controller using the Notification  
995 Functional Element and the use case ends.

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

1000

1001 2: Error Category Not Found.

1002 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  
1003 Element writes the system state variables into the Error Management Data Sources using  
1004 Log Utility Functional Element and notifies the system domain controller using the Notification  
1005 Functional Element and the use case ends.

1006 **2.2.7.1.3 Special Requirements**

1007 None.

1008 **2.2.7.1.4 Pre-Conditions**

1009 None.

1010 **2.2.7.1.5 Post-Conditions**

1011 Once the Error Category is deleted, all the associated Error Code and its Error Severity will be  
1012 removed.

1013

1014 **2.2.7.2 Manage Error Code**

1015 **2.2.7.2.1 Description**

1016 This use case allows the user to manage Error Code.

1017     **2.2.7.2.2     Flow of Events**

1018     **2.2.7.2.2.1 Basic Flow**

1019     The use case begins when the user wants to create/retrieve/update/delete an error code  
1020     associated to an error category.

1021     1: The user sends a request to manipulate an error code.

1022     2: Based on the operation it specifies, one of the following sub-flows is executed:

1023     If the operation is '**Create Error Code**', the sub-flow 2.1 is executed.

1024     If the operation is '**Retrieve Error Code**', the sub-flow 2.2 is executed.

1025     If the operation is '**Update Error Code**', the sub-flow 2.3 is executed.

1026     If the operation is '**Delete Error Code**', the sub-flow 2.4 is executed.

1027         2.1: Create Error Code.

1028             2.1.1: The Functional Element gets the Error Category name

1029             2.1.2. The Functional Element gets Error Code definition for the Error Category.

1030             2.1.3: The Functional Element checks whether the Error Code exists.

1031             2.1.4: The Functional Element creates the Error Code for the Error Category name and  
1032             saves it into the Fault Management database.

1033         2.2: Retrieve Error Code.

1034             2.2.1: The Functional Element gets the Error Category name

1035             2.2.2. The Functional Element gets the Error Code name.

1036             2.2.3: The Functional Element checks whether the Error Code exists.

1037             2.2.4. The Functional Element retrieves the Error Code's information from the error  
1038             database.

1039         2.3: Update Error Code.

1040             2.3.1: The Functional Element gets the Error Category name.

1041             2.3.2. The Functional Element gets the Error Code name.

1042             2.3.3: The Functional Element checks whether the Error Code exists.

1043             2.3.4: The Functional Element updates the error code definition associated to the Error  
1044             Category and save it in the Error Management Data sources.

1045         2.4: Delete Error Code.

1046             2.4.1: The Functional Element gets the Error Category name.

1047             2.4.2. The Functional Element gets the Error Code name.

1048             2.4.3: The Functional Element checks whether the Error Code exists.

1049 2.4.4: The Functional Element checks whether the Error Severity associated to the Error  
1050 Category and Error Code exists. Depending on whether the Error Severity exists, one of  
1051 the following sub-flows will be executed.

- 1052 • If the Error Severity exists, then basic sub-flow 2.4.5 is executed.
- 1053 • If the Error Severity does not exist, the basic sub-flow 2.4.6 is executed.

1054 2.4.5: Error Severity Exists.

1055 2.4.5.1: The Functional Element removes the Error Severity associated to the Error  
1056 Category and Error Code from the Error Management Data sources.

1057 2.4.5.2: The Functional Element removes the Error Code associated to the Error  
1058 Category name from the Error Management Data sources.

1059 2.4.6: Error Severity Does Not Exist

1060 2.4.6.1: The Functional Element removes the Error Code associated to the Error  
1061 Category name from the Error Management Data sources.

1062 3: The Functional Element returns the results of the operation to the end user and the use case  
1063 ends.

#### 1064 **2.2.7.2.2.2 Alternative Flows**

1065 1: Error Code Already Exists.

1066 1.1: If in the basic flow 2.1.3, the Error Code associated to the Error Category name is  
1067 already defined, the Functional Element writes the system state variables into the Error  
1068 Management Data sources using the Log Utility Functional Element and notifies the system  
1069 domain controller using the Notification Functional Element and the use case ends.

1070 2: Error Code Not Found.

1071 2.1: If in the basic flows 2.2.3, 2.3.3 and 2.4.3 the Error Code associated to the Error  
1072 Category name does not exist, the Functional Element writes the system state variables into  
1073 the Error Management Data sources using the Log Utility Functional Element and notifies the  
1074 system domain controller using the Notification Functional Element and the use case ends.

#### 1075 **2.2.7.2.3 Special Requirements**

1076 None.

#### 1077 **2.2.7.2.4 Pre-Conditions**

1078 None.

#### 1079 **2.2.7.2.5 Post-Conditions**

1080 None.

1081

### 1082    **2.2.7.3 Manage Error Severity**

#### 1083    **2.2.7.3.1 Description**

1084    This use case allows the user to manage error severity.

#### 1085    **2.2.7.3.2 Flow of Events**

##### 1086    **2.2.7.3.2.1 Basic Flow**

1087    The use case begins when the user wants to create/retrieve/update/delete an Error Severity  
1088    associated to an Error Category and Error Code.

1089    1: The user sends a request to manipulate an error severity.

1090    2: Based on the operation it specifies, one of the following sub-flows is executed:

1091    If the operation is '**Create Error Severity**', the sub-flow 2.1 is executed.

1092    If the operation is '**Retrieve Error Severity**', the sub-flow 2.2 is executed.

1093    If the operation is '**Update Error Severity**', the sub-flow 2.3 is executed.

1094        1. If the operation is '**Delete Error Severity**', the sub-flow 2.4 is executed

1095        2.1: Create Error Severity.

1096            2.1.1: The Functional Element gets Error Category name.

1097            2.1.2: The Functional Element gets Error Code name.

1098            2.1.3: The Functional Element gets Error Severity definition.

1099            2.1.4: The Functional Element checks whether the Error Severity associated to the Error  
1100            Category and error Code name exists.

1101            2.1.5: The Functional Element creates the Error Severity associated to the Error  
1102            Category name and Error Code name and saves it into the Error Management Data  
1103            sources.

1104        2.2 Retrieve Error Severity.

1105            2.2.1: The Functional Element gets the Error Category name.

1106            2.2.2: The Functional Element gets the Error Code name.

1107            2.2.3. The Functional Element gets the Error Severity name.

1108            2.2.4: The Functional Element checks whether the Error Severity exists associated to the  
1109            Error Category and Error Code names.

1110            2.2.5. The Functional Element retrieves the Error Severity's information associated to the  
1111            Error Category and Error Code names from the Error Management Data sources.

1112        2.3: Update Error Severity.

1113            2.3.1: The Functional Element gets the Error Category name.

1114            2.3.2: The Functional Element gets the Error Code name.

1115 2.3.3. The Functional Element gets the Error Severity name.

1116 2.3.4: The Functional Element checks whether the Error Severity exists associated to the

1117 Error Category and Error Code names.

1118 2.3.5: The Functional Element updates the Error Severity definition associated to the

1119 Error Category and Error Code names and saves it into the Error Management Data

1120 sources.

1121 2.4: Delete Error Severity.

1122 2.4.1: The Functional Element gets the Error Category name.

1123 2.4.2: The Functional Element gets the Error Code name.

1124 2.4.3. The Functional Element gets the Error Severity name.

1125 2.4.4: The Functional Element checks whether the Error Severity exists associated to the

1126 Error Category and Error Code names.

1127 2.4.5: The Functional Element removes the Error Severity associated to the Error

1128 Category and Error Code names from the Error Management Data sources.

1129 3: The Functional Element returns the results of the operation to the end user and the use case

1130 ends.

1131 **2.2.7.3.2.2 Alternative Flows**

1132 1: Error Severity Already Exists.

1133 1.1: If in the basic flow 2.1.4, the Error Severity associated to the Error Category and Error

1134 Code names is already defined, the Functional Element writes the system state variables into

1135 the Error Management Data sources using the Log Utility Functional Element and notifies the

1136 system domain controller using the Notification Functional Element and the use case ends.

1137 2: Error Severity Not Found.

1138 2.1: If in the basic flows 2.2.4, 2.3.4 and 2.4.4, the Error Severity associated to the Error

1139 Category and Error Code names does not exist, the Functional Element writes the system

1140 state variables into the Error Management Data sources using the Log Utility Functional

1141 Element and notifies the system domain controller using the Notification Functional Element

1142 and the use case ends.

1143 **2.2.7.3.3 Special Requirements**

1144 None

1145 **2.2.7.3.4 Pre-Conditions**

1146 None

1147 **2.2.7.3.5 Post-Conditions**

1148 None

## 2.2.7.4 Manage Fault

### 2.2.7.4.1 Description

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

### 2.2.7.4.2 Flow of Events

#### 2.2.7.4.2.1 Basic Flow

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

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

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

1. Internal Application Error.

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

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

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

Table 1 System State Information for Internal Application Error

## 2. External Application Error

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

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

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	<p>The contact information of the domain controller.</p> <p>The contact information entails:</p> <p>Name of domain controller</p> <p>Email Id / Short Messaging Services (SMS) / Telephone / Mobile Phone</p> <p>Means of Notification</p>	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

Table 2. System State Information for External Application

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

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

#### 2.2.7.4.2.2 Alternative Flow

1: Error Category Does Not Exist

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

2. Error Code Does Not Exist

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

3. Error Severity Does Not Exist

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

4. Log Utility Functional Element Not Available.

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

#### 2.2.7.4.3 Special Requirements

None

1205	<b>2.2.7.4.4</b>	<b>Pre-Conditions</b>
1206	None	
1207	<b>2.2.7.4.5</b>	<b>Post-Conditions</b>
1208	None	
1209		
1210		
1211		

## 2.3 Event Handler Functional Element

### 2.3.1 Motivation

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

Managing the information flow through a subscription based mechanism,

Streamlining information into meaningful categories so as to improve relevancy to a potential consumer of the information, and

Refining information flow via a filtering mechanism

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

Primary Requirements

- MANAGEMENT-111,
- PROCESS-005, and
- PROCESS-100 to PROCESS-117.

Secondary Requirements

- None

### 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.

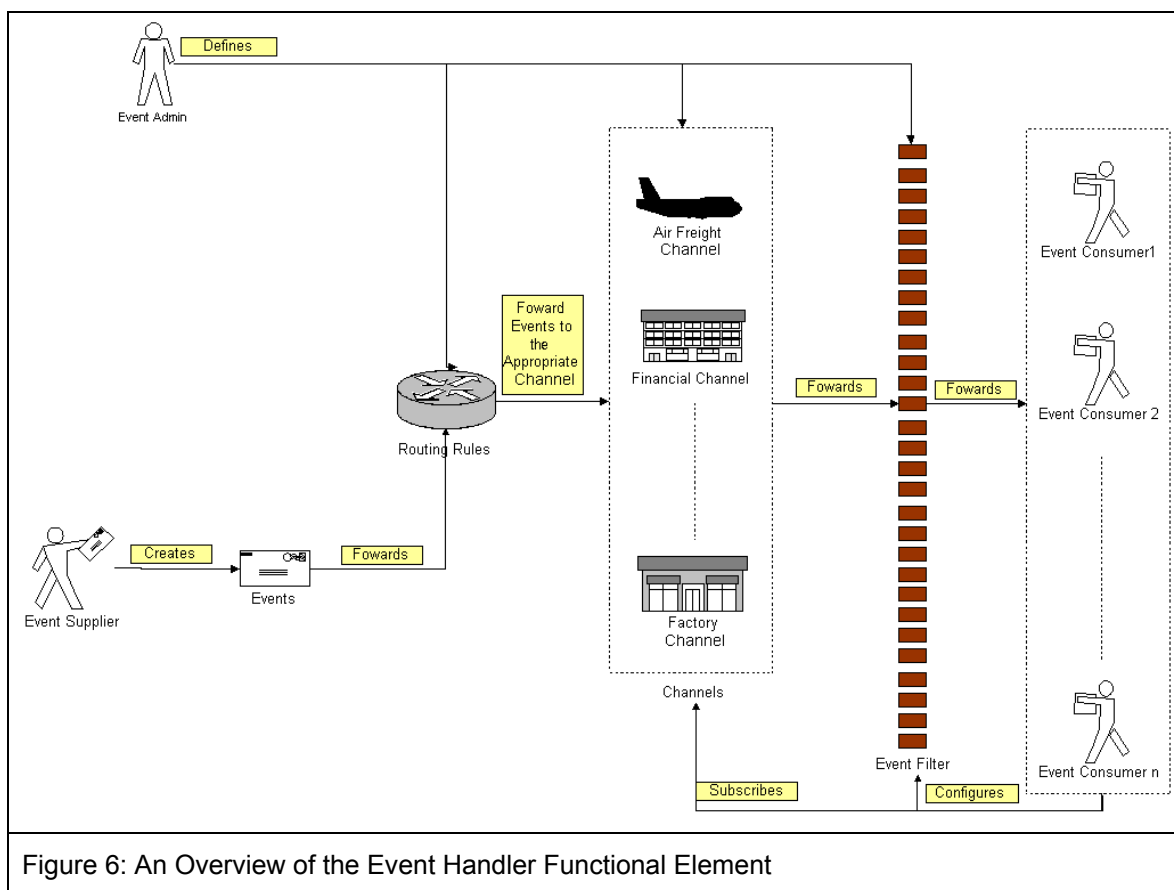


Figure 6: An Overview of the Event Handler Functional Element

1233

1234 Figure 3 depicts the basic concepts of how the participating entities collaborate together in the  
 1235 Event Handler Functional Element. Beginning with the event supplier who generates an event,  
 1236 the event is subsequently routed to the routing rules engine. Depending on the rules specified by  
 1237 the event administrator on the engine, the event could be routed to an appropriate channel, for  
 1238 example, the airfreight channel. In this case, a notification message will be sent to the subscribing  
 1239 event consumers. In between that, there is a filtering engine to determine if a particular event is  
 1240 meaningful to the intended recipients and this is configurable by the recipients themselves.

### 1241 2.3.3 Key Features

1242 Implementations of the Event Handler Functional Element are expected to provide the following  
 1243 key features:

- 1244 1. The Functional Element MUST provide the capability to manage the creation (or registration)  
 1245 and deletion of instances of the following concepts based on a pre-defined structure:
- 1246 1.1. Event Supplier,
- 1247 1.2. Event Consumer,
- 1248 1.3. Event,
- 1249 1.4. Filter,
- 1250 1.5. Channel, and
- 1251 1.6. Routing Rule.

- 1252 2. The Functional Element MUST provide the capability to manage all the information (attribute  
1253 values) stored in such concepts. This includes the capability to retrieve and update  
1254 attribute's values belonging to the concepts mentioned in Key Feature (1).
- 1255 3. The Functional Element MUST provide the capability to enable Event Suppliers to trigger  
1256 relevant Events.
- 1257 4. The Functional Element MUST provide a mechanism to associate/unassociate Routing  
1258 Rules to an Event.
- 1259 *Example: As shown in Figure 1, where an event can be routed to Air Freight or Financial*  
1260 *Channel or even to all channels based on the Routing Rules that are associated*  
1261 *with the Event.*
- 1262 5. As part of Key Feature (3), the Routing Rules must be able to route an event to all, specified  
1263 Channels or individual Event Consumers.
- 1264 6. The Functional Element MUST enable Event Consumers to execute the following tasks to  
1265 improve the relevancy of the incoming events"
- 1266 6.1. Subscribe/Unsubscribe to relevant Channel(s), and  
1267 6.2. Apply a filter to the appropriate channel or event, which helps to refine the criteria of a  
1268 useful event further.
- 1269 7. The Functional Element MUST provide the capability to notify relevant Event Consumers  
1270 when an event occurs.
- 1271 Examples of notification types include SMS, email and Web Services invocations.
- 1272 8. As part of Key Feature (6), the notification must be able to handle differing requirements  
1273 arising from different notification formats.
- 1274 *Example: If the incoming event contains 2 important attributes, the order or position of*  
1275 *these 2 attributes must be configurable to suit the convenience of the Event*  
1276 *Consumer. This is extremely important in the case of Web Service Invocations.*
- 1277 10. The Functional Element MUST provide a mechanism for managing the concepts specified  
1278 across different application domains.
- 1279 *Example: Namespace control mechanism*  
1280
- 1281 In addition, the following key features could be provided to enhance the Functional Element  
1282 further:
- 1283 1. The Functional Element MAY provide a mechanism to enable active event detection.
- 1284 2. If Key Feature (1) is implemented, then the Functional Element MUST provide the following  
1285 capabilities also:
- 1286 2.1. Non-intrusive detection  
1287 *Example: The detection of a new event through periodic inspection of the audit log.*
- 1288 2.2. Configurable event detection schedule  
1289 *Example: To inspect the audit log every 2 hours, where the duration between*  
1290 *inspections is configurable.*
- 1291 2.3. Ability to retrieve relevant data from external source(s) for further event processing by  
1292 Event Handler  
1293 *Example: To retrieve Error Type and Message from audit log.*
- 1294 3. The Functional Element MAY provide the capability to record event processing within the  
1295 Event Handler. The logging of event processing includes the occurrences of event, sending  
1296 of notifications, warning and error messages generated in the processing of events.
- 1297 4. The Functional Element MAY provide the capability scheduled-based event notification.  
1298

1299 **2.3.4 Interdependencies**

**Direct Dependency**

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

1300

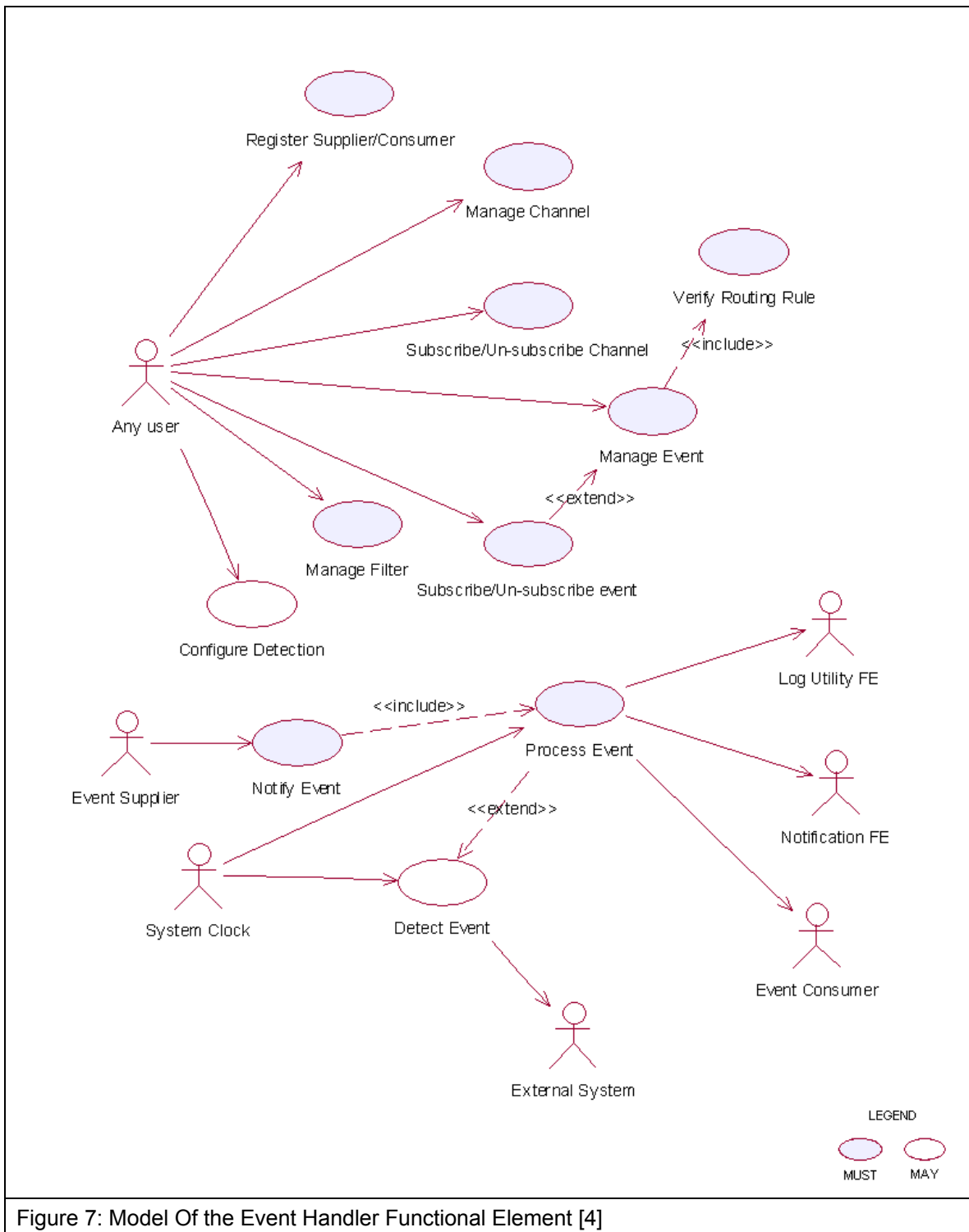
**Interaction Dependency**

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

1301

1302 **2.3.5 Related Technologies and Standards**

1303 None



## 1305    **2.3.7 Usage Scenarios**

### 1306    **2.3.7.1 Register Supplier/Consumer**

#### 1307    **2.3.7.1.1 Description**

1308    This use case allows the user to register itself to the Event Handler Functional Element as an  
1309    event supplier or an event consumer.

#### 1310    **2.3.7.1.2 Flow of Events**

##### 1311    **2.3.7.1.2.1 Basic Flow**

1312    The use case begins when the user of the Event Handler wants to register an event supplier or  
1313    event consumer with the Event Handler.

1314    1: The user sends a request to Event Handler together with its profile data and operation.

1315    2: Based on the operation it specified, one of the following sub-flows is executed:

1316    If the operation is '**Register as supplier**', then sub-flow 2.1 is executed.

1317    If the operation is '**Register as consumer**', then sub-flow 2.2 is executed.

1318    If the operation is '**Un-register as supplier**', then sub-flow 2.3 is executed.

1319    If the operation is '**Un-register as consumer**', then sub-flow 2.4 is executed.

1320    If the operation is '**Update supplier**', then sub-flow 2.5 is executed.

1321    If the operation is '**Update consumer**', then sub-flow 2.6 is executed.

1322    If the operation is '**Retrieve supplier**', then sub-flow 2.7 is executed.

1323    If the operation is '**Retrieve consumer**', then sub-flow 2.8 is executed.

1324        2.1: Register as Supplier.

1325            2.1.1: The Functional Element gets the user profile data, i.e. namespace, name,  
1326            description and type.

1327            2.1.2: The Functional Element registers the user as event supplier.

1328            2.1.3: The Functional Element returns the Supplier Id to the user.

1329        2.2: Register as Consumer.

1330            2.2.1: The Functional Element gets the user profile data, i.e. namespace, name,  
1331            description and type.

1332            2.2.2: The Functional Element registers the user as event consumer.

1333            2.2.3: The Functional Element returns the Consumer Id to the user.

1334        2.3: Un-register as Supplier.

1335            2.3.1: The Functional Element gets the user namespace and name or User Id.

1336            2.3.2: The Functional Element checks whether the user is a supplier.

1337            2.3.3: The Functional Element removes the user as supplier.



1338 2.4: Un-register as Consumer.

1339 2.4.1: The Functional Element gets the user namespace and name or User Id.

1340 2.4.2: The Functional Element checks whether the user is a consumer.

1341 2.4.3: The Functional Element removes the user as consumer.

1342 2.5: Update Supplier.

1343 2.5.1: The Functional Element gets the user namespace and name or User Id together

1344 with the user profile.

1345 2.5.2: The Functional Element checks whether the user is a supplier.

1346 2.5.2: The Functional Element updates the user profile.

1347 2.6: Update Consumer.

1348 2.6.1: The Functional Element gets the user namespace and name or User Id together

1349 with the user profile.

1350 2.6.2: The Functional Element checks whether the user is a consumer.

1351 2.6.3: The Functional Element updates the user profile.

1352 2.7: Retrieve Supplier.

1353 2.7.1: The Functional Element gets the user namespace and name or User Id.

1354 2.7.2: The Functional Element checks whether the user is a supplier.

1355 2.7.3: The Functional Element returns the user profile.

1356 2.8: Retrieve Consumer.

1357 2.8.1: The Functional Element gets the user namespace and name or User Id.

1358 2.8.2: The Functional Element checks whether the user is a consumer.

1359 2.8.3: The Functional Element returns the user profile.

1360 3: The Functional Element returns the results to indicate the success or failure of this operation to

1361 the user and the use case ends.

1362 **2.3.7.1.2.2 Alternative Flows**

1363 1: Supplier Already Registered.

1364 1.1: If in the basic flow 2.1.2, the user already registered as supplier, Functional Element will

1365 return an error message to the user and the use case ends.

1366 2: Consumer Already Registered.

1367 2.1: If in the basic flow 2.2.2, the user already registered as consumer, Functional Element

1368 will return an error message to the user and the use case ends.

1369 3: Supplier or Consumer Not Registered.

1370 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  
1371 registered, Functional Element will return an error message to the user and the use case  
1372 ends.

1373 4: Persistency Mechanism Error.

1374 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  
1375 perform data persistency, Functional Element will return an error message to the user and the  
1376 use case ends.

1377

1378 **2.3.7.1.3 Special Requirements**

1379 None.

1380 **2.3.7.1.4 Pre-Conditions**

1381 None.

1382 **2.3.7.1.5 Post-Conditions**

1383 None.

1384 **2.3.7.2 Manage Channel**

1385 **2.3.7.2.1 Description**

1386 This use case allows the user to manage channels.

1387 **2.3.7.2.2 Flow of Events**

1388 **2.3.7.2.2.1 Basic Flow**

1389 The use case begins when the user wants to create/retrieve/update/delete a channel

1390 1: The user sends request to manipulate a channel.

1391 2: Based on the operation it specifies, one of the following sub-flows is executed:

1392 If the operation is '**Create Channel**', the sub-flow 2.1 is executed.

1393 If the operation is '**Retrieve Channel**', the sub-flow 2.2 is executed.

1394 If the operation is '**Update Channel**', the sub-flow 2.3 is executed.

1395 If the operation is '**Delete Channel**', the sub-flow 2.4 is executed.

1396 2.1: Create Channel.

1397 2.1.1: The Functional Element gets channel definition, i.e. namespace, channel name  
1398 and description.

1399 2.1.2: The Functional Element checks whether the channel exists.

1400 2.1.3: The Functional Element creates the channel.

1401 2.2: Retrieve Channel.

1402 2.2.1: The Functional Element gets namespace, channel name and retrieve condition.

1403 2.2.2: The Functional Element retrieves the channel's information according to the  
1404 condition.

1405 2.3: Update Channel.

1406 2.3.1: The Functional Element gets namespace, channel name and description.

1407 2.3.2: The Functional Element checks whether the channel exists.

1408 2.3.3: The Functional Element updates the channel definition.

1409 2.4: Delete Channel.

1410 2.4.1: The Functional Element gets namespace and channel name.

1411 2.4.2: The Functional Element checks whether the channel exists.

1412 2.4.3: The Functional Element removes the channel from the Functional Element.

1413 3: The Functional Element returns the results of the operation to the user and the use case ends.

1414 **2.3.7.2.2.2 Alternative Flows**

1415 1: Channel Already Exists.

1416 1.1: If in the basic flow 2.1.2, the channel is already defined, Functional Element returns an  
1417 error message and the use case ends.

1418 2: Conditional Retrieving.

1419 2.1: In the basic flow 2.2.2:

1420 2.1 1: If the condition is the retrieval by channel name and the channel does not exist,  
1421 then it will go to Alternative Flow 3.

1422 2.1.2: If the condition is the retrieval of one channel definition, it returns the definition of  
1423 that channel and the use case ends.

1424 2.1.3: If the condition is the retrieval of all channels' information, it returns all channels  
1425 definition and the use case ends.

1426 2.1.4: If the condition is the retrieval of channel through channel description, it will return  
1427 all matched channels and the use case ends.

1428 2.1.5: If the condition is the retrieval of registered consumers, it returns the list of  
1429 consumer registered on the channel and the use case ends.

1430 3: Channel Not Found.

1431 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the channel does not exist, Functional  
1432 Element will return an error message and the use case ends.

1433 4: Consumer Not Found.

1434 4.1: If in the basic flow 2.1.3, 2.5.3 and 2.6.3, the event consumer does not exist,  
1435 Functional Element will return an error message and the use case ends.

1436 5: Extension Point.

1437 5.1: If in the basic flow 2.1.3, and 2.3.3, the event consumers that subscribed to the  
1438 channel are provided, the use case Subscribe/un-subscribe channel will be extended.

### 1439 **2.3.7.2.3 Special Requirements**

1440 None.

### 1441 **2.3.7.2.4 Pre-Conditions**

1442 None.

### 1443 **2.3.7.2.5 Post-Conditions**

1444 None.

## 1445 **2.3.7.3 Subscribe/Un-subscribe To Channel**

### 1446 **2.3.7.3.1 Description**

1447 This use case performs the subscription or un-subscription on a channel for an event consumer.

### 1448 **2.3.7.3.2 Flow of Events**

#### 1449 **2.3.7.3.2.1 Basic Flow**

1450 The use case begins when the user wants to subscribe or un-subscribe to a channel.

1451 1: The user sends the request.

1452 2: Based on the operation it specifies, one of the following sub-flows is executed:

1453 If the operation is '**Subscribe to Channel**', then sub-flow 2.1 is executed.

1454 If the operation is '**Un-Subscribe to Channel**', then sub-flow 2.2 is executed.

1455 2.1: Subscribe To Channel.

1456 2.1.1: The Functional Element gets event consumer Id, or consumer namespace and  
1457 consumer name, together with channel namespace and channel name.

1458 2.1.2: The Functional Element checks whether the channel exists.

1459 2.1.3: The Functional Element adds the subscription of the consumer to the channel.

1460 2.2: Un-Subscribe To Channel.

1461 2.2.1: The Functional Element gets event consumer Id, or consumer namespace and  
1462 consumer name, together with channel namespace and channel name.

1463 2.2.2: The Functional Element checks whether the channel exists.

1464 2.2.3: The Functional Element removes the subscription of the consumer to the channel.

1465 3: The Functional Element returns the results of the operation to the user and the use case ends.

#### 1466 **2.3.7.3.2.2 Alternative Flows**

1467 1: Channel Not Found.

1468 1.1: If in the basic flow 2.1.2 and 2.2.2, the channel specified does not exist, Functional  
1469 Element will return an error message to the user and the use case ends.

1470 2: Event Consumer Not Found.

1471 2.1: If in the basic flow 2.1.2 and 2.2.2, the event consumer related does not exist, Functional  
1472 Element will return an error message to the user and the use case ends.

1473 **2.3.7.3.3 Special Requirements**

1474 None.

1475 **2.3.7.3.4 Pre-Conditions**

1476 None.

1477 **2.3.7.3.5 Post-Conditions**

1478 None.

1479 **2.3.7.4 Manage Event**

1480 **2.3.7.4.1 Description**

1481 This use case describes the scenarios of managing events.

1482 **2.3.7.4.2 Flow of Events**

1483 **2.3.7.4.2.1 Basic Flow**

1484 The use case begins when the user wants to manage events.

1485 1: The user sends a request to the Functional Element.

1486 2: Based on the operation it specifies, one of the following sub-flows is executed:

1487 If the operation is '**Create Event**', then sub-flow 2.1 is executed.

1488 If the operation is '**Retrieve Event Information**', then sub-flow 2.2 is executed.

1489 If the operation is '**Update Event Definition**', then sub-flow 2.3 is executed.

1490 If the operation is '**Delete Event**', then sub-flow 2.4 is executed.

1491 If the operation is '**Assign Flow**', then sub-flow 2.5 is executed.

1492 If the operation is '**Un-Assign Flow**', then sub-flow 2.6 is executed.

1493 2.1: Create Event

1494 2.1.1: The Functional Element gets event definition including namespace, event name,  
1495 event description, event routing rule, and event attributes definition.

1496 2.1.2: The Functional Element verifies the parameters.

1497 2.1.3: The Functional Element verifies the routing rule through use case verify routing  
1498 rule.

1499 2.1.4: The Functional Element creates event definition by recording the definition of  
1500 event.

1501 2.2: Retrieve Event.

1502 2.2.1: The Functional Element gets namespace, event name, and condition.

1503 2.2.2: The Functional Element retrieves the event definition according to the condition.

1504 2.3: Update Event Definition

1505 2.3.1: The Functional Element gets event definition including namespace, event name,

1506 event description, event routing rule, and event attributes definition.

1507 2.3.2: The Functional Element verifies the parameters.

1508 2.3.3: The Functional Element verifies the routing rule through use case verify routing

1509 rule.

1510 2.3.4: The Functional Element updates the event definition.

1511 2.4: Delete Event.

1512 2.4.1: The Functional Element gets namespace and event name.

1513 2.4.2: The Functional Element checks whether the event exists.

1514 2.4.3: The Functional Element deletes the event definition.

1515 2.5: Assign Flow.

1516 2.5.1: The Functional Element gets namespace, event name and flow name.

1517 2.5.2: The Functional Element checks whether the event exists and flow defined.

1518 2.5.3: The Functional Element assigns the flow to the event.

1519 2.6: Un-assign Flow.

1520 2.6.1: The Functional Element gets namespace, event name and flow name.

1521 2.6.2: The Functional Element checks whether the event exists and flow defined.

1522 2.6.3: The Functional Element un-assigns the flow to the event.

1523 3: The Functional Element returns the results of the operation to the user and the use case ends.

1524 **2.3.7.4.2.2 Alternative Flows**

1525 1: Event Already Exist.

1526 1.1: If in the basic flow 2.1.2, the event already exists, Functional Element will return an error

1527 message to the user and the use case ends.

1528 2: Parameters Are Invalid.

1529 2.1: If in the basic flow 2.1.2 and 2.3.2, the parameters provided are invalid, Functional

1530 Element will return an error message to the user and the use case ends.

1531 3: Event Not Found.

1532 3.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the event does not exist, Functional Element

1533 will return an error message to the user and the use case ends.

1534 4: Flow Not Defined.

1535 4.1: If in the basic flow 2.1.2 and 2.3.2, the flow does not exist, Functional Element will return  
1536 an error message to the user and the use case ends.

1537 5: Condition Retrieve.

1538 5.1: In the basic flow 2.2.2:

1539 5.1.1: If the retrieving condition is the retrieval of event definition based on event name, it  
1540 returns event definition and the use case ends.

1541 5.1.2: If the retrieving condition is the retrieval of all event definition, it returns all event  
1542 definition and the use case ends.

1543 5.1.3: If the retrieving condition is the retrieval of events assigned to specified channel, it  
1544 returns the list of event definitions.

1545 5.1.4: If the retrieving condition is the retrieval of channels associated with specified  
1546 event, it returns the list of channel definition.

1547 6: Extension Point.

1548 6.1: If in the basic flow 2.1.4, and 2.3.4, the event consumers that subscribed to the event are  
1549 provided, the use case Subscribe/Un-subscribe event will be extended.

1550 **2.3.7.4.3 Special Requirements**

1551 None.

1552 **2.3.7.4.4 Pre-Conditions**

1553 None.

1554 **2.3.7.4.5 Post-Conditions**

1555 None.

1556 **2.3.7.5 Subscribe/Un-subscribe To Event**

1557 **2.3.7.5.1 Description**

1558 This use case performs the subscription or un-subscription on an event for an event consumer.

1559 **2.3.7.5.2 Flow of Events**

1560 **2.3.7.5.2.1 Basic Flow**

1561 The use case begins when the user wants to subscribe or un-subscribe an event.

1562 1: The user sends a request.

1563 2: Based on the operation it specifies, one of the following sub-flows is executed:

1564 If the operation is '**Subscribe to Event**', then sub-flow 2.1 is executed.

1565 If the operation is '**Un-Subscribe to Event**', then sub-flow 2.2 is executed.

1566 2.1: Subscribe To Event.

1567 2.1.1: The Functional Element gets event consumer Id, or consumer namespace and  
1568 consumer name, together with event namespace and event name.

1569 2.1.2: The Functional Element checks whether the event exists.

1570 2.1.3: The Functional Element adds the subscription of the consumer to the event.

1571 2.2: Un-Subscribe To Event.

1572 2.2.1: The Functional Element gets event consumer Id, or consumer namespace and  
1573 consumer name, together with event namespace and event name.

1574 2.2.2: The Functional Element checks whether the event exists.

1575 2.2.3: The Functional Element removes the subscription of the consumer to the event.

1576 3: The Functional Element returns the results of the operation to the user and the use case ends.

1577 **2.3.7.5.2.2 Alternative Flows**

1578 1: Event Not Found.

1579 1.1: If in the basic flow 2.1.2 and 2.2.2, the event specified does not exist, Functional Element  
1580 will return an error message to the user and the use case ends.

1581 2: Event Consumer Not Found.

1582 2.1: If in the basic flow 2.1.2 and 2.2.2, the event consumer related does not exist, Functional  
1583 Element will return an error message to the user and the use case ends.

1584 **2.3.7.5.3 Special Requirements**

1585 None.

1586 **2.3.7.5.4 Pre-Conditions**

1587 None.

1588 **2.3.7.5.5 Post-Conditions**

1589 None.

1590 **2.3.7.6 Verify Routing Rule**

1591 **2.3.7.6.1 Description**

1592 This use case verifies the syntax of routing rule.

1593 **2.3.7.6.2 Flow of Events**

1594 **2.3.7.6.2.1 Basic Flow**

1595 The use case begins when the user wants to verify the correctness of a routing expression.

1596 1: The user sends a request.

1597 2: The Functional Element gets the routing expression.



- 1598 3: The Functional Element checks the syntax of routing expression.  
1599 4: The Functional Element verifies the parameters.  
1600 5: The Functional Element returns the status of the operation to the user and the use case ends.

#### 1601 **2.3.7.6.2.2 Alternative Flows**

- 1602 1: Routing Rule Expression Syntax Error.  
1603 1.1: If in the basic flow 3, there is a syntax error, Functional Element will return an error  
1604 message to the user and the use case ends.  
1605 2: Event Consumer Not Found.  
1606 2.1: If in the basic flow 4, the event consumer related does not exist, Functional Element will  
1607 return an error message to the user and the use case ends.

#### 1608 **2.3.7.6.3 Special Requirements**

1609 None.

#### 1610 **2.3.7.6.4 Pre-Conditions**

1611 None.

#### 1612 **2.3.7.6.5 Post-Conditions**

1613 None.

### 1614 **2.3.7.7 Manage Filter**

#### 1615 **2.3.7.7.1 Description**

1616 A filter is used to filter out certain events to those event consumers even though they are the  
1617 intended receivers according to the routing rules.

#### 1618 **2.3.7.7.2 Flow of Events**

##### 1619 **2.3.7.7.2.1 Basic Flow**

1620 The use case begins when the user wants to create/retrieve/update/delete a filter.

- 1621 1: The user sends a request to manage a filter.  
1622 2: Based on the operation it specifies, one of the following sub-flows is executed:  
1623 If the operation is '**Create Filter**', then sub-flow 2.1 is executed.  
1624 If the operation is '**Retrieve Filter**', then sub-flow 2.2 s executed.  
1625 If the operation is '**Update Filter**', then sub-flow 2.3 is executed.  
1626 If the operation is '**Delete Filter**', then sub-flow 2.4 is executed.

1627 2.1: Create Filter.

1628 2.1.1: The Functional Element gets filter definition, i.e. consumer namespace, consumer  
1629 name, filter name, description, event name or channel name.

1630            2.1.2: The Functional Element checks whether the event or channel exists.

1631            2.1.3: The Functional Element saves the filter definition.

1632            2.2: Retrieve Filter.

1633            2.2.1: The Functional Element gets the filter name.

1634            2.2.2: The Functional Element retrieves the filter information according to the name.

1635            2.3: Update Filter.

1636            2.3.1: The Functional Element gets filter definition, i.e. consumer namespace, name, filter

1637            name, description, event name or channel name.

1638            2.3.2: The Functional Element checks the parameters.

1639            2.3.3: The Functional Element updates the filter definition.

1640            2.4: Delete Filter.

1641            2.4.1: The Functional Element gets namespace and filter name.

1642            2.4.2: The Functional Element checks whether the filter exists.

1643            2.4.3: The Functional Element removes the filter from the Functional Element.

1644            3: The Functional Element returns the results of the operation to the user and the use case ends.

#### 1645    **2.3.7.7.2.2 Alternative Flows**

1646            1: Filter Already Exists.

1647            1.1: If in the basic flow 2.1.2, the filter is already defined, Functional Element will return an

1648            error message and the use case ends.

1649            2: Event Not Found.

1650            2.1: If in the basic flow 2.1.2 and 2.3.2, the event used does not exist, Functional Element will

1651            return an error message and the use case ends.

1652            3: Channel Not Found.

1653            3.1: If in the basic flow 2.1.2 and 2.3.2, the channel used does not exist, Functional Element

1654            will return an error message and the use case ends.

1655            4: Consumer Not Found.

1656            4.1: If in the basic flow 2.1.3, 2.5.3, and 2.6.3, the event consumer does not exist, Functional

1657            Element will return an error message and the use case ends.

#### 1658    **2.3.7.7.3 Special Requirements**

1659            None.

#### 1660    **2.3.7.7.4 Pre-Conditions**

1661            None.

1662     **2.3.7.7.5 Post-Conditions**

1663     None.

1664     **2.3.7.8 Notify Event**

1665     **2.3.7.8.1 Description**

1666     This use case allows the event supplier to notify an event to the Event Handler Functional  
1667     Element. Once the Event Handler Functional Element receives the notification, it will process the  
1668     event based on the processing logic defined.

1669     **2.3.7.8.2 Flow of Events**

1670     **2.3.7.8.2.1 Basic Flow**

1671     The use case begins when the user wants to notify an event.

1672     1: The user sends a notification.

1673     2: The Functional Element receives the notification with parameters, i.e. event supplier id or event  
1674     supplier namespace and name.

1675     3: The Functional Element checks whether the event is defined and event supplier is registered.

1676     4: Include use case Process Event to process the notification of event.

1677     5: The Functional Element returns the status of the operation to the user and the use case ends.

1678     **2.3.7.8.2.2 Alternative Flows**

1679     1: User Is Not Registered.

1680         1.1: If in the basic flow 3, the user is not registered, Functional Element will return an error  
1681         message to the user and the use case ends.

1682     2: Event Not Defined.

1683         2.1: If in the basic flow 3, the event is not defined, Functional Element will return an error  
1684         message to the user and the use case ends.

1685     3: Error Returned.

1686         3.1: If in the basic flow 4, an error is returned by use case Process event, Functional Element  
1687         will return an error message to the user and the use case ends.

1688     **2.3.7.8.3 Special Requirements**

1689     None.

1690     **2.3.7.8.4 Pre-Conditions**

1691     None.

1692     **2.3.7.8.5 Post-Conditions**

1693     None.

1694     **2.3.7.9 Configure Monitoring**

1695     **2.3.7.9.1 Description**

1696     This use case describes the capability of configuration on event monitoring. Based on the  
1697     configuration, Event Handler will pro-actively check whether an event has happened.

1698     **2.3.7.9.2 Flow of Events**

1699     **2.3.7.9.2.1 Basic Flow**

1700     The use case begins when the user wants to configure the event monitoring.

1701     1: The user sends a request to manage a filter.

1702     2: Based on the operation it specifies, one of the following sub-flows is executed:

1703     If the operation is '**Add Configuration**', then sub-flow 2.1 is executed.

1704     If the operation is '**Remove Configuration**', then sub-flow 2.2 is executed.

1705         2.1: Add Configuration.

1706             2.1.1: The Functional Element gets configuration definition, i.e. configuration name,  
1707             namespace, event name, connection parameters, condition that signifies the events and  
1708             schedule.

1709             2.1.2: The Functional Element saves filter definition.

1710         2.2: Remove Configuration.

1711             2.2.1: The Functional Element gets configuration name.

1712             2.2.2: The Functional Element removes the configuration.

1713     3: The Functional Element returns the results of the operation to the user and the use case ends.

1714     **2.3.7.9.2.2 Alternative Flows**

1715     1: Configuration Exist.

1716         1.1: If in the basic flow 2.1.2, the configuration already exists, Functional Element will return  
1717         an error message and the use case ends.

1718     **2.3.7.9.3 Special Requirements**

1719     None.

1720     **2.3.7.9.4 Pre-Conditions**

1721     None.

1722     **2.3.7.9.5 Post-Conditions**

1723     None.

1724    **2.3.7.10      Detect Event**

1725    **2.3.7.10.1    Description**

1726    This use case describes the event monitoring capability that Event Handler provides. Once Event  
1727    Handler detects an event, it will trigger the pre-defined process for the event.

1728    **2.3.7.10.2    Flow of Events**

1729    **2.3.7.10.2.1   Basic Flow**

1730    The use case begins when the Functional Element clock generates the trigger.

1731    1: The Functional Element clock generates a trigger.

1732    2: The Functional Element receives the trigger and checks the condition for pre-defined  
1733    monitoring sources.

1734    3: The Functional Element checks whether the event happens.

1735    4: The Functional Element returns the results of the operation and the use case ends.

1736    **2.3.7.10.2.2   Alternative Flows**

1737    1: External Functional Element Not Available.

1738        1.1: If in the basic flow 3, the external Functional Element is not available and the Event  
1739        Handler cannot make a connection, Functional Element will return an error message and the  
1740        use case ends.

1741    2: Data Not Available.

1742        2.1: If in the basic flow 3, the data that signifies the event cannot be accessed, Functional  
1743        Element will return an error message and the use case ends.

1744    3: Extension Point.

1745        3.1: If in the basic flow 3, the event happens, Functional Element will extend to use case  
1746        Process event.

1747    **2.3.7.10.3    Special Requirements**

1748    None.

1749    **2.3.7.10.4    Pre-Conditions**

1750    None.

1751    **2.3.7.10.5    Post-Conditions**

1752    None.

1753    **2.3.7.11      Process Event**

1754    **2.3.7.11.1    Description**

1755    This use case describes the core functionality of Event Handler. It is the engine that processes  
1756    the events. Actor can be the Functional Element clock that triggers the scheduled event  
1757    notification, or any user who wants to notify the event.

1758    **2.3.7.11.2    Flow of Events**

1759    **2.3.7.11.2.1   Basic Flow**

1760    The use case begins when there is a request to process the event.

1761    1: The user sends a request to process an event.

1762    2: Based on the actor of this use case, one of the sub-flows is executed.

1763    If the initiator is the Functional Element clock, then sub-flow '**Initiated By Functional Element**  
1764    **Clock**' is executed.

1765    If the initiator is other than Functional Element clock, then sub-flow '**Initiated By Any User**' is  
1766    executed.

1767        2.1: Initiated By Functional Element Clock.

1768            2.1.1: The Functional Element looks up scheduled events defined to find out time-due  
1769            notification.

1770            2.1.2: The Functional Element retrieves the routing rule for the event.

1771            2.1.3: The Functional Element looks up the corresponding consumers based on the  
1772            routing rule.

1773            2.1.4: The Functional Element retrieves filters defined and find out the event receivers.

1774            2.1.5: The Functional Element notifies or invokes the event consumers based on the  
1775            routing rule defined.

1776        2.2: Initiated By Any User.

1777            2.2.1: The Functional Element retrieves the routing rule for the event.

1778            2.2.2: The Functional Element looks up the corresponding consumers.

1779            2.2.3: The Functional Element retrieves filters defined and find out the event receivers.

1780            2.2.4: The Functional Element notifies or invokes the event consumers based on the  
1781            routing rule defined.

1782    3: The Functional Element logs the notification of event and the use case ends.

1783    **2.3.7.11.2.2   Alternative Flows**

1784    1: Notify Event.

1785    In basic flow 2.1.4 and 2.2.4, based on the type of consumer, one of the sub-flows is execute.

1786    If the consumer type is '**SMTP**', then sub-flow Notify via SMTP is executed.

1787 If the consumer type is '**SMS Gateway**', then sub-flow Notify via SMS Gateway is executed.

1788 If the consumer type is '**Notify RPC-Web Service**', then sub-flow Notify RPC-Web Service is  
1789 executed.

1790 If the consumer type is '**Notify Document Style Web Service**' then sub-flow Notify Document  
1791 style Web Service is executed.

1792 1.1: Notify via SMTP.

1793 1.1.1: The Functional Element gets the pre-defined message for event and forms the  
1794 parameters.

1795 1.1.2: The Functional Element gets the parameters for SMTP server.

1796 1.1.3: The Functional Element sends out the pre-defined message and the use case  
1797 ends.

1798 1.2: Notify via SMS Gateway.

1799 1.2.1: The Functional Element gets the pre-defined message for event and forms the  
1800 parameters.

1801 1.2.2: The Functional Element gets the parameters for the SMS gateway.

1802 1.2.3: The Functional Element sends out the pre-defined message and the use case  
1803 ends.

1804 1.3: Notify RPC-Web Service.

1805 1.3.1: The Functional Element gets the operation parameter.

1806 1.3.2: The Functional Element gets Web Services endpoint parameters.

1807 1.3.3: The Functional Element dynamically invokes the Web Service and the use case  
1808 ends.

1809 1.4: Notify Document Style Web Service.

1810 1.4.1: The Functional Element gets the operation parameter.

1811 1.4.2: The Functional Element gets Web Services endpoint parameters.

1812 1.4.3: The Functional Element dynamically generates the SOAP message and sends to  
1813 the Web Services and the use case ends.

1814 2: Flow Is Defined.

1815 If in the basic flow 2.1.2 and 2.2.1, a flow is defined for the event, Functional Element will perform  
1816 the following steps:

1817 2.1: The Functional Element retrieves all the intended event consumers defined in the flow.

1818 2.2: The Functional Element will go to basic flow 2.2.

1819 2.3: The Functional Element will resume the execution from basic flow 2.1.2 or 2.2.1.

1820 3: Log Utility Not Available.

1821 3.1: If in the basic flow 3, the Log Utility Functional Element is not available, Functional  
1822 Element will return an error message to the user and the use case ends.

1823 4: SMS Gateway Not Available.

1824 4.1: If in the Alternative Flow 1.2.3, the SMS Gateway is not available, Functional Element will  
1825 return an error message to the user and the use case ends.

1826 5: SMTP Server Not Available.

1827 5.1: If in the Alternative Flow 1.1.3, the SMTP server is not available, Functional Element will  
1828 return an error message to the user and the use case ends.

1829 6: RPC Web Service Not Available.

1830 6.1: If in the Alternative Flow 1.3.3, the Web Service is not available, Functional Element will  
1831 return an error message to the user and the use case ends.

1832 7: Document Style Web Service Not Available.

1833 7.1: If in the Alternative Flow 1.4.3, document style Web Service is not available, Functional  
1834 Element will return an error message to the user and the use case ends.

1835 **2.3.7.11.3 Special Requirements**

1836 **2.3.7.11.3.1 Supportability**

1837 The application server used must have a JMS service provided.

1838 **2.3.7.11.4 Pre-Conditions**

1839 None.

1840 **2.3.7.11.5 Post-Conditions**

1841 None.

1842



## 2.4 Group Management Functional Element

### 2.4.1 Motivation

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

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

Primary Requirements

- MANAGEMENT-050 to MANAGEMENT-053, and
- MANAGEMENT-078

Secondary Requirements

- None

### 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 used to segregate the instantiation of the application across different application domains. If a company has two separate standalone applications, 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 aspects 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.

### 2.4.3 Key Features

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

1. The Functional Element MUST provide a basic Group structure with a set of pre-defined attributes.
2. The Functional Element MUST provide the capability to extend on the basic Group structure dynamically.
3. As part of Key Feature (2), this dynamic extension MUST be definable and configurable at runtime implementation of the Functional Element.
4. The Functional Element MUST provide the capability to manage the creation and deletion of instances of Groups based on defined structure.
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.
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.
6. The Functional Element MUST provide a mechanism for managing Groups across different application domains.

*Example: Namespace control mechanism*

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

1. The Functional Element MAY provide a mechanism to enable different Groups to be related to one another.
2. The Functional Element MAY also provide a mechanism to enable hierarchical relationships between Groups.  
*Example: Parent and Child Relationship.*
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.  
*Example: Adding of Group "Dept-A" to "Company-XYZ" – "Dept-A" is a Group, and also part of the collection of Group "Company-XYZ".*

4. The Functional Element MAY provide validity checks when managing information stored in a Group.  
*Example: Adding of User "john" – A validity check could be imposed to ensure that a user "john" exists before adding to into the Group.*

#### 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.

#### 2.4.5 Related Technologies and Standards

None.

## 2.4.6 Model

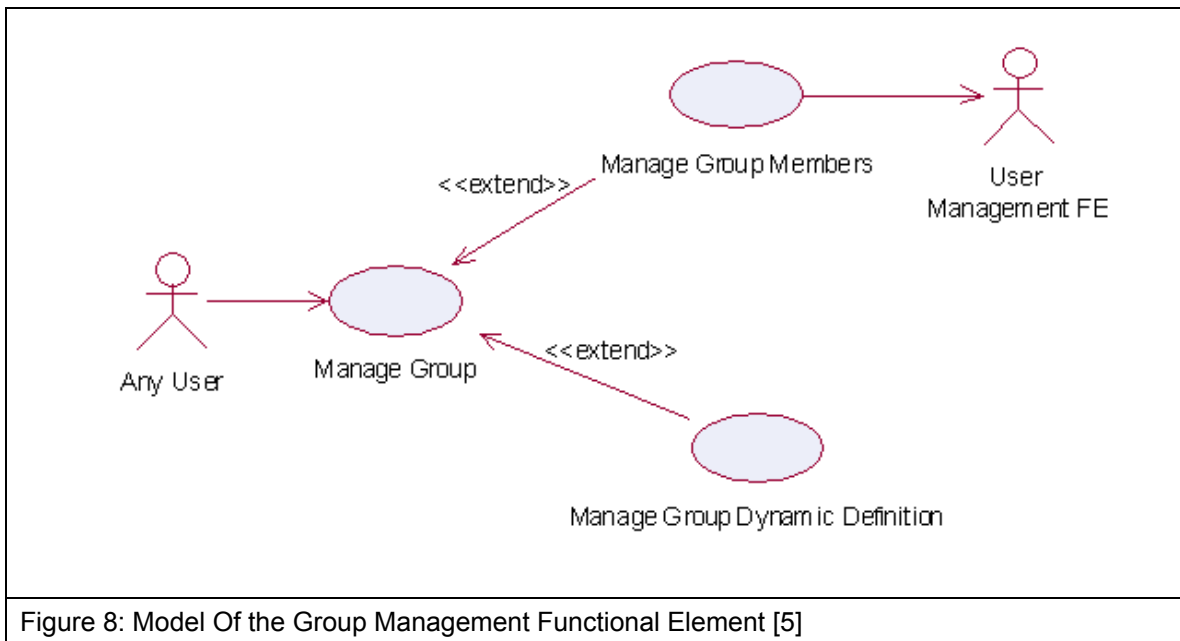


Figure 8: Model Of the Group Management Functional Element [5]

## 2.4.7 Usage Scenarios

### 2.4.7.1 Manage Group

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

#### 2.4.7.1.1 Flow of Events

##### 2.4.7.1.1.1 Basic Flow

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

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

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

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

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

1: Create Group.

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

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

2: Retrieve Group.

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

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

1921 3: Update Group.

1922 3.1: User provides the necessary information for updating the group's attributes.

1923 3.2: Functional Element updates the group and the use case ends.

1924 4: Delete Group.

1925 4.1: User provides the necessary information for deleting a particular group.

1926 4.2: Functional Element deletes the group and the use case ends.

1927 **2.4.7.1.1.2 Alternative Flows**

1928 1: Group Exist.

1929 1.1: In basic flow 1.2, Functional Element detects an identical group. Functional Element

1930 returns an error message and the use case ends.

1931 2: Group Does Not Exist.

1932 2.1: In basic flow 2.2, 3.2 and 4.2, Functional Element cannot find a group that matches the

1933 user's criteria. Functional Element returns an error message and the use case ends.

1934 3: Save Updated Information.

1935 3.1: In basic flow 1.2, 2.2, 3.2 and 4.2, Functional Element fails to save the updated

1936 information. Functional Element returns an error message and the use case ends.

1937 **2.4.7.1.2 Special Requirements**

1938 None.

1939 **2.4.7.1.3 Pre-Conditions**

1940 None.

1941 **2.4.7.1.4 Post-Conditions**

1942 None.

1943 **2.4.7.2 Manage Group Members**

1944 **2.4.7.2.1 Description**

1945 This use case is an extension of the manage group use case. Specifically, it describes the

1946 scenarios to manage members in the group.

1947 **2.4.7.2.2 Flow of Events**

1948 **2.4.7.2.2.1 Basic Flow**

1949 This use case starts when the user wants to manage members in a group.

1950 If user wants to '**Create Members In A Group**', then basic flow 1 is executed.

1951 If user wants to '**Retrieve Members From A Group**', then basic flow 2 is executed.

1952 If user wants to '**Delete Members From A Group**', then basic flow 3 is executed.

- 1953 1: Create Members In A Group.
- 1954 1.1: User provides the necessary information for retrieving the group.
- 1955 1.2: Functional Element adds members to the group and the use case ends.
- 1956 2: Retrieve Members In A Group.
- 1957 2.1: User provides the necessary information for retrieving the group.
- 1958 2.2: Functional Element returns the members and the use case ends.
- 1959 3: Delete Members From Group.
- 1960 3.1: User provides the necessary information for retrieving the group.
- 1961 3.2: User provides the necessary information for deleting members in the group.
- 1962 3.3: Functional Element deletes members from group and the use case ends.
- 1963 **2.4.7.2.2.2 Alternative Flows**
- 1964 1: Group Does Not Exist.
- 1965 1.1: In basic flow 1.1, 2.1 and 3.1, Functional Element cannot find the group requested.
- 1966 Functional Element returns an error message and the use case ends.
- 1967 2: Members Does Not Exist
- 1968 2.1: In basic flow 3.3, the Functional Element attempts to delete a non-existence member.
- 1969 Functional Element returns an error message and the use case ends.
- 1970 **2.4.7.2.3 Special Requirements**
- 1971 None.
- 1972 **2.4.7.2.4 Pre-Conditions**
- 1973 None.
- 1974 **2.4.7.2.5 Post-Conditions**
- 1975 None.
- 1976 **2.4.7.3 Manage Group Dynamic Definition**
- 1977 **2.4.7.3.1 Description**
- 1978 This use case describes scenario involved in managing the dynamic group definition.
- 1979 **2.4.7.3.2 Flow of Events**
- 1980 **2.4.7.3.2.1 Basic Flow**
- 1981 This use case starts when the user wants to manage dynamic group definition. This include
- 1982 create, retrieve, update and delete dynamic group definition.
- 1983 If user wants to 'Create Dynamic Definition For A Group', then basic flow 1 is executed.

1984 If user wants to '**Retrieve Dynamic Definition For A Group**', then basic flow 2 is executed.

1985 If user wants to '**Delete Dynamic Definition For A Group**', then basic flow 3 is executed.

1986 If user wants to '**Update Dynamic Definition For A Group**', then basic flow 4 is executed.

1987

1988 1: Create Dynamic Definition For A Group.

1989 1.1: User provides the additional definition for the group.

1990 1.2: Functional Element creates the additional definition for the group and the use case ends.

1991 2: Retrieve Dynamic Definition For A Group.

1992 2.1: User provides the necessary information to retrieve a particular group.

1993 2.2: Functional Element returns the additional definition for the group and the use case ends.

1994 3: Delete Dynamic Definition For Group.

1995 3.1: User provides the necessary information to retrieve a particular group.

1996 3.2: Functional Element deletes the dynamic definition belonging to the group and the use case ends.

1997

1998 4: Update Dynamic Definition For Group.

1999 4.1: User provides the necessary information to retrieve a particular group.

2000 4.2: User provides the necessary dynamic definition that needs to be updated.

2001 4.3: Functional Element update the dynamic definition and the use case ends.

2002 **2.4.7.3.2.2 Alternative Flows**

2003 1: Group Does Not Exist.

2004 1.1: In basic flow 1.1, 2.1, 3.1 and 4.1, Functional Element cannot find the group specified.

2005 Functional Element returns an error message and the use case ends.

2006 2: Dynamic Group Definition Already Exists.

2007 2.1: In basic flow 1.2, Functional Element returns the error message and the use case ends.

2008 3: Dynamic Group Definition Does Not Exist.

2009 3.1: In basic flow 4.3, Functional Element cannot update the dynamic group definition.

2010 Functional Element returns an error message and the use case ends.

2011 **2.4.7.3.3 Special Requirements**

2012 None.

2013 **2.4.7.3.4 Pre-Conditions**

2014 None.

2015 **2.4.7.3.5 Post-Conditions**

2016 None.

## 2.5 Identity Management Functional Element

### 2.5.1 Motivation

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

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

Identity management generally covers the following aspects:

- Basic user accounts management facilities

- User authentication mechanism(s)

- User authorisation mechanism(s)

- Generation of audit trails for user activities

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

Primary Requirements

- SECURITY-001,
- SECURITY-003 (all),
- SECURITY -004 (all),
- SECURITY -040 and
- SECURITY -041.

Secondary Requirements

- None

### 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.

2050

2051 The following terms mentioned in this document are used in accordance with the terms defined in  
2052 the Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) v1.1  
2053 specification.

2054 Assertion [section 2.3.2]

2055 AudienceRestrictionCondition [section 2.3.2.1.3]

2056 AuthenticationQuery [section 3.3.3]

2057 AuthenticationStatement [section 2.4.3]

2058 KeyInfo [section 5.4.5]

2059 Request [section 3.2.2]

2060 Response [section 3.4.2]

2061 Subject [section 2.4.2.1]

2062

### 2063 2.5.3 Key Features

2064 Implementations of the Identity Management Functional Element are expected to provide the  
2065 following key features:

- 2066 7. The Functional Element MUST be have the mechanism to access an Identity Repository.
- 2067 8. The Functional Element MUST provide the capability to manage the creation and deletion of  
2068 instances of Identity in the said Identity Repository.
- 2069 9. The Functional Element MUST have the mechanisms to manage all the information (attribute  
2070 values) stored in such Identities. This includes the capability to:
- 2071 4.1. Retrieve and update attribute's values belonging to a Identity,
- 2072 4.2. Encrypt sensitive user information,
- 2073 4.3. Authenticate a user, and
- 2074 4.4. Assign/Unassign Access Policy (or Policies).
- 2075 *Example: Different levels of privileges to access protected resources.*
- 2076 10. As part of Key Feature (3.3), the authentication of an Identity MUST be achieved at least  
2077 through the use of a password.
- 2078 11. As part of Key Feature (3.3), the Functional Element MUST also provide the capability to use  
2079 an Assertion Authority for Single Sign-On (SSO) authentication.
- 2080 12. As part of Key Feature (5), the SSO message exchange and protocol MUST use an  
2081 approved standard. Recommendations are available in section 2.5.5.
- 2082 13. As part of Key Feature (3.4), a mechanism MUST be provided to verify the Identity's Access  
2083 Policy on protected Resources.
- 2084 14. The Functional Element MUST provide the capability to create audit trails.
- 2085 *Example: Timestamp of an Identity's access to Resources.*

2086

2087 In addition, the following key features could be provided to enhance the Functional Element  
2088 further:

- 2089 1. The Functional Element MAY provide an Identity Repository.
- 2090 2. If Key Feature (1) is provided, the Functional Element MUST provide the capability to  
2091 manage the creation and deletion of instances of Identities based on a pre-defined structure.
- 2092 3. The Functional Element MAY provide additional storage in the Identity Repository for an  
2093 Identity to customise its preferences.
- 2094 *Example: Identity's preferred subscription of notifications/alerts for news.*
- 2095 4. The Functional Element MAY provide a capability to use an Identity Provider for Federated  
2096 Identity SSO authentication.
- 2097 5. If Key Feature (4) is provided, the Federated Identity SSO message exchange and protocol  
2098 MUST use an approved standard.

2099

### 2100 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.
--------------------------------	--

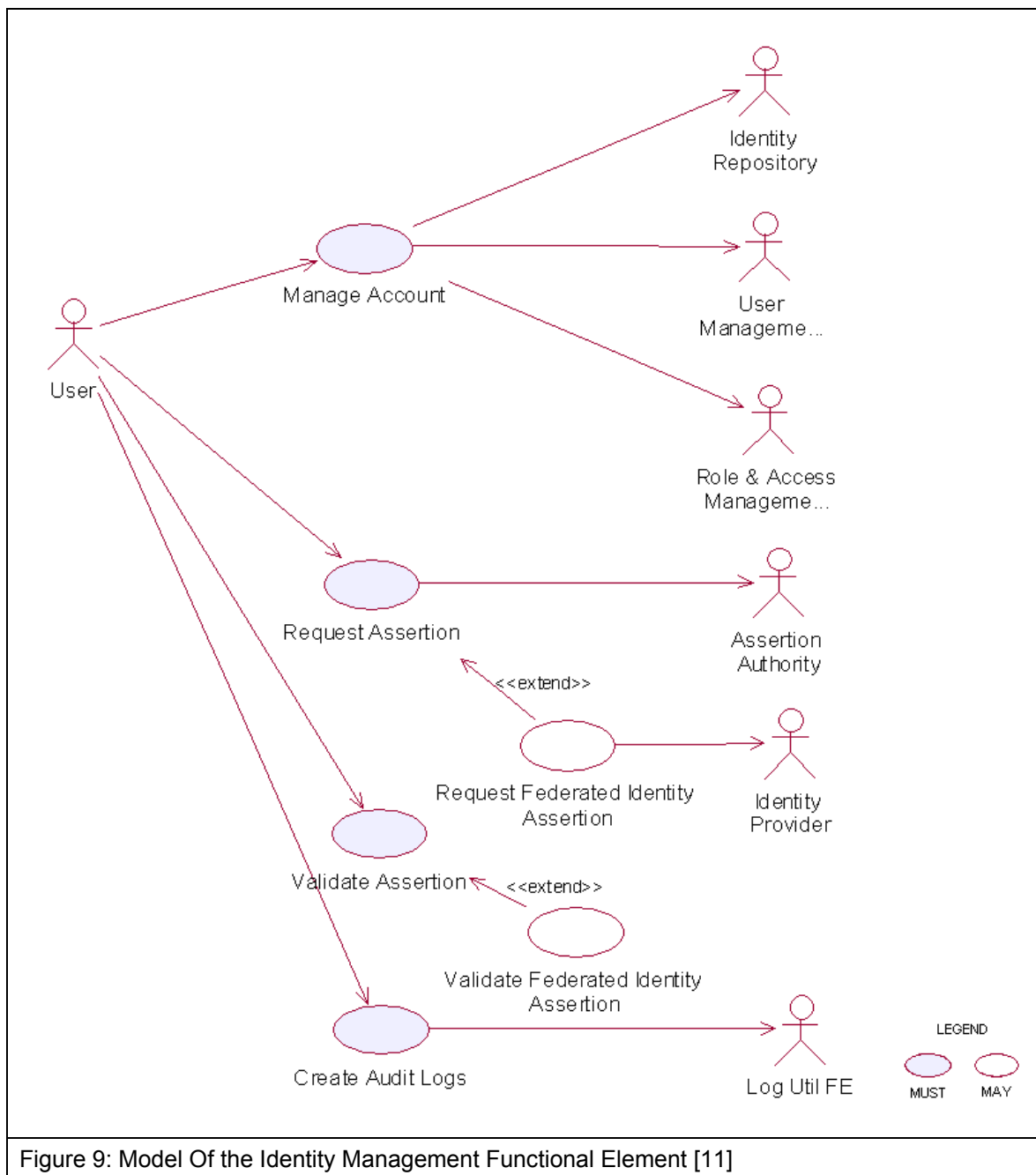
2101

## 2102 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"> <li>• Assertion Schema</li> <li>• Protocol Schema</li> </ul>
Liberty Alliance Project Specifications	<p>Liberty Alliance ID-FF 1.2 Specifications [8]</p> <p>Liberty Alliance ID-WSF 1.0 Specifications [9]</p>
WS-Federation [10]	Web Services Federation Language (WS-Federation) - 08 July 2003

2103

2104



## 2107 **2.5.7 Usage Scenarios**

### 2108 **2.5.7.1 Manage Account**

#### 2109 **2.5.7.1.1 Description**

2110 This use case describes the creation/retrieval/update/deletion of an identity's account. An  
2111 identity's account usually consists of two elements: i) the user information and ii) the associated  
2112 access policy.

2113 As Identity Management Functional Element leverages on the User Management Functional  
2114 Element and Role and Access Management Functional Element to provide for these  
2115 functionalities, please refer to these Functional Elements' use cases for details.

### 2116 **2.5.7.2 Request Assertion**

#### 2117 **2.5.7.2.1 Description**

2118 This use case describes the composition of either 1) an authentication query or 2) an  
2119 authorisation decision query and sending it to the assertion authority.

#### 2120 **2.5.7.2.2 Flow of Events**

##### 2121 **2.5.7.2.2.1 Basic Flow**

2122 This use case starts when the user wants to compose a query to the assertion authority.

2123 If the user requests for an authentication query, then sub-flow 1 is executed.

2124 If the user requests for an authorisation decision query, then sub-flow 2 is executed.

2125 1: Request for Authentication Assertion

2126 1.1: The user composes a valid SAML Request with an AuthenticationQuery and sends it to  
2127 the assertion authority.

2128 1.2: The user waits for an SAML Response from the assertion authority.

2129 1.3: The user obtains the SAML Assertion from the SAML Response and use case ends.

2130 2: Request for Authorisation Decision Assertion

2131 2.1: The user composes a valid SAML Request with an AuthorizationDecisionQuery and  
2132 sends it to the assertion authority.

2133 2.2: The user waits for an SAML Response from the assertion authority.

2134 2.3: The user obtains the SAML Assertion from the SAML Response and use case ends.

##### 2135 **2.5.7.2.2.2 Alternative Flows**

2136 1: Invalid Request

2137 1.1: If in basic flow 1.1 or 2.1, if any of the parameters passed into the request is invalid, the  
2138 Functional Element flag an exception and use case ends.

2139 2: Error message from assertion authority

2140 2.1: If in basic flow 1.3 or 2.3, the assertion authority is unable to return an assertion (e.g.  
 2141 user has not logged on etc.), it returns an error code and an error message.

2142 2.2: The Functional Element flag an error with the error message attached and use case  
 2143 ends.

2144 **2.5.7.2.3 Special Requirements**

2145 None.

2146 **2.5.7.2.4 Pre-Conditions**

2147 None.

2148 **2.5.7.2.5 Post-Conditions**

2149 None.

2150 **2.5.7.3 Validate Assertion**

2151 **2.5.7.3.1 Description**

2152 This use case describes the validation of either 1) the Authentication Assertion or 2) the  
 2153 Authorisation Decision Assertion

2154 **2.5.7.3.2 Flow of Events**

2155 **2.5.7.3.2.1 Basic Flow**

2156 This use case starts when the user wants to check if the assertion it is a valid assertion from the  
 2157 assertion authority.

2158 1: The user passes the assertion to the Functional Element for validation.

2159 2: The Functional Element checks if the assertion is signed by the assertion authority.

2160 3: The Functional Element checks for an un-expired assertion.

2161 4: The Functional Element checks if the assertion has an AudienceRestrictionCondition and  
 2162 verifies that the service provider using the Functional Element is in the audience list.

2163 5: Based on the type of assertion, one of the sub-flows is executed.

2164 • If the user wants to check for a valid authentication assertion, then sub-flow 5.1 is executed.

2165 • If the user wants to check for a valid authorisation decision assertion, then sub-flow 5.2 is  
 2166 executed.

2167 5.1: Validate Authentication Statement

2168 5.1.1: The Functional Element checks if the assertion has indeed an  
 2169 AuthenticationStatement.

2170 5.1.2: The Functional Element checks if the Subject in the AuthenticationStatement  
 2171 matches the userid of the principal.

2172 5.1.3: The Functional Element verifies the Subject with its KeyInfo.

2173            5.1.4: The Functional Element returns the status code to the user and use case ends.

2174            5.2: Validate Authorisation Decision Statement

2175            5.2.1: The Functional Element checks if the assertion has indeed an  
2176            AuthorizationDecisionStatement.

2177            5.2.2: The Functional Element checks if the Resource in the  
2178            AuthorizationDecisionStatement matches the id of the requested resource.

2179            5.2.3: The Functional Element determines if the decision is Permit.

2180            5.2.4: The Functional Element returns the status code to the user and use case ends.

2181            **2.5.7.3.2.2 Alternative Flows**

2182            1: Signature Error

2183            1.1: If in basic flow 2, the Functional Element is unable to verify that the signature is from the  
2184            assertion authority, it returns an error and use case ends.

2185            2: Expired Assertion

2186            2.1: If in basic flow 3, the Functional Element finds that the assertion has already expired, it  
2187            returns an error and use case ends.

2188            3: Audience Error

2189            3.1: If in basic flow 4, the service provider is not in the AudienceRestrictionCondition, the  
2190            Functional Element returns an error and use case ends.

2191            4: Invalid Authentication Assertion

2192            4.1: If in basic flow 5.1.1, the Functional Element is unable to find an  
2193            AuthenticationStatement, it returns an error and use case ends.

2194            5: Mismatch Subject

2195            5.1: If in basic flow 5.1.2, the Functional Element is unable to match the Subject in  
2196            AuthenticationStatement, it returns an error and use case ends.

2197            6: Subject Error

2198            6.1: If in basic flow 5.1.3, the Functional Element is unable to verify the Subject with the  
2199            KeyInfo, it returns an error and use case ends.

2200            7: Invalid Authorisation Decision Assertion

2201            7.1: If in basic flow 5.2.1, the Functional Element is unable to find an  
2202            AuthorizationDecisionStatement, it returns an error and use case ends.

2203            8: Mismatch Resource

2204            8.1: If in basic flow 5.2.2, the Functional Element is unable to match the resource in  
2205            AuthorizationDecisionStatement, it returns an error and use case ends.

2206            **2.5.7.3.3 Special Requirements**

2207            None.

2208     **2.5.7.3.4 Pre-Conditions**

2209     None.

2210     **2.5.7.3.5 Post-Conditions**

2211     None.

2212     **2.5.7.4 Create Audit Logs**

2213     **2.5.7.4.1 Description**

2214     This use case describes logging all identity management activities for audit purposes.

2215     **2.5.7.4.2 Flow of Events**

2216     **2.5.7.4.2.1 Basic Flow**

2217     This use case starts when any of other Functional Element use cases are triggered.

2218     1: The Functional Element opens an audit log file.

2219     2: The Functional Element writes a timestamp identity management activity message into the  
2220     audit log file.

2221     3: The Functional Element closes the audit log file and the use case ends.

2222     **2.5.7.4.2.2 Alternative Flows**

2223     1: Log File Not Created

2224         1.1: If in the basic flow 1, the Functional Element cannot open the audit file, it creates a new  
2225         audit file and use case continues.

2226     2: Error Writing Log

2227         2.1: If in the basic flow 2, the Functional Element has error writing to file, it will flag an  
2228         exception and the use case ends.

2229     **2.5.7.4.3 Special Requirements**

2230     None.

2231     **2.5.7.4.4 Pre-Conditions**

2232     None.

2233     **2.5.7.4.5 Post-Conditions**

2234     None.



## 2.6 Information Catalogue Functional Element (new)

### 2.6.1 Motivation

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

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

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

This Functional Element fulfills the following requirements from the Functional Elements Requirements:

Primary Requirements

- PROCESS-200,
- PROCESS-201, and
- PROCESS-202.

Secondary Requirements

- PROCESS-203,
- PROCESS-204,
- PROCESS-205, and
- PROCESS-206.

### 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

### 2.6.3 Key Features

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

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

- 2269 3. The Functional Element MUST provide the capability to *execute basic searches* like  
2270 categorical, names, keywords on the catalogue information.  
2271 4. The Functional Element MUST provide the capability to return results formatted based on the  
2272 Catalogue Structure.

2273

2274 In addition, the following key features could be provided to enhance the Information Catalogue  
2275 Functional Element further:

- 2276 1. The Functional Element MAY provide the ability to enable secured access to catalogue  
2277 structure as well as catalogue information.  
2278 2. The Functional Element MAY provide the ability to present catalogue information in different  
2279 languages, i.e. multi-lingual support.  
2280 3. The Functional Element MAY provide the ability to import catalogue structure and information  
2281 from different data sources.  
2282 4. The Functional Element MAY provide the ability to export catalogue structure and information  
2283 to different data sources.

2284

## 2285 2.6.4 Interdependencies

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

2286

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.

## 2287 2.6.5 Related Technologies and Standards

2288 None

2289

## 2290 2.6.6 Model

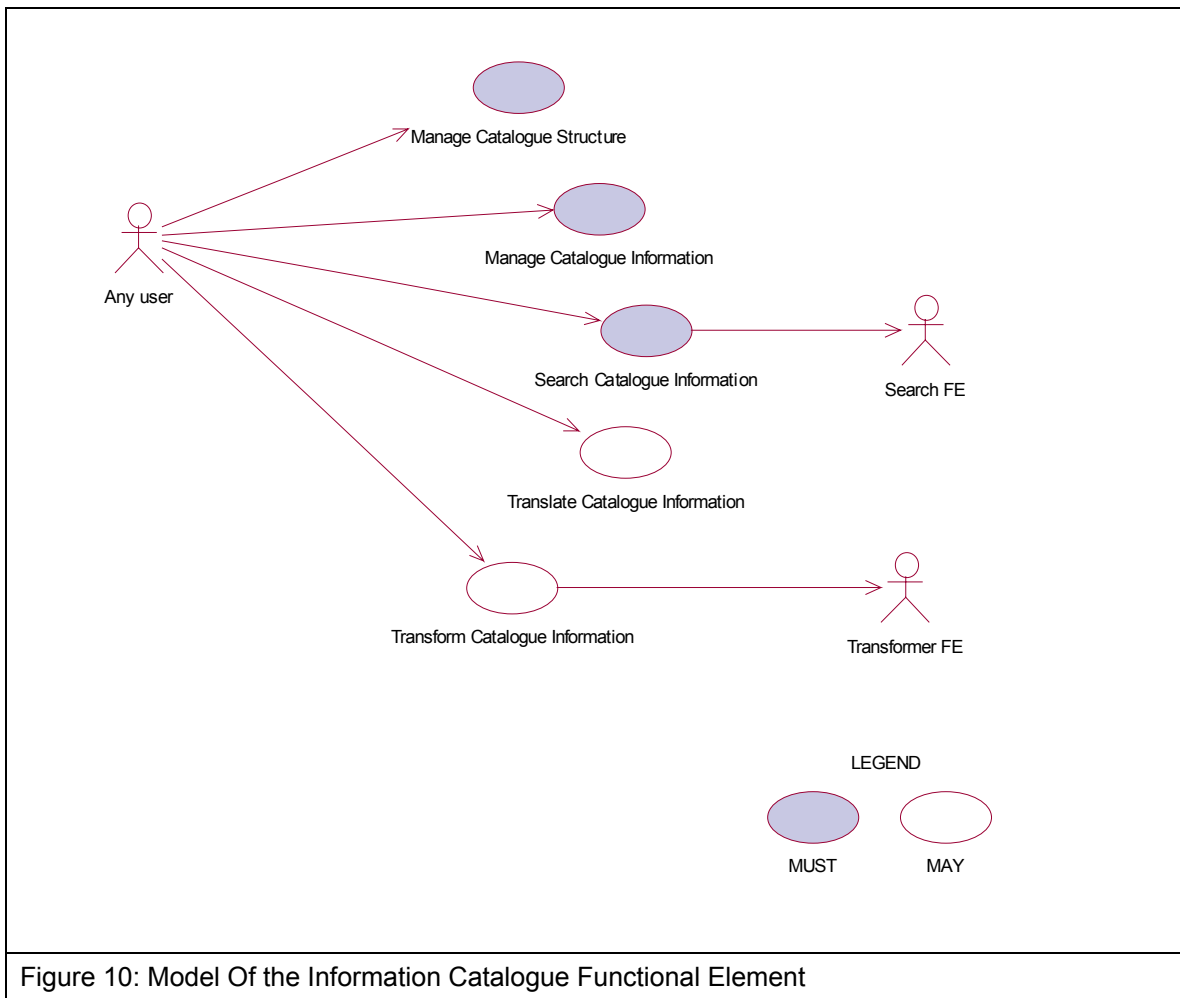


Figure 10: Model Of the Information Catalogue Functional Element

2291

## 2292 2.6.7 Usage Scenario

### 2293 2.6.7.1 Manage Catalogue Structure

#### 2294 2.6.7.1.1 Description

2295 This use case allows any users to configure and manage various data source(s), type(s) and  
 2296 structure(s) on which information is to be stored and retrieved.

#### 2297 2.6.7.1.2 Flow of Events

##### 2298 2.6.7.1.2.1 Basic Flow

2299 This use case starts when users / other Functional Elements wishes to configure and manage  
 2300 various data source(s), type(s) and structure(s).

2301 1. Users / Other Functional Elements initiates a request to configure data source, type and  
 2302 structure by providing name, format, and definition of the data source(s) to be added, removed or  
 2303 retrieved.

- 2304 2. The Functional Element checks whether the data source configuration file exists.
- 2305 3. Based on the operation it specified, one of the following sub-flows is executed:
- 2306 If the operation is '**Create Data Source, Type and Structure**', then sub-flow 3.1 is executed.
- 2307 If the operation is '**View Data Source, Type and Structure**', then sub-flow 3.2 is executed.
- 2308 If the operation is '**Remove Data Source, Type and Structure**', then sub-flow 3.3 is executed.
- 2309 3.1. Create Data Source, Type and Structure.
- 2310 3.1.1. The Functional Element checks whether the same data source, type, and structure
- 2311 has been created.
- 2312 3.1.2. The Functional Element appends the new data source, type and structure in the
- 2313 data source configuration specified.
- 2314 3.2. View Data Source, Type and Structure.
- 2315 3.2.1. The Functional Element retrieves all the data source, type and structure
- 2316 information from the data source configuration file.
- 2317 3.2.2. The Functional Element returns the data source(s), type(s) and structure(s).
- 2318 3.3. Delete Data Source, Type and Structure.
- 2319 3.3.1. The Functional Element checks whether the data source, type and structure exist
- 2320 in the data source configuration based on data source id from the data source
- 2321 configuration file.
- 2322 3.3.2. The Functional Element removes the old data source, type and structure from the
- 2323 data source configuration file.
- 2324 4. The Functional Element returns a success or failure flag indicating the status of the operation
- 2325 being performed and use case ends.
- 2326 **2.6.7.1.2.2 Alternative Flows**
- 2327 1. Data Source Configuration File Not Found.
- 2328 1.1. If in Basic Flow 2, the data source configuration does not exist, Functional Element
- 2329 creates empty data source configuration.
- 2330 2. Duplicate Data Source, Type and Structure.
- 2331 2.1. If in Sub Flow 3.1.1, the same data source, type and structure have been defined already
- 2332 in data source configuration, Functional Element throws an exception with error code as
- 2333 'Duplicate Data Source, Type, and Structure'.
- 2334 3. Data Source, Type, and Structure Do Not Exist.
- 2335 3.1. If in Sub Flow 3.2.1 and 3.3.1, a particular data source, type and structure cannot be
- 2336 found in the specified data source configuration, Functional Element throws an exception with
- 2337 error code as 'Data Source, Type, and Structure does not exist'.
- 2338 **2.6.7.1.3 Special Requirements**
- 2339 None.

2340     **2.6.7.1.4     Pre-Conditions**

2341     None.

2342     **2.6.7.1.5     Post-Conditions**

2343     None.

2344     **2.6.7.2   Manage Catalogue Information**

2345     **2.6.7.2.1     Description**

2346     This use case describes the management of catalogue information, namely the creation, deletion,  
2347     retrieval and update of the catalogue information.

2348     **2.6.7.2.2     Flow of Events**

2349     **2.6.7.2.2.1   Basic Flow**

2350     The use case begins when the user wants to create/view/update/delete catalogue information.

2351     1. The user sends request to manipulate catalogue information.

2352     2. Based on the operation it specifies, one of the following sub-flows is executed:

2353     If the operation is '**Create Catalogue Information**', the sub-flow 2.1 is executed.

2354     If the operation is '**View Catalogue Information**', the sub-flow 2.2 is executed.

2355     If the operation is '**Update Catalogue Information**', the sub-flow 2.3 is executed.

2356     If the operation is '**Delete Catalogue Information**', the sub-flow 2.4 is executed.

2357     2.1. Create Catalogue Information

2358         2.1.1. User provides the basic information that is necessary for creating catalogue  
2359         information.

2360         2.1.2. The Functional Element checks whether the catalogue information exists.

2361         2.1.3. The Functional Element creates the catalogue.

2362     2.2. View Catalogue Information

2363         2.2.1. User provides the necessary information for retrieving the complete catalogue's  
2364         attributes.

2365         2.2.2. The Functional Element checks whether the catalogue information exists.

2366         2.2.3. The Functional Element returns the catalogue's information.

2367     2.3. Update Catalogue Information

2368         2.3.1. User provides the necessary information for updating the catalogue's attributes.

2369         2.3.2. The Functional Element checks whether the catalogue information exists.

2370         2.3.3. The Functional Element updates the catalogue.

2371     2.4. Delete Catalogue Information

2372 2.4.1. User provides the necessary information for deleting particular catalogue  
2373 information.

2374 2.4.2. The Functional Element checks whether the catalogue information exists.

2375 2.4.3. Functional Element deletes the catalogue information.

2376 **2.6.7.2.2.2 Alternative Flows**

2377 1. Catalogue Information Exist.

2378 1.1. In Sub Flow 2.1.2, Function Element detects an identical catalogue information.  
2379 Functional Element returns an error message and the use case ends.

2380 2. Catalogue Information Does Not Exist.

2381 2.1. In Sub Flow 2.2.2, 2.3.2, and 2.4.2, Functional Element cannot find the catalogue  
2382 information that matches the user's criteria. Functional Element returns an error message  
2383 and the use case ends.

2384 3. Save Updated Catalogue Information.

2385 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  
2386 catalogue information. Functional Element returns an error message and the use case ends.

2387 **2.6.7.2.3 Special Requirements**

2388 None.

2389 **2.6.7.2.4 Pre-Conditions**

2390 None.

2391 **2.6.7.2.5 Post-Conditions**

2392 None.

2393 **2.6.7.3 Search Catalogue Information**

2394 **2.6.7.3.1 Description**

2395 This use case allows any users to perform search on various types of disparate catalogues that  
2396 are configured to be searched and returns the matching results.

2397 **2.6.7.3.2 Flow of Events**

2398 **2.6.7.3.2.1 Basic Flow**

2399 This use case starts when users / other Functional Elements wishes to perform information  
2400 search on any given catalogue.

2401 1. Users / other Functional Elements initiates a request to perform information search on a given  
2402 catalogue by providing information to be searched, the catalogue type(s) and the catalogue  
2403 structure(s).

2404 2. The Functional Element checks for the existence of the specified catalogue type(s) and  
2405 structure(s).

- 2406 3. The Functional Element validates the catalogue type(s) and structure(s) against the set of  
2407 supported data type(s) and structure(s) configured within the Functional Element that are  
2408 available for information search.
- 2409 4. The Functional Element performs information search based on the search parameters given by  
2410 the users or the other Functional Elements.
- 2411 5. The Functional Element returns the result of the information search performed to the users or  
2412 other Functional Elements and use case ends.

#### 2413 **2.6.7.3.2.2 Alternative Flows**

- 2414 1. Catalogue(s) Are Not Available.
- 2415 1.1. In Basic Flow 2, if the identified catalogue is not available, Functional Element displays  
2416 an error message and exits the use case.
- 2417 2. Invalid Catalogue Type and Structure.
- 2418 2.1. In Basic Flow 3, if the catalogue type and structure are invalid, Functional Element  
2419 displays catalogue type and structure failure message and prompts for the data source type  
2420 and structure again and performs another search.
- 2421 3. No Matching Result.
- 2422 3.1. In Basic Flow 4, if the search results in no matching results, Functional Element displays  
2423 a message "No search results found" and performs another search.

#### 2424 **2.6.7.3.3 Special Requirements**

2425 None.

#### 2426 **2.6.7.3.4 Pre-Conditions**

2427 None.

#### 2428 **2.6.7.3.5 Post-Conditions**

2429 None.

### 2430 **2.6.7.4 Translate Catalogue Information**

#### 2431 **2.6.7.4.1 Description**

2432 This use case allows the user to translate a catalogue information file from one language to  
2433 another language.

#### 2434 **2.6.7.4.2 Flow of Events**

##### 2435 **2.6.7.4.2.1 Basic Flow**

- 2436 This use case starts when a user wants to translate a catalogue information file from one  
2437 language to another language.
- 2438 1. The user set the file name to be translated and the destination language.
- 2439 2. The system checks whether the particular destination language as output can be translated  
2440 within all the supported translation methods available.

2441 4. Select the appropriate method based on the destination language.

2442 5. Invoke the translate method and save the catalogue information which is translated in that  
2443 particular destination language.

2444 6: Return the results and the use case ends.

2445 **2.6.7.4.2.2 Alternative Flows**

2446 1. If in Basic Flow 2 there is no method to do the translation, the system return error message to  
2447 the user and this use case ends.

2448 **2.6.7.4.3 Special Requirements**

2449 None.

2450 **2.6.7.4.4 Pre-Conditions**

2451 None.

2452 **2.6.7.4.5 Post-Conditions**

2453 None.

2454

2455 **2.6.7.5 Transform Catalogue Information**

2456 **2.6.7.5.1 Description**

2457 This use case allows the user to transform a catalogue information file from one format to another  
2458 format.

2459 **2.6.7.5.2 Flow of Events**

2460 **2.6.7.5.2.1 Basic Flow**

2461 This use case starts when a user wants to transform a catalogue information file from one format  
2462 to another format.

2463 1. The user set the file name to be transformed and the destination format.

2464 2. This use case call the TRANSFORMER functional elements' transform flow.

2465 3. Return the results from the transformer functional elements' transform flow and the use case  
2466 ends.

2467 **2.6.7.5.2.2 Alternative Flows**

2468 1. If in Basic Flow 2 there is no method to do the transformation, the system return error message  
2469 to the user and this use case ends.

2470 **2.6.7.5.3 Special Requirements**

2471 None.



2472    **2.6.7.5.4    Pre-Conditions**

2473    None.

2474    **2.6.7.5.5    Post-Conditions**

2475    None.

2476

2477 **2.7 Information Reporting Functional Element (new)**

2478 **2.7.1 Motivation**

2479 Information reporting is quite common in enterprise applications nowadays. In many scenarios,  
2480 an enterprise does need to present its business information to, for example, business partners,  
2481 sales representatives, and customers, in some form of information reporting. An information  
2482 report is filled with the data that is retrieved from a data source using some type of queries. Such  
2483 kind of information reporting is also used internally within an enterprise, or even within an  
2484 individual department, to verify the business performance and other business scenarios.

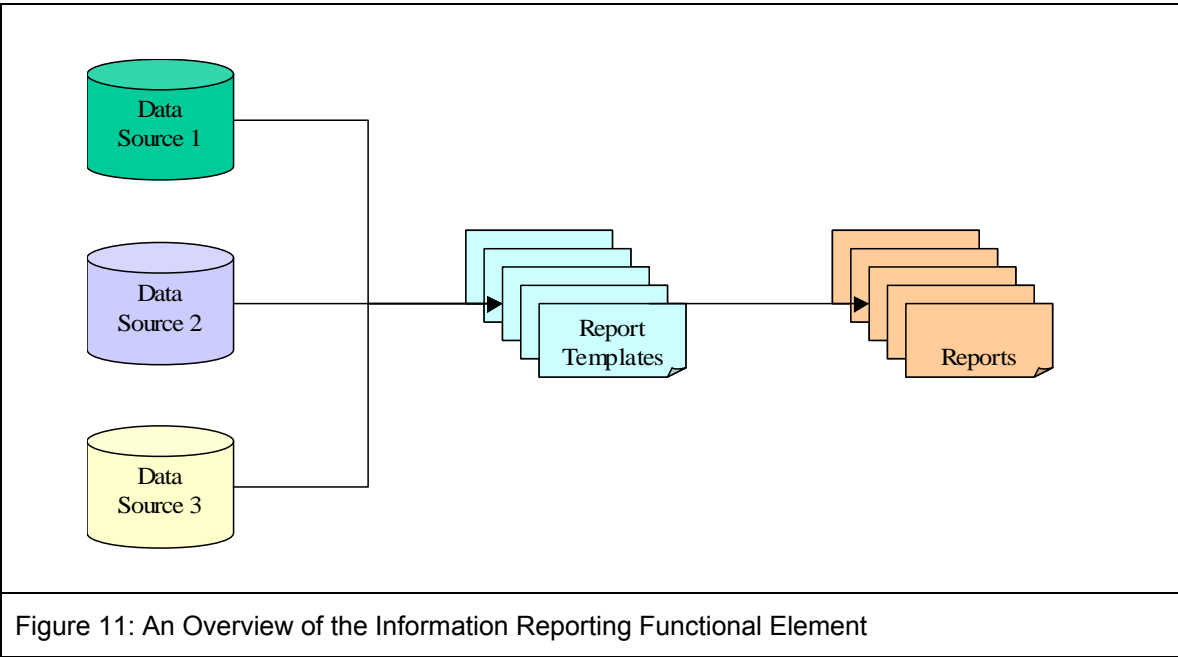


Figure 11: An Overview of the Information Reporting Functional Element

2486 This Functional Element aims to provide the core features of information reporting solution to be  
2487 used in general enterprise applications. It fulfills the following requirements from the Functional  
2488 Elements Requirements:

- 2490 • Primary Requirements:
  - 2491 • DELIVERY-100,
  - 2492 • DELIVERY-101,
  - 2493 • DELIVERY-102,
  - 2494 • DELIVERY-103, and
  - 2495 • DELIVERY-104.
- 2496 • Secondary Requirements:
  - 2497 • DELIVERY-105, and
  - 2498 • DELIVERY-106.

2499 **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.

2501

## 2502 2.7.3 Key Features

2503 Implementations of the Information Reporting Functional Element are expected to provide the  
2504 following key features:

- 2505 1. The Functional Element MUST provide an approach to capture the report templates and  
2506 provide the guidelines how to secure the report templates.
- 2507 2. The Functional Element MUST be able to generate reports in the format defined by report  
2508 templates.
- 2509 3. The Functional Element MUST provide a way to specify data sources where information is  
2510 retrieved to fill out the generated reports.
- 2511 4. The Functional Element MUST provide an approach to capture user-defined queries, and  
2512 MUST be able to execute user-defined queries to retrieve information to fill out the generated  
2513 reports.
- 2514 5. The Functional Element MUST be able to store and retrieve generated reports as stated in  
2515 key feature #2.
- 2516 6. The Functional Element MUST provide a security approach to control report access. A  
2517 considered approach is to use user, role, and access management.

2518

2519 In addition, the following key features could be provided to enhance the Information Reporting  
2520 Functional Element further:

- 2521 1. The Functional Element MAY provide an approach, such as an IDE, to design report  
2522 templates.
- 2523 2. The Functional Element MAY provide the capability to export reports to different electronic  
2524 file formats.
- 2525 3. The Functional Element MAY provide the capability to log the activities of report access.
- 2526 4. The Functional Element MAY allow the users to subscribe to the reports they want to  
2527 receive.

2528

## 2529 2.7.4 Interdependencies

Interaction Dependency	
Transformer Functional Element	The Transformer Functional Element helps to provide the import and export report information capabilities.
Notification Functional Element	The Notification Functional Element helps to send SMS / email to the appropriate Report Subscriber.

2530 **2.7.5 Related Technologies and Standards**

2531 None.

2533 **2.7.6 Model**

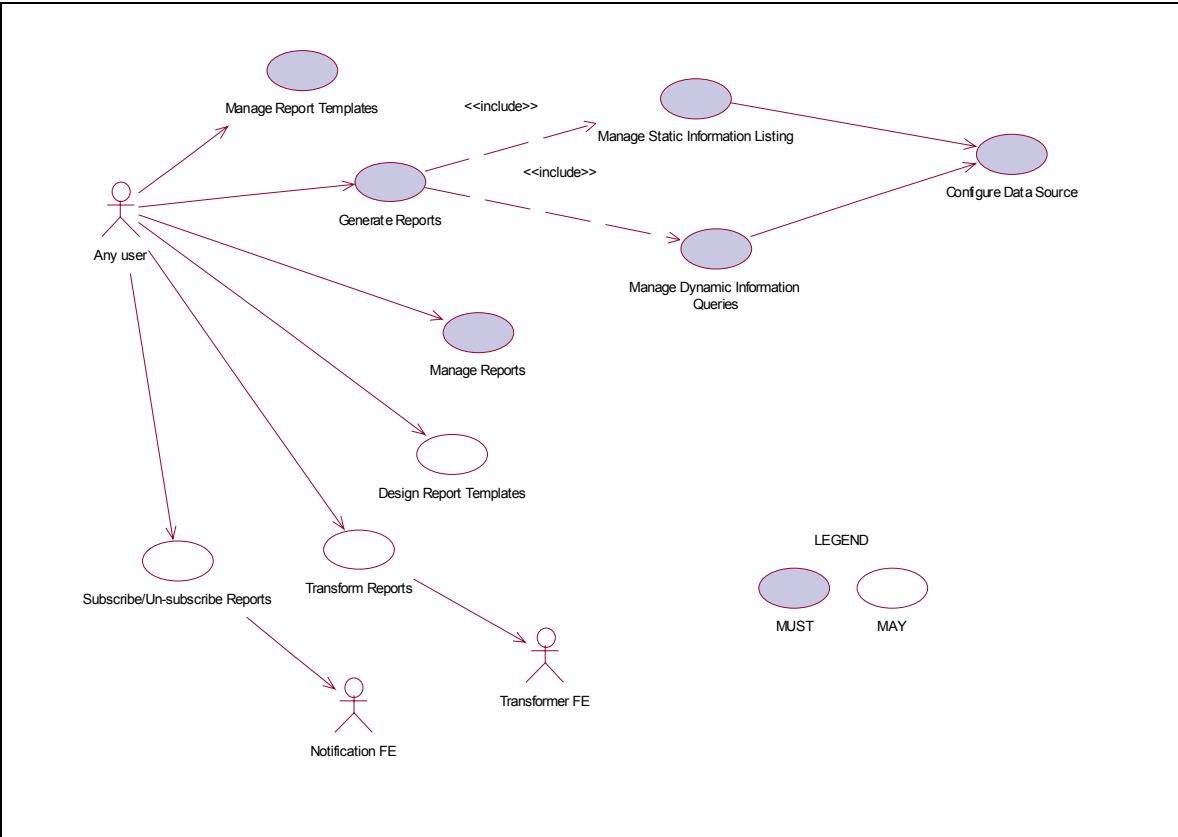


Figure 12: Model Of the Information Reporting Functional Element

2535 **2.7.7 Usage Scenario**

2536 **2.7.7.1 Manage Report Templates**

2537 **2.7.7.1.1 Description**

2538 This use case allows any users to create, update, remove and view reporting templates.

2539 **2.7.7.1.2 Flow of Events**

2540 **2.7.7.1.2.1 Basic Flow**

2541 The use case begins when the user wants to create/view/update/delete reporting templates.

2542 1: Any user initiates a request type to the Functional Element stating whether to create, view,  
2543 update, or delete reporting templates.

2544 2: The Functional Element checks whether the reporting template exists.

2545 3: Based on the operation it specified, one of the following sub-flows is executed:

- 2546 • If the operation is '**Create Reporting Template**', then sub-flow 3.1 is executed.
- 2547 • If the operation is '**View Reporting Template**', then sub-flow 3.2 is executed.
- 2548 • If the operation is '**Update Reporting Template**', then sub-flow 3.3 is executed.
- 2549 • If the operation is '**Delete Reporting Template**', then sub-flow 3.4 is executed.

2550 3.1: Create Reporting Template.

2551 3.1.1: Any user provides reporting template information to be created.

2552 3.1.2: The Functional Element checks for the duplicate reporting template information.

2553 3.1.3: The Functional Element creates the reporting template information, if it does not  
2554 exist and the use case ends.

2555 3.2: View Reporting Template.

2556 3.2.1: The Functional Element retrieves all the reporting templates.

2557 3.2.2: The Functional Element returns the reporting template information to any user and  
2558 the use case ends.

2559 3.3: Update Reporting Template.

2560 3.3.1: Any user provides reporting template information to be updated.

2561 3.3.2: The Functional Element checks for the availability of reporting template  
2562 information.

2563 3.3.3: The Functional Element updates the reporting template information, if it exist and  
2564 the use case ends.

2565 3.4: Delete Reporting Template.

2566 3.4.1: Any user provides reporting template information to be removed.

2567 3.4.2: The Functional Element removes the reporting template information.

2568 4: The Functional Element responses the status of the operation whether it is successful or failure  
2569 to any user and the use case ends.

#### 2570 **2.7.7.1.2.2 Alternative Flows**

2571 1: Reporting Template Information Not Found.

2572 1.1: In the Sub Flow 3.2.1, 3.3.2, & 3.4.1, if the reporting template information cannot be  
2573 found, Functional Element throws exception with error code as 'Reporting Template does not  
2574 exist'.

2575 2: Duplicate Reporting Template Information.

2576 2.1: In the Sub Flow 3.1.2, If the same reporting template information has been defined,  
2577 Functional Element throws exception with error code as 'Duplicate reporting template  
2578 information'.

2579     **2.7.7.1.3     Special Requirements**

2580     None.

2581     **2.7.7.1.4     Pre-Conditions**

2582     None.

2583     **2.7.7.1.5     Post-Conditions**

2584     None.

2585

2586     **2.7.7.2   Generate Reports**

2587     This use case allows any user to generate reports, which includes Static Information Listing and  
2588     Dynamic Information Queries.

2589     **2.7.7.2.1     Flow of Events**

2590     **2.7.7.2.1.1   Basic Flow**

2591     This use case starts when the user of the data source wishes to generate reports that include  
2592     Static Information Listing and Dynamic Information Queries.

2593     1: Any user initiates a request type to the Functional Element stating whether to generate reports  
2594     that includes Static Information Listing or Dynamic Information Queries.

2595     2: Based on the operation it specified, one of the following basic flows is executed:

- 2596         • If the operation is 'Manage Static Information Listing', then Manage Static Information  
2597         Listing Basic Flow is executed.
- 2598         • If the operation is 'Manage Dynamic Information Queries', then Manage Dynamic  
2599         Information Queries Basic Flow is executed.

2600     3: Whenever a report is generated using a particular reporting template, the respective report  
2601     subscribers are notified via email using NOTIFICATION Functional Element and the use case  
2602     ends.

2603

2604     **2.7.7.3   Manage Static Information Listing**

2605     **2.7.7.3.1     Description**

2606     This use case allows any users to create, view, update, and delete Static Information Listing.

2607     **2.7.7.3.2     Flow of Events**

2608     **2.7.7.3.2.1   Basic Flow**

2609     This use case starts when the users of the data source wishes to create, view, update, and delete  
2610     Static Information Listing.

2611 1: Any user initiates a request type to the Functional Element stating whether to create, view,  
2612 update, or delete Static Information Listing.

2613 2: The Functional Element checks whether the Static Information Listing exists.

2614 3: Based on the operation it specified, one of the following sub-flows is executed:

- 2615 • If the operation is '**Create Static Information Listing**', then sub-flow 3.1 is executed.
- 2616 • If the operation is '**View Static Information Listing**', then sub-flow 3.2 is executed.
- 2617 • If the operation is '**Update Static Information Listing**', then sub-flow 3.3 is executed.
- 2618 • If the operation is '**Delete Static Information Listing**', then sub-flow 3.4 is executed.

2619 3.1: Create Static Information Listing.

2620 3.1.1: Any user provides Static Information Listing to be created.

2621 3.1.2: The Functional Element checks for the duplicate Static Information Listing.

2622 3.1.3: The Functional Element creates the Static Information Listing, if it does not exist and the  
2623 use case ends.

2624 3.2: View Static Information Listing.

2625 3.2.1: The Functional Element retrieves all the Static Information Listing.

2626 3.2.2: The Functional Element returns the Static Information Listing to any user and the use case  
2627 ends.

2628 3.3: Update Static Information Listing.

2629 3.3.1: Any user provides Static Information Listing to be updated.

2630 3.3.2: The Functional Element checks for the availability of Static Information Listing.

2631 3.3.3: The Functional Element updates the Static Information Listing, if it exist and the use case  
2632 ends.

2633 3.4: Delete Static Information Listing.

2634 3.4.1: Any user provides Static Information Listing to be removed.

2635 3.4.2: The Functional Element removes the Static Information Listing.

2636 4: The Functional Element responses the status of the operation whether it is successful or failure  
2637 to any user and the use case ends.

#### 2638 **2.7.7.3.2.2 Alternative Flows**

2639 1: Static Information Listing Not Found.

2640 1.1: In the Sub Flow 3.2.1, 3.3.2, & 3.4.1, if the Static Information Listing cannot be found,  
2641 Functional Element throws exception with error code as 'Static Information Listing does not  
2642 exist'.

2643 2: Duplicate Static Information Listing.

2644 2.1: In the Sub Flow 3.1.2, If the same Static Information Listing has been defined, Functional  
2645 Element throws exception with error code as 'Duplicate Static Information Listing'.

2646     **2.7.7.3.3     Special Requirements**

2647     This use case requires the following three elements:

- 2648
  - A data source
  - 2649         • A static information query
  - 2650         • A reporting template

2651     **2.7.7.3.4     Pre-Conditions**

2652     None.

2653     **2.7.7.3.5     Post-Conditions**

2654     None.

2655

2656     **2.7.7.4    Manage Dynamic Information Queries**

2657     **2.7.7.4.1     Description**

2658     This use case allows any users to create, view, update, and delete dynamic information queries.

2659     **2.7.7.4.2     Flow of Events**

2660     This use case starts when the users of the data source wishes to create, view, update, or delete  
2661     dynamic information queries.

2662     1: Any user initiates a request type to the Functional Element stating whether to create, view,  
2663     update, or delete Dynamic Information Queries.

2664     2: The Functional Element checks whether the Dynamic Information Query exists.

2665     3: Based on the operation it specified, one of the following sub-flows is executed:

- 2666
  - If the operation is '**Create Dynamic Information Query**', then sub-flow 3.1 is executed.
  - 2667         • If the operation is '**View Dynamic Information Query**', then sub-flow 3.2 is executed.
  - 2668         • If the operation is '**Update Dynamic Information Query**', then sub-flow 3.3 is executed.
  - 2669         • If the operation is '**Delete Dynamic Information Query**', then sub-flow 3.4 is executed.

2670     3.1: Create Dynamic Information Query.

2671         3.1.1: Any user provides Dynamic Information Query to be created.

2672         3.1.2: The Functional Element checks for the duplicate Dynamic Information Query.

2673         3.1.3: The Functional Element creates the Dynamic Information Query, if it does not exist  
2674         and the use case ends.

2675     3.2: View Dynamic Information Query.

2676         3.2.1: The Functional Element retrieves all the Dynamic Information Queries.

2677         3.2.2: The Functional Element returns the Dynamic Information Query to any user and  
2678         the use case ends.



2679           3.3: Update Dynamic Information Query.

2680           3.3.1: Any user provides Dynamic Information Query to be updated.

2681           3.3.2: The Functional Element checks for the availability of Dynamic Information Query.

2682           3.3.3: The Functional Element updates the Dynamic Information Query, if it exist and the  
2683           use case ends.

2684           3.4: Delete Dynamic Information Query.

2685           3.4.1: Any user provides Dynamic Information Query to be removed.

2686           3.4.2: The Functional Element removes the Dynamic Information Query.

2687           4: The Functional Element responses the status of the operation whether it is successful or failure  
2688           to any user and the use case ends.

2689           **2.7.7.4.2.1 Alternative Flows**

2690           1: Dynamic Information Query Not Found.

2691           1.1: In the Sub Flow 3.2.1, 3.3.2, & 3.4.1, if the Dynamic Information Query cannot be found,  
2692           Functional Element throws exception with error code as 'Dynamic Information Query does not  
2693           exist'.

2694           2: Duplicate Dynamic Information Query.

2695           2.1: In the Sub Flow 3.1.2, If the same Dynamic Information Query has been defined,  
2696           Functional Element throws exception with error code as 'Duplicate Dynamic Information  
2697           Query'.

2698           **2.7.7.4.3 Special Requirements**

2699           This use case requires the following three elements:

2700           • A data source

2701           • A dynamic information query

2702           • A reporting template

2703           **2.7.7.4.4 Pre-Conditions**

2704           None.

2705           **2.7.7.4.5 Post-Conditions**

2706           None.

2707

2708           **2.7.7.5 Manage Reports**

2709           **2.7.7.5.1 Description**

2710           This use case allows any users to view, update, and delete reports.

2711     **2.7.7.5.2     Flow of Events**

2712     **2.7.7.5.2.1 Basic Flow**

2713     This use case starts when the users of the data source wishes to view, update, or delete reports.

2714     1: Any user initiates a request type to the Functional Element stating whether to view, update, or  
2715     delete reports.

2716     2: The Functional Element checks whether the report exists.

2717     3: Based on the operation it specified, one of the following sub-flows is executed:

- 2718         • If the operation is '**View Report**', then sub-flow 3.1 is executed.
- 2719         • If the operation is '**Update Report**', then sub-flow 3.2 is executed.
- 2720         • If the operation is '**Delete Report**', then sub-flow 3.3 is executed.

2721     3.1: View Report.

2722         3.1.1: The Functional Element retrieves all the reports.

2723         3.1.2: The Functional Element returns the report information to any user and the use  
2724         case ends.

2725     3.2: Update Report.

2726         3.2.1: Any user provides report information to be updated.

2727         3.2.2: The Functional Element checks for the availability of report information.

2728         3.2.3: The Functional Element updates the report information, if it exist and the use case  
2729         ends.

2730     3.3: Delete Report.

2731         3.3.1: Any user provides report information to be removed.

2732         3.3.2: The Functional Element removes the report information.

2733     4: The Functional Element responses the status of the operation whether it is successful or failure  
2734     to any user and the use case ends.

2735     **2.7.7.5.2.2 Alternative Flows**

2736     1: Report Information Not Found.

2737         1.1: In the Sub Flow 3.1.1, 3.2.2, & 3.3.1, if the report information cannot be found, Functional  
2738         Element throws exception with error code as 'Report does not exist'.

2739     **2.7.7.5.3     Special Requirements**

2740     None.

2741     **2.7.7.5.4     Pre-Conditions**

2742     None.

2743     **2.7.7.5.5     Post-Conditions**

2744     None.

2745

2746     **2.7.7.6   Configure Data Source**

2747     **2.7.7.6.1     Description**

2748     This use case allows any users to create, view, update, and delete data source.

2749     **2.7.7.6.2     Flow of Events**

2750     **2.7.7.6.2.1   Basic Flow**

2751     This use case starts when the users of the data source wishes to create, view, update, or delete  
2752     data source.

2753     1: Any user initiates a request type to the Functional Element stating whether to create, view,  
2754     update, or delete source.

2755     2: The Functional Element checks whether the data source exists.

2756     3: Based on the operation it specified, one of the following sub-flows is executed:

- 2757         • If the operation is '**Create Data Source**', then sub-flow 3.1 is executed.
- 2758         • If the operation is '**View Data Source**', then sub-flow 3.2 is executed.
- 2759         • If the operation is '**Update Data Source**', then sub-flow 3.3 is executed.
- 2760         • If the operation is '**Delete Data Source**', then sub-flow 3.4 is executed.

2761     3.1: Create Data Source.

2762         3.1.1: Any user provides data source information to be created.

2763         3.1.2: The Functional Element checks for the duplicate data source information.

2764         3.1.3: The Functional Element creates the data source information, if it does not exist and the use  
2765         case ends.

2766     3.2: View Data Source.

2767         3.2.1: The Functional Element retrieves all the data sources.

2768         3.2.2: The Functional Element returns the data source information to any user and the use case  
2769         ends.

2770     3.3: Update Data Source.

2771         3.3.1: Any user provides data source information to be updated.

2772         3.3.2: The Functional Element checks for the availability of data source information.

2773         3.3.3: The Functional Element updates the data source information, if it exist and the use case  
2774         ends.

2775     3.4: Delete Data Source.

2776 3.4.1: Any user provides data source information to be removed.

2777 3.4.2: The Functional Element removes the data source information.

2778 4: The Functional Element responses the status of the operation whether it is successful or failure  
2779 to any user and the use case ends.

2780 **2.7.7.6.2 Alternative Flows**

2781 1: Data Source Information Not Found.

2782 1.1: In the Sub Flow 3.2.1, 3.3.2, & 3.4.1, if the data source information cannot be found,  
2783 Functional Element throws exception with error code as 'Data source does not exist'.

2784 2: Duplicate Data Source Information.

2785 2.1: In the Sub Flow 3.1.2, If the same data source information has been defined, Functional  
2786 Element throws exception with error code as 'Duplicate data source information'.

2787 **2.7.7.6.3 Special Requirements**

2788 None.

2789 **2.7.7.6.4 Pre-Conditions**

2790 None.

2791 **2.7.7.6.5 Post-Conditions**

2792 None.

2793

2794 **2.7.7.7 Design Report Templates**

2795 **2.7.7.7.1 Description**

2796 This use case allows any users to design reporting templates.

2797 **2.7.7.7.2 Flow of Events**

2798 **2.7.7.7.2.1 Basic Flow**

2799 The use case begins when the user wants to design reporting templates.

2800 1: Any user provides reporting template information to be designed.

2801 2: The Functional Element checks for the duplicate reporting template information designed.

2802 3: The Functional Element designs and saves the reporting template information, if it does not  
2803 exist and the use case ends.

2804 **2.7.7.7.2.2 Alternative Flows**

2805 1: Duplicate Reporting Template Design Information.

2806 1.1: In the Basic Flow 2, if the same reporting template information has been designed,  
2807 Functional Element throws exception with error code as 'Duplicate reporting template design  
2808 information'.

2809 **2.7.7.7.3 Special Requirements**

2810 None.

2811 **2.7.7.7.4 Pre-Conditions**

2812 None.

2813 **2.7.7.7.5 Post-Conditions**

2814 None.

2815

2816 **2.7.7.8 Transform Reports**

2817 **2.7.7.8.1 Description**

2818 This use case allows the user to transform a report information file from one format to another  
2819 format.

2820 **2.7.7.8.2 Flow of Events**

2821 **2.7.7.8.2.1 Basic Flow**

2822 This use case starts when a user wants to transform a report information file from one format to  
2823 another format.

2824 1: The user set the file name to be transformed and the destination format.

2825 2: This use case call the TRANSFORMER functional elements' transform flow.

2826 3: Return the results from the transformer functional elements' transform flow and the use case  
2827 ends.

2828 **2.7.7.8.2.2 Alternative Flows**

2829 1: If in Basic Flow 2 there is no method to do the transformation, the system return error message  
2830 to the user and this use case ends.

2831 **2.7.7.8.3 Special Requirements**

2832 None.

2833 **2.7.7.8.4 Pre-Conditions**

2834 None.

2835     **2.7.7.8.5     Post-Conditions**

2836     None.

2837

2838     **2.7.7.9   Subscribe/Un-subscribe Reports**

2839     **2.7.7.9.1     Description**

2840     This use case performs the subscription or un-subscription on desired reports for any user.

2841     **2.7.7.9.2     Flow of Events**

2842     **2.7.7.9.2.1   Basic Flow**

2843     The use case begins when the user wants to subscribe or un-subscribe those desired reports.

2844     1: The user sends a request.

2845     2: Based on the operation it specifies, one of the following sub-flows is executed:

- 2846         • If the operation is '**Subscribe to Report**', then sub-flow 2.1 is executed.
- 2847         • If the operation is '**Un-Subscribe to Report**', then sub-flow 2.2 is executed.

2848     2.1: Subscribe To Report.

2849         2.1.1: The Functional Element gets user id, together with those desired report name.

2850         2.1.2: The Functional Element checks whether the report exists.

2851         2.1.3: The Functional Element adds the subscription of the user to the report.

2852     2.2: Un-Subscribe To Report.

2853         2.2.1: The Functional Element gets user id, together with those desired report name.

2854         2.2.2: The Functional Element checks whether the report exists.

2855         2.2.3: The Functional Element removes the subscription of the user to the report.

2856     3: The Functional Element returns the results of the operation to the user and the use case ends.

2857     **2.7.7.9.2.2   Alternative Flows**

2858     1: Report Not Found.

2859         1.1: If in the basic flow 2.1.2 and 2.2.2, the report specified does not exist, Functional  
2860         Element will return an error message to the user and the use case ends.

2861     2: User Not Found.

2862         2.1: If in the basic flow 2.1.2 and 2.2.2, the user related does not exist, Functional Element  
2863         will return an error message to the user and the use case ends.

2864     **2.7.7.9.3     Special Requirements**

2865     None.

2866    **2.7.7.9.4    Pre-Conditions**

2867    None.

2868    **2.7.7.9.5    Post-Conditions**

2869    None.

## 2.8 Key Management Functional Element (new)

### 2.8.1 Motivation

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

This Functional Element fulfills the following requirements from the Functional Elements Requirements, Working Draft Version 2.0 (fws-fe-2.0-requirements-doc-wd-01.doc):

Primary Requirement

- SECURITY-010.

### 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.

### 2.8.3 Key Features

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

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



- 2893 3. The Functional Element MUST provide the capability to recover a registered key or key pair  
2894 with an XKMS-compliant service.
- 2895 4. The Functional Element MUST provide the capability to retrieve a public key registered with  
2896 an XKMS-compliant service. The public can in turn be used to encrypt a document or verify  
2897 a signature.
- 2898 5. The Functional Element MUST provide the capability to ensure that a public key registered  
2899 with an XKMS-compliant service is valid and has not expired or been revoked.
- 2900
- 2901 In addition, the following key features could be provided to enhance the Functional Element  
2902 further:
- 2903 1. The Functional Element MAY provide the capability to generate key pairs.
- 2904

## 2905 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.

2906

## 2907 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 <sup>th</sup> 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 <sup>th</sup> 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).
---	---

### 2.8.6 Model

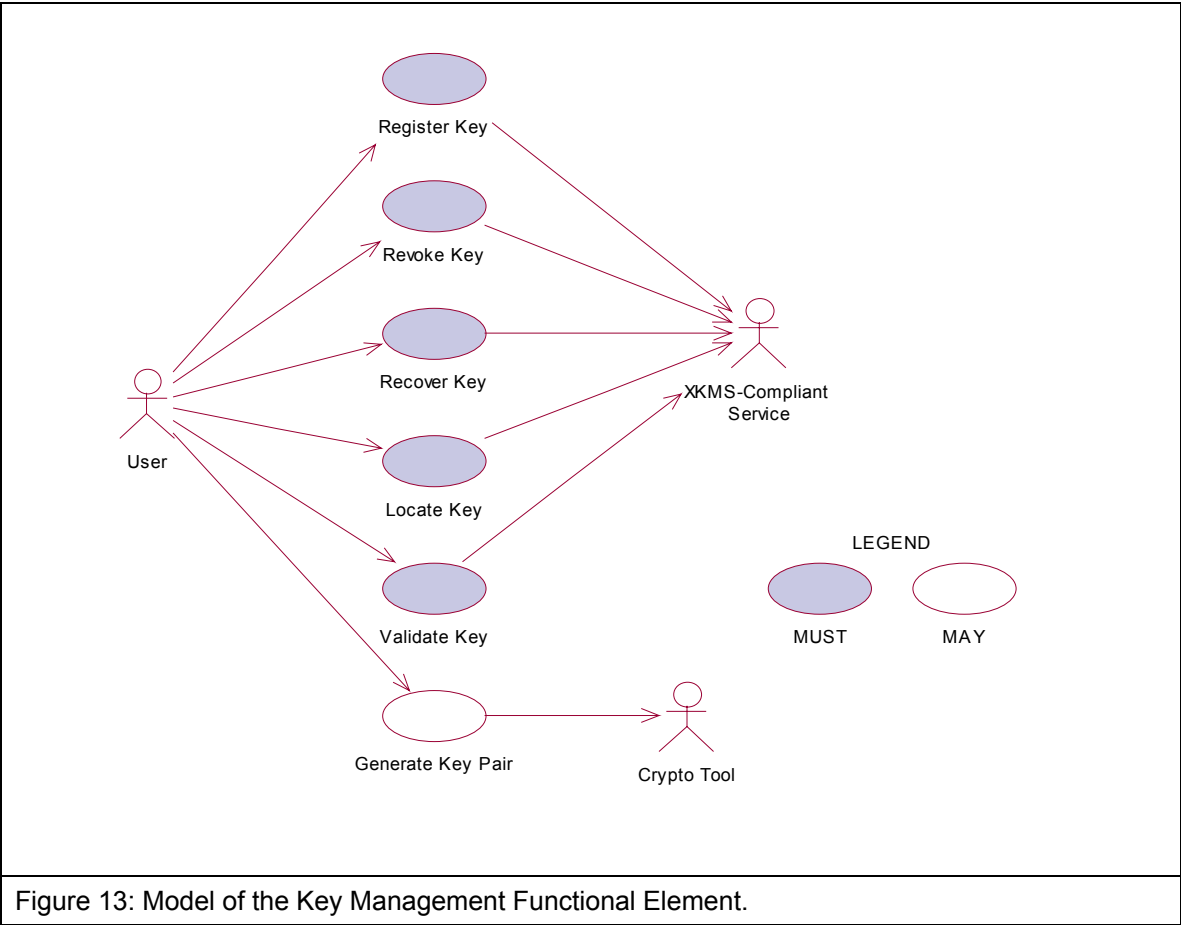


Figure 13: Model of the Key Management Functional Element.

## 2911 **2.8.7 Usage Scenarios**

### 2912 **2.8.7.1 Register Key**

#### 2913 **2.8.7.1.1 Description**

2914 This use case allows any user to register a key or key pair with a XKMS-compliant service.

#### 2915 **2.8.7.1.2 Flow of Events**

##### 2916 **2.8.7.1.2.1 Basic Flow**

2917 This use case starts when any user wants to register a key or key pair with a XKMS-compliant  
2918 service. The register request is used to assert a binding of information to a public key pair.  
2919 Generation of the public key pair MAY be performed by either the client or the XKMS-compliant  
2920 service.

2921 1: The user sends request to register a key or key pair by providing necessary registering  
2922 information, which include key information, a prototype of the requested assertion, optional  
2923 additional information to authenticate the user. If the public key pair to be registered is generated  
2924 by the user, the user may provide Proof of Possession of the private key.

2925 2: On receipt of a registering request from the user, the functional element transforms the request  
2926 to X-KRSS request format and sends to targeted XKMS-compliant service.

2927 3: The XKMS-compliant service verifies the authentication and Proof of Possession information  
2928 provided if any. If the service accepts the request, an assertion is registered. The service returns  
2929 part or all of the registered assertion in format of X-KRSS to the functional element.

2930 4: The Functional Element passes the response from the service to the user and the use case  
2931 ends.

##### 2932 **2.8.7.1.2.2 Alternative Flows**

2933 1: Information Not Enough.

2934 1.1: If in the basic flow 2, Functional Element detects the information provided by the user is  
2935 not enough to form a X-KRSS request, Functional Element returns general error message  
2936 and ends the use case.

2937 2: POP Needed.

2938 2.1: If in the basic flow 2, Functional Element checks that key pair is generated but the POP  
2939 is not provided by the user in the request message, the Functional Element returns an error  
2940 and ends the use case.

#### 2941 **2.8.7.1.3 Special Requirements**

#### 2942 **2.8.7.1.4 Pre-Conditions**

2943 None.

#### 2944 **2.8.7.1.5 Post-Conditions**

2945 None.

2946

## 2947 **2.8.7.2 Revoke Key**

### 2948 **2.8.7.2.1 Description**

2949 The use case allows any user to revoke previously issued assertions.

### 2950 **2.8.7.2.2 Flow of Events**

#### 2951 **2.8.7.2.2.1 Basic Flow**

2952 This use case starts when any user wants to revoke previous issued assertions.

2953 1: The user sends request to revoke a key or key pair by providing information, which include key  
2954 information, a prototype of the requested assertion, optional additional information to authenticate  
2955 the user. If the public key pair to be registered is generated by the user, the user may provide  
2956 Proof of Possession of the private key.

2957 2: On receipt of a revoking request from the user, the Functional Element transforms the request  
2958 to X-KRSS request format and sends to targeted XKMS-compliant service.

2959 3: The XKMS-compliant service verifies the authentication and Proof of Possession information  
2960 provided if any. If the service accepts the request, an assertion is revoked. The service returns  
2961 response in X-KRSS to indicate that the assertion is revoked.

2962 4: The Functional Element passes the response from the service to the user and the use case  
2963 ends.

### 2964 **2.8.7.2.3 Alternative Flows**

2965 1: Information Not Enough.

2966 1.1: If in the basic flow 2, Functional Element detects the information provided by the user is  
2967 not enough to form an X-KRSS request, Functional Element returns general error message  
2968 and ends the use case.

2969 2: POP Needed.

2970 2.1: If in the basic flow 2, Functional Element checks that key pair is generated but the POP  
2971 is not provided by the user in the request message, the Functional Element returns an error  
2972 and ends the use case.

### 2973 **2.8.7.2.4 Special Requirements**

2974 None.

### 2975 **2.8.7.2.5 Pre-Conditions**

2976 None.

### 2977 **2.8.7.2.6 Post-Conditions**

2978 If the use case was successful, the assertion issued previously would be revoked.

2979

2980

2981 **2.8.7.3 Recover Key**

2982 This use case allows any user to recover previously issued assertions.

2983 **2.8.7.3.1 Flow of Events**

2984 **2.8.7.3.1.1 Basic Flow**

2985 This use case starts when any user wants to recover previous issued assertions.

2986 1: The user sends request to recover a key or key pair by providing information, which include  
2987 key information, a prototype of the requested assertion, optional additional information to  
2988 authenticate the user. If the public key pair to be registered is generated by the user, the user  
2989 may provide Proof of Possession of the private key.

2990 2: On receipt of a recover request from the user, the Functional Element transforms the request  
2991 to X-KRSS request format and sends to targeted XKMS-compliant service.

2992 3: The XKMS-compliant service verifies the authentication and Proof of Possession information  
2993 provided if any. If the service accepts the request, an assertion is recovered. The service returns  
2994 response in X-KRSS to indicate that the assertion is recovered.

2995 4: The Functional Element passes the response from the service to the user and the use case  
2996 ends.

2997 **2.8.7.3.1.2 Alternative Flows**

2998 1: Information Not Enough.

2999 1.1: If in the basic flow 2, Functional Element detects the information provided by the user is  
3000 not enough to form an X-KRSS request, Functional Element returns general error message  
3001 and ends the use case.

3002 2: POP Needed.

3003 2.1: If in the basic flow 2, Functional Element checks that key pair is generated but the POP  
3004 is not provided by the user in the request message, the Functional Element returns an error  
3005 and ends the use case.

3006 **2.8.7.3.2 Special Requirements**

3007 None.

3008 **2.8.7.3.3 Pre-Conditions**

3009 None.

3010 **2.8.7.3.4 Post-Conditions**

3011 If the use case successes, the registered assertion is recovered in the XKMS-compliant service.

3012

3013     **2.8.7.4 Locate Key**

3014     **2.8.7.4.1 Description**

3015     This use case allows users to retrieve a public key registered with an XKMS-compliant service.  
3016     The public key can be in turn be used to encrypt a document or verify a signature.

3017     **2.8.7.4.1.1 Basic Flow**

3018     This use case starts when any user wants to retrieve a public key registered with an XKMS-  
3019     compliant service.

3020     1: The user sends request to retrieve a public key registered with an XKMS-compliant service by  
3021     providing related information.

3022     2: On receipt of a recover request from the user, the Functional Element transforms the request  
3023     to X-KISS request format and sends to targeted XKMS-compliant service.

3024     3: The XKMS-compliant service may obtain an X509V3 certificate. The certificate is parsed to  
3025     obtain the public key value that is return to the Functional Element in the format of X-KISS.

3026     4: The Functional Element checks the response message is issued by the target XKMS-compliant  
3027     service; ensures that the response message has not been modified; and confirms that the  
3028     response message corresponds to the request that made by the user.

3029     5: The Functional Element passes the response from the service to the user and the use case  
3030     ends.

3031     **2.8.7.4.1.2 Alternative Flows**

3032     1: Information Not Enough.

3033         1.1: If in the basic flow 2, Functional Element detects the information provided by the user is  
3034         not enough to form an X-KISS request, Functional Element returns general error message  
3035         and ends the use case.

3036     2: Fault Response.

3037         2.1: If in basic flow 4, Functional Element detects the response message has problem in  
3038         authenticity, integrity and does not correspond to the request, Functional Element returns  
3039         general error message and ends the use case.

3040     **2.8.7.4.2 Special Requirements**

3041     None.

3042     **2.8.7.4.3 Pre-Conditions**

3043     None.

3044     **2.8.7.4.4 Post-Conditions**

3045     None.

3046

3047     **2.8.7.5 Validate Key**

3048     This use case enables the user to obtain an assertion specifying the status of the binding  
3049     between the public key and other data, for example a name or a set of extended attributes.

3050     **2.8.7.5.1 Flow of Events**

3051     **2.8.7.5.1.1 Basic Flow**

3052     This use case starts when the user wants to obtain the status of the binding of a public key with  
3053     an assertion.

3054     1: The user sends request to validate a key or key pair by providing necessary validating  
3055     information defined in X-KISS, which include key information, a prototype of the requested  
3056     assertion, optional additional information to authenticate the user. If the public key pair to be  
3057     registered is generated by the user, the user may provide Proof of Possession of the private key.

3058     2: On receipt of a registering request from the user, the Functional Element transforms the  
3059     request to XKRSS request format and sends to targeted XKMS-compliant service.

3060     3: The XKMS-compliant service verifies the authentication and Proof of Possession information  
3061     provided if any. If the service accepts the request, an assertion is registered. The service returns  
3062     part or all of the registered assertion in format of XKRSS to the functional element.

3063     4: The Functional Element checks the response message is issued by the target XKMS-compliant  
3064     service; ensures that the response message has not been modified; and confirms that the  
3065     response message corresponds to the request that made by the user.

3066     5: The Functional Element passes the response from the service to the user and the use case  
3067     ends.

3068     **2.8.7.5.1.2 Alternative Flows**

3069     1: Information Not Enough.

3070         1.1: If in the basic flow 2, Functional Element detects the information provided by the user is  
3071         not enough to form an X-KISS request, Functional Element returns general error message  
3072         and ends the use case.

3073     2: Fault Response.

3074         2.1: If in basic flow 4, Functional Element detects the response message has problem in  
3075         authenticity, integrity and does not correspond to the request, Functional Element returns  
3076         general error message and ends the use case.

3077     **2.8.7.5.2 Special Requirements**

3078     None.

3079     **2.8.7.5.3 Pre-Conditions**

3080     None.

3081     **2.8.7.5.4 Post-Conditions**

3082     None.

3083

3084     **2.8.7.6 Generate Key Pair**

3085     This use case enables the user to generate key pair using the desired cryptographic tool.

3086     **2.8.7.6.1 Flow of Events**

3087     **2.8.7.6.1.1 Basic Flow**

3088     This use case starts when the user wants to obtain generate key pair using the desired  
3089     cryptographic tool.

3090     1: The user sends request to generate key pair by specifying related information.

3091     2: On receipt of request from the user, the functional element validates the provided information  
3092     and dispatch the request to Crypto Tool to generate key pair.

3093     3: The Crypto Tool generates key pair and returns them to the Functional Element according to  
3094     the request.

3095     4: The Functional Element checks and dispatches the message to the user and the use case  
3096     ends.

3097     **2.8.7.6.1.2 Alternative Flows**

3098     1: Invalid Input Parameter.

3099         1.1: If in the basic flow 2, Functional Element detects the information provided by the user is  
3100         not valid to generate key pair, Functional Element returns general error message and ends  
3101         the use case.

3102     **2.8.7.6.2 Special Requirements**

3103     None.

3104     **2.8.7.6.3 Pre-Conditions**

3105     None.

3106     **2.8.7.6.4 Post-Conditions**

3107     If the use case successes, a key pair is generated and stored in the key store specified by the  
3108     user.

3109

3110



## 2.9 Log Utility Functional Element

### 2.9.1 Motivation

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

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

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

#### Primary Requirements

- MANAGEMENT-007, [*\*To be fulfilled in next working draft*]
- MANAGEMENT-110,
- MANAGEMENT-112 to MANAGEMENT-114, and
- PROCESS-009.

#### Secondary Requirements

- MANAGEMENT-006,
- MANAGEMENT-095,
- MANAGEMENT-111,
- PROCESS-008,
- PROCESS-115, and
- PROCESS-118.

### 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.

### 2.9.3 Key Features

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

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

- 3143 1.1. The capability to define the format of the log information,  
3144 1.2. The capability to choose the data source to logged to, and  
3145 1.3. The capability to define the name of the log category.
- 3146 2. The Functional Element MUST provide the capability to manage logging of events/records.  
3147 This includes:
- 3148 2.1. The capability to insert a new record into the log,  
3149 *Examples of a log record could include events, transactions status, usages status or*  
3150 *users' activities.*
- 3151 2.2. The capability to search and return result sets of search on log records, and  
3152 2.3. The capability to archive or delete obsolete log records.
- 3153
- 3154 In addition, the following key features could be provided to enhance the Functional Element  
3155 further:
- 3156 1. The Functional Element MAY also provide the capability to perform conditional search or  
3157 viewing of log records.
- 3158 2. The Functional Element MAY provide the capability to perform basic statistical analysis on  
3159 log records. Basic statistical analysis capabilities include:
- 3160 2.1. Minimum and maximum value calculations on numerical values,  
3161 2.2. Mean values calculations on numerical values, and  
3162 2.3. Standard deviation calculations on numerical values.
- 3163 *Note: Report Structure Creation, Generation and Notification are expected to be added in the*  
3164 *next Working Draft version under this optional key features.*  
3165

#### 3166 **2.9.4 Interdependencies**

3167 None

#### 3168 **2.9.5 Related Technologies and Standards**

3169 None

## 3170 2.9.6 Model

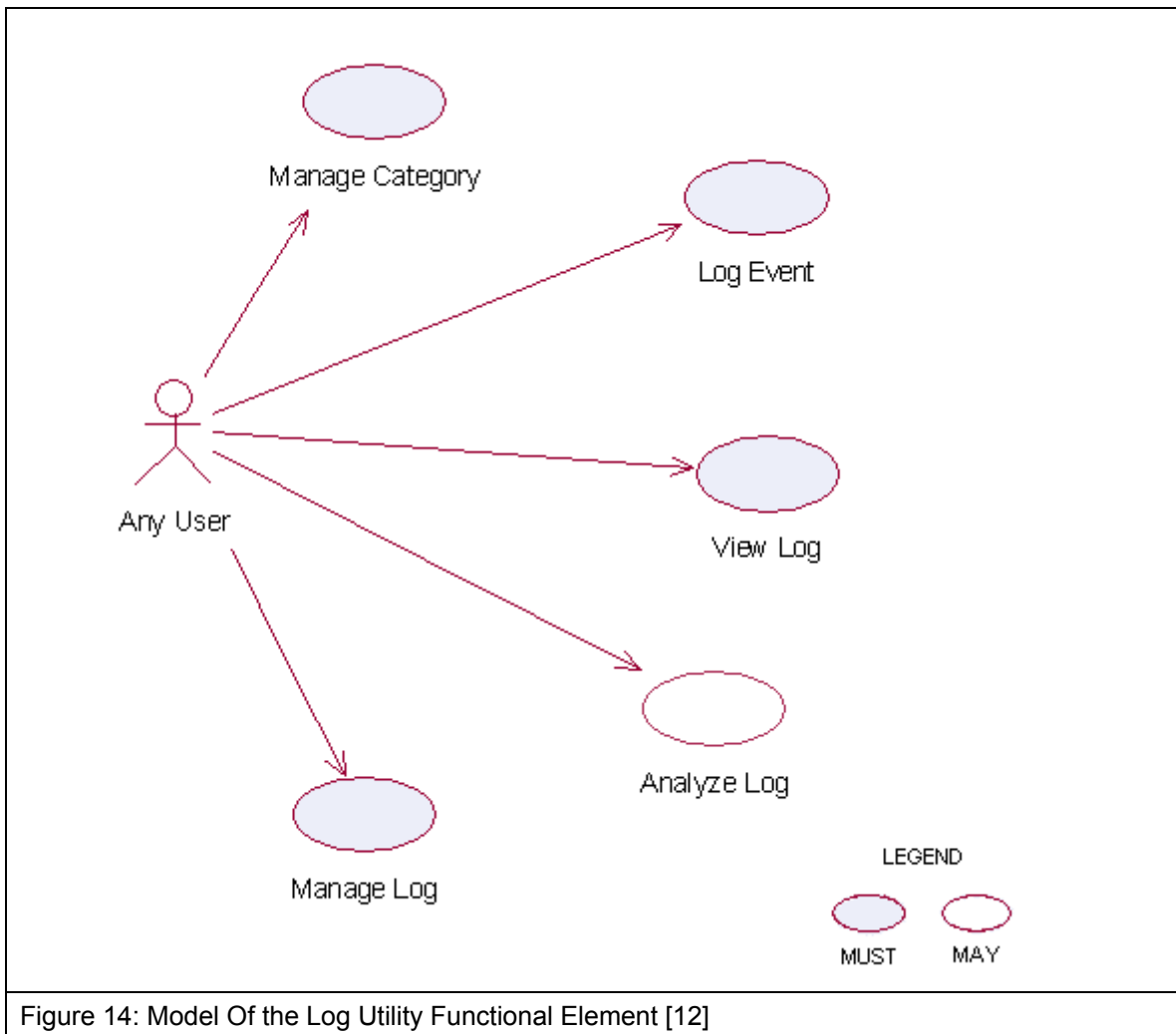


Figure 14: Model Of the Log Utility Functional Element [12]

3171

## 3172 2.9.7 Usage Scenarios

### 3173 2.9.7.1 Manage Category

#### 3174 2.9.7.1.1 Description

3175 This use case allows any user to manage log category. Log category defines the data fields that  
3176 the user wants to log.

#### 3177 2.9.7.1.2 Flow of Events

##### 3178 2.9.7.1.2.1 Basic Flow

3179 This use case starts when users wants to manage the log category.

3180 1: The users send the request to the Functional Element. The request contains the operations  
3181 the users want to perform.

3182 2: The Functional Element receives the request. Based on the operation specified, one of the  
3183 following sub-flows is executed.

3184 If the operation is '**Create Log Category**', then sub-flow 2.1 is executed.

3185 If the operation is '**Retrieve Log Category Information**', then sub-flow 2.2 is executed.

3186 If the operation is '**Delete Log Category**', then sub-flow 2.3 is executed.

3187 2.1: Create Log Category.

3188 2.1.1: The Functional Element gets the following data from the users.

3189 • Category name

3190 • The definition of category

3191 • The data source where the log is located

3192 2.1.2: The Functional Element checks the uniqueness of the category name.

3193 2.1.3: The Functional Element connects to the data source according to the specified  
3194 data source.

3195 2.1.4: The Functional Element creates the empty log in the data source.

3196 2.1.5: The Functional Element writes the category name and its definition in its own  
3197 category definition record and the use case end.

3198 2.2: Retrieve Log Category Information.

3199 2.2.1: The Functional Element gets the category name.

3200 2.2.2: The Functional Element checks the existence of this category.

3201 2.2.3: The Functional Element retrieves the definition of this category.

3202 2.2.4: The Functional Element returns the definition of this category to the user and the  
3203 use case ends.

3204 2.3: Delete Log Category.

3205 2.3.1: The Functional Element gets the category name.

3206 2.3.2: The Functional Element checks the existence of this category.

3207 2.3.3: The Functional Element deletes its own records of category definition and the use  
3208 case ends.

3209 **2.9.7.1.2.2 Alternative Flows**

3210 1: Category Already Exists.

3211 1.1: In sub-flow 2.1.2, if the category name is already used by others, the Functional Element  
3212 returns an error message and the use case ends.

3213 2: Data Source Not Available.

3214 2.1: In sub-flow 2.1.3, if the data source is not available, the Functional Element returns an  
3215 error message and the use case ends.

- 3216 3: Create Log Error.
- 3217 3.1: In sub-flow 2.1.4, if the log cannot be created on the specified data source, the  
3218 Functional Element returns an error message and the use case ends.
- 3219 4: Category Does Not Exist.
- 3220 4.1: In sub-flow 2.2.1 and 2.3.1, the category cannot be found in Functional Element category  
3221 definition, the Functional Element returns an error message and the use case ends.
- 3222 5: Delete Category Error.
- 3223 5.1: In sub-flow 2.3.3, the log category cannot be deleted, the Functional Element returns an  
3224 error message and the use case ends.
- 3225 **2.9.7.1.3 Special Requirements**
- 3226 None
- 3227 **2.9.7.1.4 Pre-Conditions**
- 3228 None.
- 3229 **2.9.7.1.5 Post-Conditions**
- 3230 If the use case was successful, the category definition is saved to the Functional Element and an  
3231 empty log is created in the specified data source. Otherwise, the Functional Element's state is  
3232 unchanged.
- 3233 **2.9.7.2 Log Event**
- 3234 **2.9.7.2.1 Description**
- 3235 The use case allows any user to log any event.
- 3236 **2.9.7.2.2 Flow of Events**
- 3237 **2.9.7.2.2.1 Basic Flow**
- 3238 This use case starts when users want to write to a log.
- 3239 1: The users provide the event data, category name he/she wants to log to the Functional  
3240 Element.
- 3241 2: The Functional Element gets the definition of the category.
- 3242 3: The Functional Element connects the log data source.
- 3243 4: The Functional Element writes the log record into the end of the log file and the use case ends.
- 3244 **2.9.7.2.2.2 Alternative Flows**
- 3245 1: Category Does Not Exist.
- 3246 1.1: If in basic flow 2, the category that the users want to write does not exist, the Functional  
3247 Element returns an error message and the use case ends.
- 3248 2: Data Source Not Available.

3249 2.1: If in basic flow 3, the data source is not available, the Functional Element returns an error  
3250 message and the use case ends.

3251 3: Data Not Match.

3252 3.1: If in basic flow 4, the data provided by the users for logging does not match with the  
3253 category definition in the Functional Element, the Functional Element returns an error  
3254 message and the use case ends.

### 3255 **2.9.7.2.3 Special Requirements**

3256 None.

### 3257 **2.9.7.2.4 Pre-Conditions**

3258 None.

### 3259 **2.9.7.2.5 Post-Conditions**

3260 If the use case was successful, the log record is saved to the Functional Element. Otherwise, the  
3261 Functional Element's state is unchanged.

## 3262 **2.9.7.3 View Log**

### 3263 **2.9.7.3.1 Description**

3264 The use case allows users to retrieve the log content.

### 3265 **2.9.7.3.2 Flow of Events**

#### 3266 **2.9.7.3.2.1 Basic Flow**

3267 This use case starts when users want to view a log.

3268 1: The users specify the category name and the search criteria, such as searching by event type  
3269 or searching by time period (starting time and end time).

3270 2: The Functional Element connects to the data storage where the log records are stored.

3271 3: The Functional Element retrieves the log content and returns to the service users and the use  
3272 case ends.

#### 3273 **2.9.7.3.2.2 Alternative Flows**

3274 1: Search Criteria Not Valid.

3275 1.1: If in basic flow 1 and 3, the search criteria specified by the users is invalid for Search  
3276 Service, the Functional Element returns an error message and the use case ends.

### 3277 **2.9.7.3.3 Special Requirements**

3278 None.

### 3279 **2.9.7.3.4 Pre-Conditions**

3280 None.

3281 **2.9.7.3.5 Post-Conditions**

3282 None.

3283 **2.9.7.4 Analyze Log Data**

3284 **2.9.7.4.1 Description**

3285 The use case allows users to analyze the log data, i.e., to get statistics of certain event. The  
3286 service users may get statistical results on the log data, such as the cumulative events and mean  
3287 of two numerical values.

3288 **2.9.7.4.2 Flow of Events**

3289 **2.9.7.4.2.1 Basic Flow**

3290 This use case starts when users want to analyze the log data.

3291 1: The users specify the items to analyze, i.e. field name and category name.

3292 2: The users specify the analysis method, option among max, min and mean.

3293 3: The Functional Element retrieves the definition of the category and validates the parameters  
3294 provided by the users.

3295 4: The Functional Element connects to the data source and retrieves the log data.

3296 5: The Functional Element analyses the log data and does statistics on the data with respect to  
3297 what is specified in Step 1 and 2.

3298 6: The Functional Element returns the analyzed result and the use case ends.

3299 **2.9.7.4.2.2 Alternative Flows**

3300 1: Invalid Item Specified.

3301 1.1: If in basic flow 1, the analyze items specified by the users are invalid, i.e. invalid field and  
3302 invalid data source, the Functional Element returns an error message and the use case ends.

3303 2: Category Does Not Exist.

3304 2.1: If in basic flow 3, the category that the users want to write to does not exist, the  
3305 Functional Element returns an error message and the use case ends.

3306 3: Data Source Not Available.

3307 3.1: If in basic flow 4, the data source is not available, the Functional Element returns an error  
3308 message and the use case ends.

3309 **2.9.7.4.3 Special Requirements**

3310 **2.9.7.4.3.1 Supportability**

3311 Only basic statistic methods of numerical value are supported.

3312 **2.9.7.4.4 Pre-Conditions**

3313 None.

3314 **2.9.7.4.5 Post-Conditions**

3315 None.

3316 **2.9.7.5 Manage Log**

3317 **2.9.7.5.1 Description**

3318 The use case allows users to drop log and backup log.

3319 **2.9.7.5.2 Flow of Events**

3320 **2.9.7.5.2.1 Basic Flow**

3321 The use case starts when the users want to drop and backup a log of a specific data source.

3322 1: The users specify the function name to the Functional Element.

3323 2: Based on the operation specified, one of the following sub-flows is executed.

3324 If the operation is '**Delete Log**', then sub-flow 2.1 is executed.

3325 If the operation is '**Backup Log**', then sub-flow 2.2 is executed.

3326 2.1: Delete Log

3327 2.1.1: The Functional Element gets category name from the users.

3328 2.1.2: The Functional Element retrieves the definition of the category.

3329 2.1.3: The Functional Element connects to the corresponding data source.

3330 2.1.4: The Functional Element deletes the log from the data source.

3331 2.2: Backup Log

3332 2.2.1: The Functional Element gets the category name and the destination file name from  
3333 the users.

3334 2.2.2: The Functional Element retrieves the definition of the category.

3335 2.2.3: The Functional Element connects to the corresponding data source.

3336 2.2.4: The Functional Element read the original log and writes it to the destination file.

3337 **2.9.7.5.2.2 Alternative Flows**

3338 1: Category Does Not Exist.

3339 1.1: If in basic flow 2.1.2 and 2.2.2 the category that the users want to write does not exist,  
3340 the Functional Element returns an error message and the use case ends.

3341 2: Data Source Not Available.

3342 2.1: If in basic flow 2.1.4 and 2.2.4, the data source is not available, the Functional Element  
3343 returns an error message and the use case ends.



3344    **2.9.7.5.3 Special Requirements**

3345    None.

3346    **2.9.7.5.4 Pre-Conditions**

3347    None.

3348    **2.9.7.5.5 Post-Conditions**

3349    None.

## 2.10 Notification Functional Element

### 2.10.1 Motivation

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

Providing the capability to configure and link with the various gateways so as to enable messages dissemination, and

Providing the capability to send instantaneous or scheduled messages to the intended audiences.

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

Primary Requirements

- DELIVERY-003, and
- PROCESS-118.

Secondary Requirements

- MANAGEMENT-205,
- PROCESS-005,
- PROCESS-102,
- PROCESS-107, and
- PROCESS-110.

### 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.
-------------	---

3373

### 3374 2.10.3 Key Features

3375 Implementations of the Notification Functional Element are expected to provide the following key  
3376 features:

- 3377 1. The Functional Element MUST support notifications using both the SMS and SMTP  
3378 protocols.
- 3379 7. The Functional Element MUST provide the capability to configure supported SMS gateway(s)  
3380 and the SMTP servers where applicable.  
3381 *Example: The capability to configure the username and password for SMTP server's*  
3382 *authentication.*
- 3383 8. The Functional Element MUST provide the capability to send notifications to single and  
3384 multiple recipients.
- 3385 9. The Functional Element MUST provide the capability to structure a notification based on the  
3386 selected protocol(s).

3387

3388 In addition, the following key features could be provided to enhance the Functional Element  
3389 further:

- 3390 1. The Functional Element MAY provide the capability to send notifications either instantly or  
3391 based on a pre-defined schedule.
- 3392 2. If Key Feature (1) is provided, the Functional Element MAY also provide the capability to  
3393 send scheduled messages in the following manner:  
3394 2.1. Hourly,  
3395 2.2. Daily,  
3396 2.3. Weekly, and  
3397 2.4. Monthly (based on a particular date or particular day of the week).

3398 *Note: The next working draft version will attempt to look at other available protocols.*

3399

### 3400 2.10.4 Interdependencies

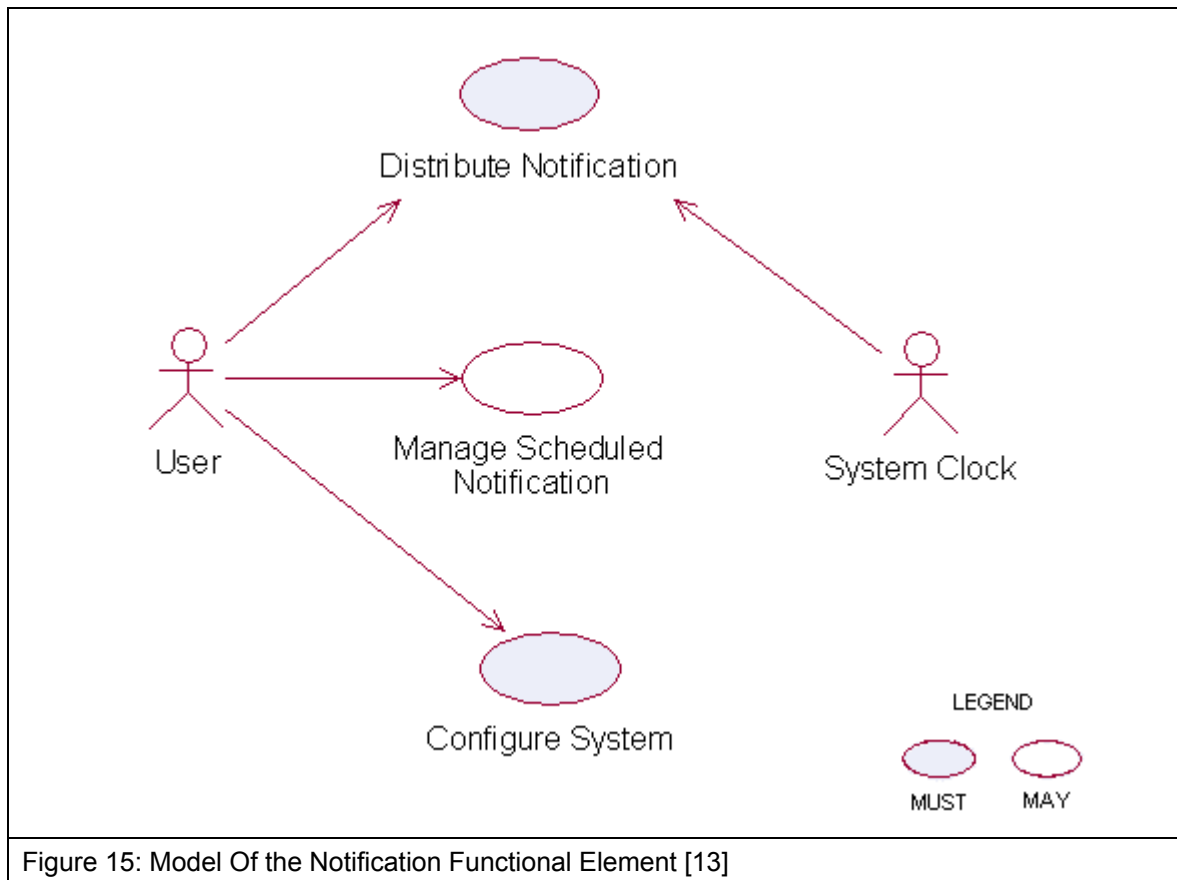
3401 None

### 3402 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.

3403

## 3404 2.10.6 Model



## 3405 2.10.7 Usage Scenarios

### 3406 2.10.7.1 Distribute Notification

#### 3407 2.10.7.1.1 Description

3408 This use case allows the Functional Element to distribute messages to intended recipients.

#### 3409 2.10.7.1.2 Flow of Events

##### 3410 2.10.7.1.2.1 Basic Flow

3411 This use case starts when the service user or system clock wishes to send message to recipient.

3412 1: The Functional Element decides to send messages to recipients. Based on the operation  
3413 specified, one of the following sub-flows is executed.

3414 If the request is '**Initiated By The User**', then sub-flow 1.1 is executed.

3415 If the request is '**Initiated By The System Clock**' then sub-flow 1.2 is executed.

3416 1.1: Initiated By The User

3417 1.1.1: The Functional Element receives the request from the service user.

3418 1.1.2: The Functional Element validates passed parameters such as message type,  
3419 recipient address, and message key and message length.

3420 1.1.3: The Functional Element checks the availability of the connection.

3421 1.1.4: The Functional Element sends message to recipient(s) and the use case end

3422 1.2 : Initiated By The System Clock

3423 1.2.1: The Functional Element checks scheduled message(s) and end date for scheduled  
3424 message.

3425 1.2.2: Once the Functional Element detects scheduled messages, one of the sub-flows is  
3426 executed.

- 3427 • If the Functional Element detects the scheduled notification is once, the '**Activate**  
3428 **Once Notification**' sub-flow 1.2.2.1 is executed.
- 3429 • If the Functional Element detects the scheduled notification is daily, the '**Activate**  
3430 **Daily Notification**' sub-flow 1.2.2.2 is executed.
- 3431 • If the Functional Element detects the scheduled notification is weekly, the  
3432 '**Activate Weekly Notification**' sub-flow 1.2.2.3 is executed.
- 3433 • If the Functional Element detects the scheduled notification is Monthly, the  
3434 '**Activate Monthly Notification**' sub-flow 1.2.2.4 is executed.

3435 1.2.2.1: Activate Once Notification.

3436 1.2.2.1.1: The Functional Element compares the system time with the scheduled  
3437 message's time and gets notification details if both times are match.

3438 1.2.2.2: Activate Daily Notification.

3439 1.2.2.2.1: The Functional Element compares the system time with the scheduled  
3440 message's time and gets notification details if both times are match.

3441 1.2.2.3: Activate Weekly Notification.

3442 1.2.2.3.1: The Functional Element compares the system date and time with the  
3443 scheduled message's date and time and gets notification details if both date &  
3444 time are match.

3445 1.2.2.4: Activate Monthly Notification.

3446 1.2.2.4.1: The Functional Element compares the system date and time with the  
3447 scheduled message's date and time and gets notification ID if both date & time  
3448 are match.

3449 1.2.3: The Functional Element extracts the list of recipient(s) and message(s).

3450 1.2.4: The Functional Element checks the availability of connection.

3451 1.2.5: The Functional Element sends message to recipient(s) and the use case ends.

3452 **2.10.7.1.2.2 Alternative Flows**

3453 1: Unsupported Message Type/Recipient Address/Message.

3454 1.1: If in basic flow 1.1.2, Functional Element detects unsupported message type, recipient  
3455 address or message, the Functional Element returns an error message and the use case  
3456 ends.

3457 2: Connection Fail.

3458 2.1: If in basic flow 1.1.3 and 1.2.4, the Functional Element is unable to detect connection  
3459 type, the Functional Element returns an error message and the use case ends

3460 3: Delete Scheduled Message.

3461 3.1: If in basic flow 1.2.1, if the Functional Element detects that the scheduled message has  
3462 expired, the Functional Element will proceed to delete those messages.

3463 **2.10.7.1.3 Special Requirements**

3464 **2.10.7.1.3.1 Supportability**

3465 None

3466 **2.10.7.1.4 Pre-Conditions**

3467 None.

3468 **2.10.7.1.5 Post-Conditions**

3469 None.

3470 **2.10.7.2 Manage Scheduled Notification**

3471 **2.10.7.2.1 Description**

3472 This use case allows the service user to maintain the notification information. This includes  
3473 adding, changing and deleting notification information from the Functional Element.

3474 **2.10.7.2.2 Flow of Events**

3475 **2.10.7.2.2.1 Basic Flow**

3476 This use case starts when the service user wishes to schedule notification message(s).

3477 1: The Functional Element requests the service user to specify the function he/she would like to  
3478 perform (such as create, update and delete notification message).

3479 2: Once the Functional Element user provides the requested information, one of the sub-flows is  
3480 executed.

3481 If the service user provides '**Create Notification**', then sub-flow 2.1 is executed.

3482 If the service user provides '**Delete Notification**', then sub-flow 2.2 is executed.

3483 2.1 Create Notification

3484 2.1.1: The Functional Element receives the request from the service user.

3485 2.1.2: The Functional Element validates passed parameters such as schedule type,  
3486 message type, recipient address, message key and the message length.

3487 2.1.3: The Functional Element generates and assigns a unique Notification ID and adds  
3488 the notification information to the Functional Element and ends use case.

3489 2.2: Delete Notification

3490 2.2.1: The Functional Element requests the service user to provide the Notification  
3491 information.

3492 2.2.2: The Functional Element retrieves the existing Notification information.

3493 2.2.3: The Functional Element deletes the Notification record and use case ends.

3494 **2.10.7.2.2.2 Alternative Flows**

3495 1: Invalid Parameters.

3496 1.1: If in basic flow 2.1.2, if the Functional Element detects invalid parameters such as  
3497 schedule type, date & time, recipient address, message key and message, the Functional  
3498 Element returns an error message and the use case ends.

3499 **2.10.7.2.3 Special Requirements**

3500 None.

3501 **2.10.7.2.4 Pre-Conditions**

3502 None.

3503 **2.10.7.2.5 Post-Conditions**

3504 If the use case was successful, the schedule message information is added to Functional  
3505 Element. Otherwise, the Functional Element's state is unchanged.

3506 **2.10.7.3 Configure System**

3507 **2.10.7.3.1 Description**

3508 This use case allows the service user to maintain the notification Functional Element behaviors.  
3509 This includes configuration of supported Notification Channel, Default Notification Channel,  
3510 Schedule Types, and SMS and SMTP Gateway.

3511 **2.10.7.3.2 Flow of Events**

3512 **2.10.7.3.2.1 Basic Flow**

3513 1: The Functional Element requests the service user to specify or configure the function he/she  
3514 would like to perform (such as create, update and delete configuration parameters).

3515 2: Once the Functional Element user provides the requested information, one of the sub-flows is  
3516 executed.

3517 If user wishes to configure '**Notification Channel**', then sub-flow 2.1 is executed.

3518 If user wishes to configure '**Default Notification Channel**', then sub-flow 2.2 is executed.

3519 If user wishes to configure '**Schedule Types**', then sub-flow 2.3 is executed.

3520 If user wishes to configure '**SMTP server and SMS Gateway**', then sub-flow 2.4 is executed.

3521	2.1 Notification Channel.
3522	2.1.1: The Functional Element receives the request from the service user.
3523	2.1.2: The Functional Element validates passed parameters such as Notification Channel
3524	information.
3525	2.1.3: The Functional Element generates and assigns a unique Notification Channel ID
3526	and adds the notification information to the Functional Element and the use case ends.
3527	2.2: Default Notification Channel.
3528	2.2.1: The Functional Element requests the service user to provide the Default
3529	Notification information.
3530	2.2.2: The Functional Element validates passed parameters such as Default Notification
3531	Channel information.
3532	2.2.3: The Functional Element updates existing Default Notification or create new Default
3533	Notification information and the use case ends.
3534	2.3 Schedule Types.
3535	2.3.1: The Functional Element receives the request from the service user.
3536	2.3.2: The Functional Element validates passed parameters such as Schedule Type.
3537	2.3.3: The Functional Element generates and assigns a unique Schedule Type ID and
3538	adds the Schedule Type information to the Functional Element and the use case ends.
3539	2.4: SMTP server and SMS Gateway.
3540	2.4.1: The Functional Element requests the service user to provide the SMTP server and
3541	SMS Gateway information.
3542	2.4.2: The Functional Element validates passed parameters such as SMTP server and
3543	SMS Gateway information.
3544	2.4.3: The Functional Element updates existing SMTP server and SMS Gateway or
3545	create new SMTP server and SMS Gateway information and the use case ends.
3546	<b>2.10.7.3.2.2 Alternative Flows</b>
3547	1: Invalid Parameters.
3548	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
3549	parameters such as Notification Channel, Default Notification Channel, and SMTP server,
3550	Schedule Types and SMS Gateway information, the Functional Element returns an error
3551	message and the use case ends
3552	<b>2.10.7.3.3 Special Requirements</b>
3553	None.
3554	<b>2.10.7.3.4 Pre-Conditions</b>
3555	None.



3556    **2.10.7.3.5    Post-Conditions**

3557    None.

## 2.11 Phase and Lifecycle Management Functional Element

### 2.11.1 Motivation

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

Basic lifecycle management facilities,  
Basic phase management facilities, and  
Management of user information in phases across the whole lifecycle.

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

Primary Requirements

- MANAGEMENT-070 to MANAGEMENT-078

Secondary Requirements

- None

### 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"><li>• Project Initiation</li><li>• Design</li><li>• Construction</li><li>• Maintenance.</li></ul>
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>
------------------------------	---

3577

### 3578 2.11.3 Key Features

3579 Implementations of the Phase and Lifecycle Management Functional Element are expected to  
3580 provide the following key features:

- 3581 1. The Functional Element MUST provide basic structures based on a set of pre-defined  
3582 attributes for Lifecycle and Phase.
- 3583 2. The Functional Element MUST provide the capability to manage the creation and deletion of  
3584 instances of Lifecycle and Phase based on the pre-defined structures.
- 3585 3. The Functional Element MUST provide a means to manage the lifecycles and phases  
3586 contained within. This includes:
  - 3587 3.1. The capability to retrieve and update a lifecycle or phase
  - 3588 3.2. The capability to add/remove phases from a lifecycle
- 3589 4. The Functional Element MUST provide a mechanism to manage the collection of users in a  
3590 Phase. This includes:
  - 3591 4.1. The capability to assign and un-assign users belonging to a Phase.
  - 3592 4.2. The users could be individual Users or Groups.
- 3593 10. The Functional Element MUST provide a mechanism for managing Groups across different  
3594 application domains.

3595 *Example: Namespace control mechanism*

3596

### 3597 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..

3598

### 3599 2.11.5 Related Technologies and Standards

3600 None.

### 3601 2.11.6 Model

3602

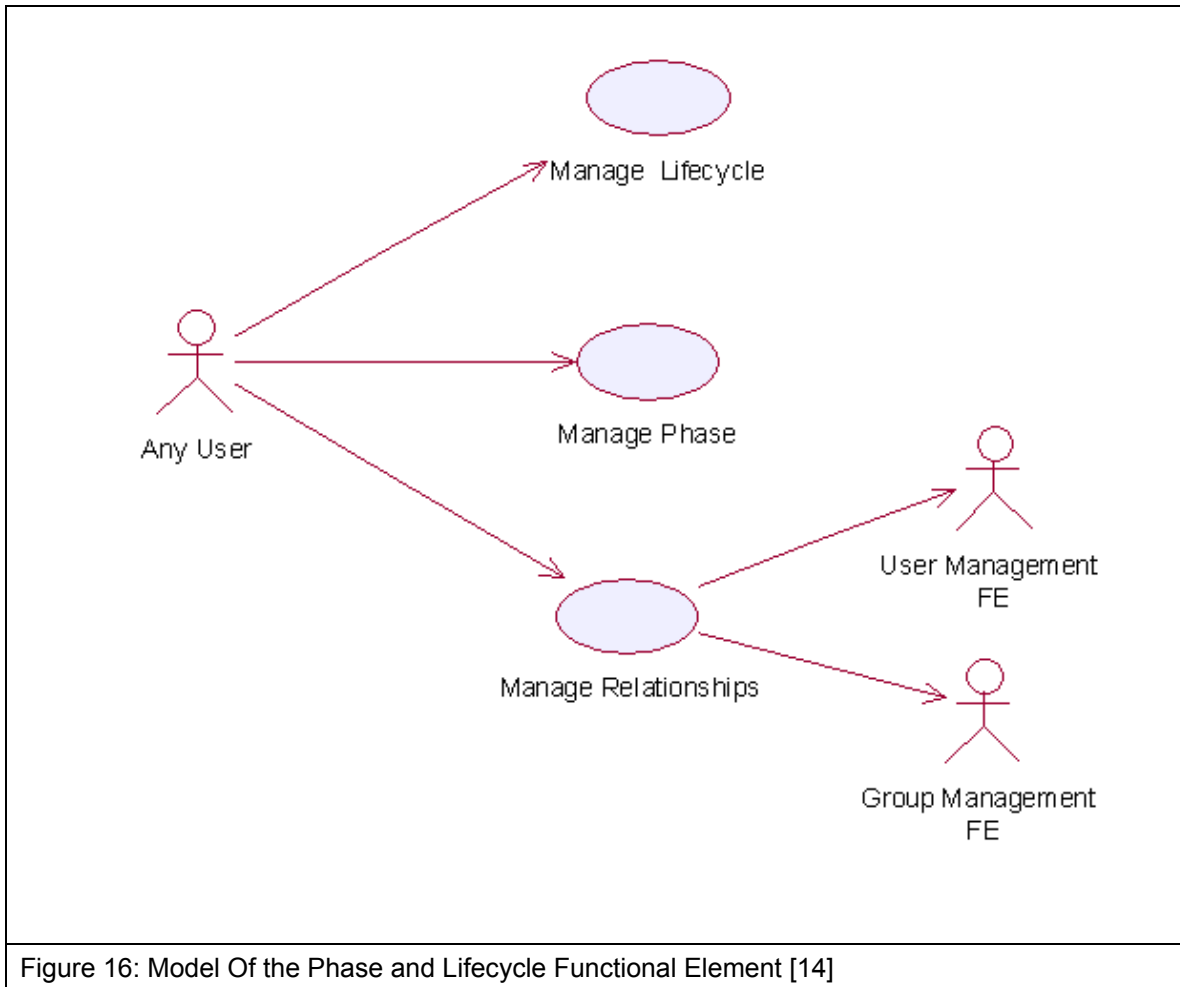


Figure 16: Model Of the Phase and Lifecycle Functional Element [14]

## 2.11.7 Usage Scenarios

### 2.11.7.1 Manage Lifecycle

#### 2.11.7.1.1 Description

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

#### 2.11.7.1.2 Flow of Events

##### 2.11.7.1.2.1 Basic Flow

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

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

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

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

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

1: Create Lifecycle.

3615           1.1: User provides information to create lifecycle.

3616           1.2: Functional Element creates lifecycle and the use case ends.

3617   2: Retrieve Lifecycle

3618           2.1: User provides the lifecycle name, lifecycle namespace.

3619           2.2: Functional Element returns the lifecycle information and the use case ends.

3620   3: Update Lifecycle.

3621           3.1: User provides the lifecycle information.

3622           3.2: Functional Element updates the lifecycle-phase and the use case ends.

3623   4: Delete Lifecycle.

3624           4.1: User provides lifecycle name and lifecycle namespace.

3625           4.2: Functional Element deletes the lifecycle and the use case ends.

3626   **2.11.7.1.2.2   Alternative Flows**

3627   1: Lifecycle Does Not Exist.

3628           1.1: In basic flow 2.1, 3.1 and 4.1, if lifecycle can not be found based on lifecycle name and

3629           lifecycle namespace provided by user, Functional Element returns an error message and the

3630           use case ends.

3631   2: Creation Of Lifecycle Fails.

3632           2.1: In basic flow 1.2, if lifecycle cannot be created, the Functional Element returns an error

3633           message and the use case ends

3634   **2.11.7.1.3   Special Requirements**

3635   None.

3636   **2.11.7.1.4   Pre-Conditions**

3637   None.

3638   **2.11.7.1.5   Post-Conditions**

3639   None.

3640   **2.11.7.2   Manage Phase**

3641   **2.11.7.2.1   Description**

3642   This use case describes the management of different phases in a project.

3643   **2.11.7.2.2   Flow of Events**

3644   **2.11.7.2.2.1   Basic Flow**

3645   This use case starts when the user wants to manage phase.

3646 If user wants to '**Create Phase**', then basic flow 1 is executed.

3647 If user wants to '**Retrieve Phase**', then basic flow 2 is executed.

3648 If user wants to '**Update Phase**', then basic flow 3 is executed.

3649 If user wants to '**Delete Phase**', then basic flow 4 is executed.

3650 1: Create Phase.

3651 1.1: User provides information to create phase.

3652 1.2: Functional Element creates phase and the use case ends.

3653 2: Retrieve Phase.

3654 2.1: User provides phase name, lifecycle name and lifecycle namespace.

3655 2.2: Functional Element returns the phase information and the use case ends.

3656 3: Update Phase.

3657 3.1: User provides the phase information.

3658 3.2: Functional Element updates the phase and the use case ends.

3659 4: Delete Phase.

3660 4.1: User provides phase name, lifecycle name and lifecycle namespace

3661 4.2: Functional Element deletes phase and the use case ends.

3662 **2.11.7.2.2.2 Alternative Flows**

3663 1: Phase Does Not Exist.

3664 1.1: In basic flow 2.1, 3.1 and 4.1 if phase cannot be found based on phase name, lifecycle

3665 name and lifecycle namespace provided by user, Functional Element returns an error

3666 message and the use case ends.

3667 2: Creation of phase fails.

3668 2.1: In basic flow 1.2, if phase cannot be created, the Functional Element returns an error

3669 message and the use case ends

3670 **2.11.7.2.3 Special Requirements**

3671 None.

3672 **2.11.7.2.4 Pre-Conditions**

3673 None.

3674 **2.11.7.2.5 Post-Conditions**

3675 None.

3676     **2.11.7.3     Manage Relationship**

3677     **2.11.7.3.1    Description**

3678     This use case describes the management of the relationship between user/group and phase in a  
3679     lifecycle.

3680     **2.11.7.3.2    Flow of Events**

3681     **2.11.7.3.2.1   Basic Flow**

3682     This use case starts when the user wants to manage the relationship between the user/group and  
3683     phase.

3684     If user refers to '**Create Relationship**', basic flow 1 is executed.

3685     If user refers to '**Update Relationship**', basic flow 2 is executed.

3686     If user wants to '**Retrieve Relationship**', basic flow 3 is executed.

3687     If user refers to '**Delete Relationship**', basic flow 4 is executed.

3688     1: Create Relationship.

3689         1.1: User provides user/group, phase and phase information.

3690         1.2: Functional Element creates relationship and the use case ends.

3691     2: Update Relationship.

3692         2.1: User provides user/group name and user/group namespace.

3693         2.2: Functional Element updates the relationship and the use case ends.

3694     3: Retrieve Relationship.

3695         3.1: User provides user/group name and user/group namespace.

3696         3.2: Functional Element returns the relationship and the use case ends.

3697     4: Delete Relationship.

3698         4.1: User provides user/group name and user/group namespace.

3699         4.2: Functional Element deletes relationship between phases and users/groups and the use  
3700         case ends.

3701     **2.11.7.3.2.2   Alternative Flows**

3702     1: Phase Does Not Exist.

3703         1.1: In basic flow 1,2, 2.2, 3.2 and 4.2, if the phase does not exist, the Functional Element  
3704         returns an error message and the use case ends.

3705     2: User/Group Does Not Exist.

3706         1.1: In basic flow 1,2, 2.2, 3.2 and 4.2, if the user/group does not exist, the Functional  
3707         Element returns an error message and the use case ends.

3708    **2.11.7.3.3    Special Requirements**

3709    None.

3710    **2.11.7.3.4    Pre-Conditions**

3711    None.

3712    **2.11.7.3.5    Post-Conditions**

3713    None.



## 2.12 Policy Management Functional Element (new)

### 2.12.1 Motivation

The Policy Management Functional Element helps enterprise to meet new challenges for IT security as the enterprise applications are now accessible from both the external partners and the customer applications. This Functional Element also helps to build consolidated view of the security configuration across all applications to ensure consistent application of a security policy across all Web Services. It also provides the mechanism for security policy management, establishment, selection and viewing for enterprises to dynamically configure the relevant policy required to protect their interests.

This Functional Element fulfills the following requirements from the Functional Elements Requirements:

Primary Requirements

- SECURITY-110 to SECURITY-119.

Secondary Requirements

- None

### 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

```

<?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

Figure 17 shows an example of a security policy used in Policy Management Functional Element. The security policy is in XACML format.

### 2.12.3 Key Features

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

11. The Functional Element MUST provide the capability to define and manage Policy Categories.
12. The Functional Element MUST provide the capability to define and manage Policies.
13. The Functional Element MUST provide version control capability to defined Policies.
14. The Functional Element MUST provide the ability to manage Policies within a Policy Category; including insertion, update, retrieval and removal of attached Policies.
15. The Functional Element MUST provide the ability to retrieve Policies that are attached to a Policy Category.

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

1. The Functional Element MAY provide the ability to translate Policy into multi-lingual.

3752 **2.12.4 Interdependency**

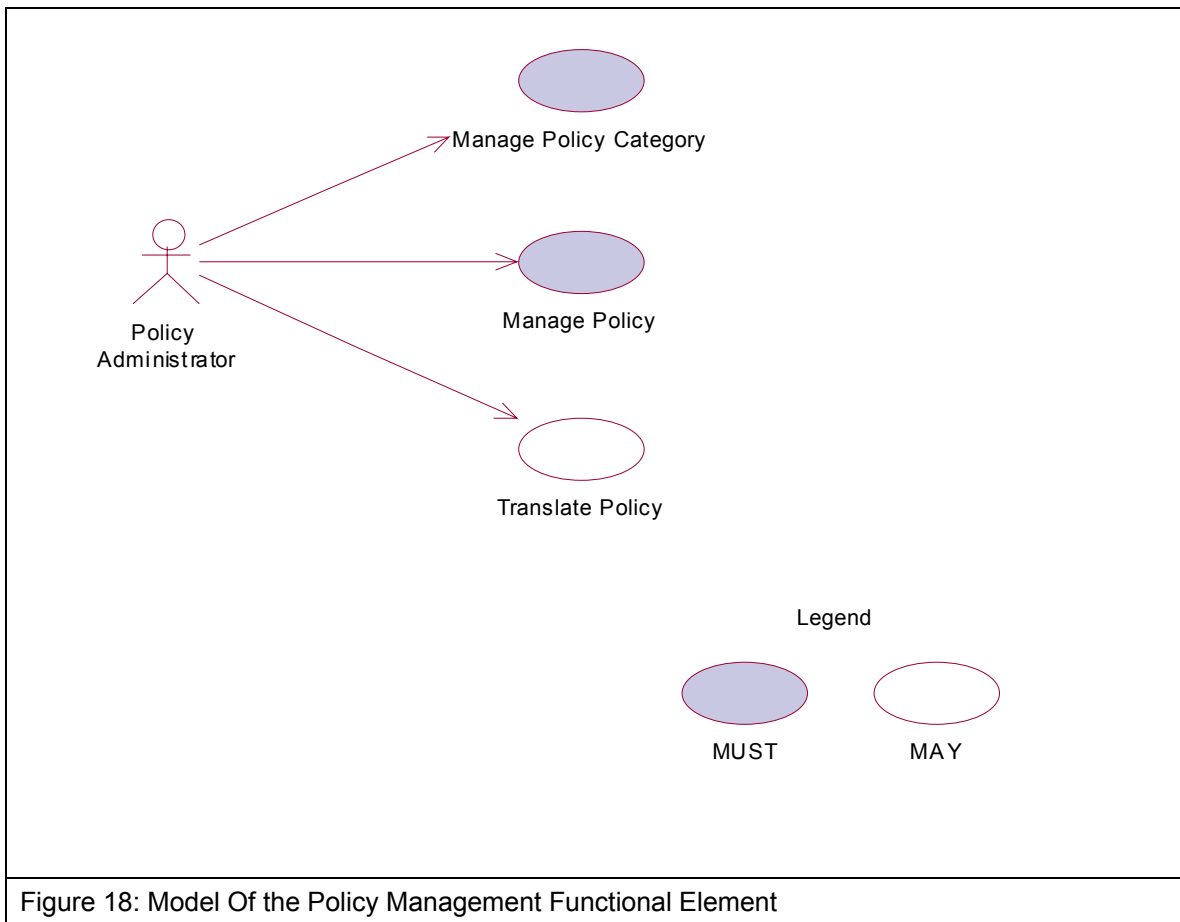
Direct Dependency	
Policy Enforcement Functional Element	The Policy Enforcement Functional Element provides the mechanism to enforce the policy associated to a service. The enforcement is based on a pre-identified access structure. The access structure could be provided by the Role & Access Management 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.

3753

3754 **2.12.5 Related Technologies and Standards**

3755 XACML.

3756 **2.12.6 Model**



3757

3758 **2.12.7 Usage Scenarios**

3759 **2.12.7.1 Manage Policy Category**

3760 **2.12.7.1.1 Description**

3761 This use case allows the policy administrator to manage policy category.

3762 **2.12.7.1.2 Flow of Events**

3763 **2.12.7.1.2.1 Basic Flow**

3764 The use case begins when the policy administrator wants to create/retrieve/update/delete a policy  
3765 category.

3766 1: The policy administrator sends a request to manipulate a policy category.

3767 2: Based on the operation it specifies, one of the following sub-flows is executed:

3768 If the operation is '**Create Policy Category**', the sub-flow 2.1 is executed.

3769 If the operation is '**Retrieve Policy Category**', the sub-flow 2.2 is executed.

3770 If the operation is '**Update Policy Category**', the sub-flow 2.3 is executed.

3771 If the operation is '**Delete Policy Category**', the sub-flow 2.4 is executed.

3772 2.1: Create Policy Category.

3773 2.1.1: The Functional Element gets the category name and definition.

3774 2.1.2: The Functional Element checks whether the category exists.

3775 2.1.3: The Functional Element creates the category.

3776 2.2: Retrieve Policy Category.

3777 2.2.1: The Functional Element gets the category name.

3778 2.2.2: The Functional Element checks whether the category exists.

3779 2.2.3: The Functional Element retrieves the category's information.

3780 2.3: Update Policy Category.

3781 2.3.1: The Functional Element gets the category name and definition.

3782 2.3.2: The Functional Element checks whether the category exists.

3783 2.3.3: The Functional Element updates the category's information.

3784 2.4: Delete Policy Category.

3785 2.4.1: The Functional Element gets the category name.

3786 2.4.2: The Functional Element checks whether the category exists.

3787 2.4.3: The Functional Element removes the category.

3788 3: The Functional Element returns the results of the operation to the policy administrator and the

3789 use case ends.

3790 **2.12.7.1.2.2 Alternative Flows**

3791 1: Category Already Exists.

3792 1.1: If in the basic flow 2.1.2, the category is already defined, Functional Element returns an

3793 error message and the use case ends.

3794 2: Category Not Found.

3795 2.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the category does not exist, Functional Element

3796 returns an error message and the use case ends.

3797 **2.12.7.1.3 Special Requirements**

3798 None.

3799 **2.12.7.1.4 Pre-Conditions**

3800 None.

3801     **2.12.7.1.5     Post-Conditions**

3802     None.

3803

3804     **2.12.7.2        Manage Policy**

3805     **2.12.7.2.1     Description**

3806     This use case allows the policy administrator to manage policy.

3807     **2.12.7.2.2     Flow of Events**

3808     **2.12.7.2.2.1   Basic Flow**

3809     The use case begins when the policy administrator wants to create/retrieve/update/delete a  
3810     policy.

3811     1: The policy administrator sends a request to manipulate a policy.

3812     2: Based on the operation it specifies, one of the following sub-flows is executed:

3813     If the operation is '**Create Policy**', the sub-flow 2.1 is executed.

3814     If the operation is '**Retrieve Policy**', the sub-flow 2.2 is executed.

3815     If the operation is '**Update Policy**', the sub-flow 2.3 is executed.

3816     If the operation is '**Delete Policy**', the sub-flow 2.4 is executed.

3817         2.1: Create Policy.

3818             2.1.1: The Functional Element gets the policy name, content and the Policy Category  
3819             where the policy is to be created.

3820             2.1.2: The Functional Element checks whether the policy exists.

3821             2.1.3: The Functional Element creates the policy.

3822         2.2: Retrieve Policy.

3823             2.2.1: The Functional Element gets the policy name and the Policy Category.

3824             2.2.2: The Functional Element checks whether the policy exists.

3825             2.2.3: The Functional Element retrieves the policy's information.

3826         2.3: Update Policy.

3827             2.3.1: The Functional Element gets the policy name, information and the Policy Category.

3828             2.3.2: The Functional Element checks whether the policy exists.

3829             2.3.3: The Functional Element updates the policy.

3830         2.4: Delete Policy.

3831             2.4.1: The Functional Element gets the policy name and the Policy Category.

3832             2.4.2: The Functional Element checks whether the policy exists.

3833            2.4.3: The Functional Element removes the policy from the Policy Category.

3834    3: The Functional Element returns the results of the operation to the policy administrator and the  
3835    use case ends.

3836    **2.12.7.2.2.2    Alternative Flows**

3837    1: Policy Already Exists.

3838            1.1: If in the basic flow 2.1.2, the policy is already created, Functional Element returns an  
3839            error message and the use case ends.

3840    2: Policy Not Found.

3841            2.1: If in the basic flow 2.2.2, 2.3.2 and 2.4.2, the policy does not exist, Functional Element  
3842            returns an error message and the use case ends.

3843    **2.12.7.2.3    Special Requirements**

3844    None.

3845    **2.12.7.2.4    Pre-Conditions**

3846    None.

3847    **2.12.7.2.5    Post-Conditions**

3848    None.

3849

3850    **2.12.7.3    Translate Policy**

3851    **2.12.7.3.1    Description**

3852    This use case allows the policy administrator to translate policy into desired languages.

3853    **2.12.7.3.2    Flow of Events**

3854    **2.12.7.3.2.1    Basic Flow**

3855    The use case begins when the policy administrator wants to translate a policy.

3856    1: The policy administrator sends a request to translate a policy.

3857    2: The Functional Element gets the policy name and the language desired.

3858    3: The Functional Element checks whether the policy exists.

3859    4: The Functional Element retrieves the policy for translation.

3860    5: The Functional Element returns the results of the operation to the policy administrator and the  
3861    use case ends.

3862    **2.12.7.3.2.2    Alternative Flows**

3863    1: Policy Not Found.

3864 1.1: If in the basic flow 3, the policy does not exist, Functional Element returns an error  
3865 message and the use case ends.

3866 **2.12.7.3.3 Special Requirements**

3867 None.

3868 **2.12.7.3.4 Pre-Conditions**

3869 None.

3870 **2.12.7.3.5 Post-Conditions**

3871 None.



## 2.13 Policy Enforcement Functional Element (new)

### 2.13.1 Motivation

The Policy Enforcement Functional Element helps enterprise to enforce policy for both the external partners and the customer applications that are authorized to access the enterprise applications. This Functional Element helps to ensure that the enterprise's interests and its confidential information are protected.

This Functional Element fulfills the following requirements from the Functional Elements Requirements:

Primary Requirements

- SECURITY-140 to SECURITY-144

Secondary Requirements

- None

### 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

### 2.13.3 Key Features

Implementations of the Policy Enforcement Functional Element are expected to provide the following key features:

1. The Functional Element MUST provide the ability to identify Policy Categories and/or Policies that are to be enforced.
2. The Functional Element MUST provide the ability to access enforced Policies for accepting/rejecting the policy.
3. The Functional Element MUST provide the ability to associate a policy to a service.
4. The Functional Element MUST provide the capability to associate a policy to its service's access privileges through a pre-identified Access structure.
5. The Functional Element MUST provide a mechanism to enforce policy upon acceptance of the policy.
6. The Functional Element MUST provide the ability to enforce policies either based on individual or groups of services.
7. The Functional Element MUST provide the capability to reject access.

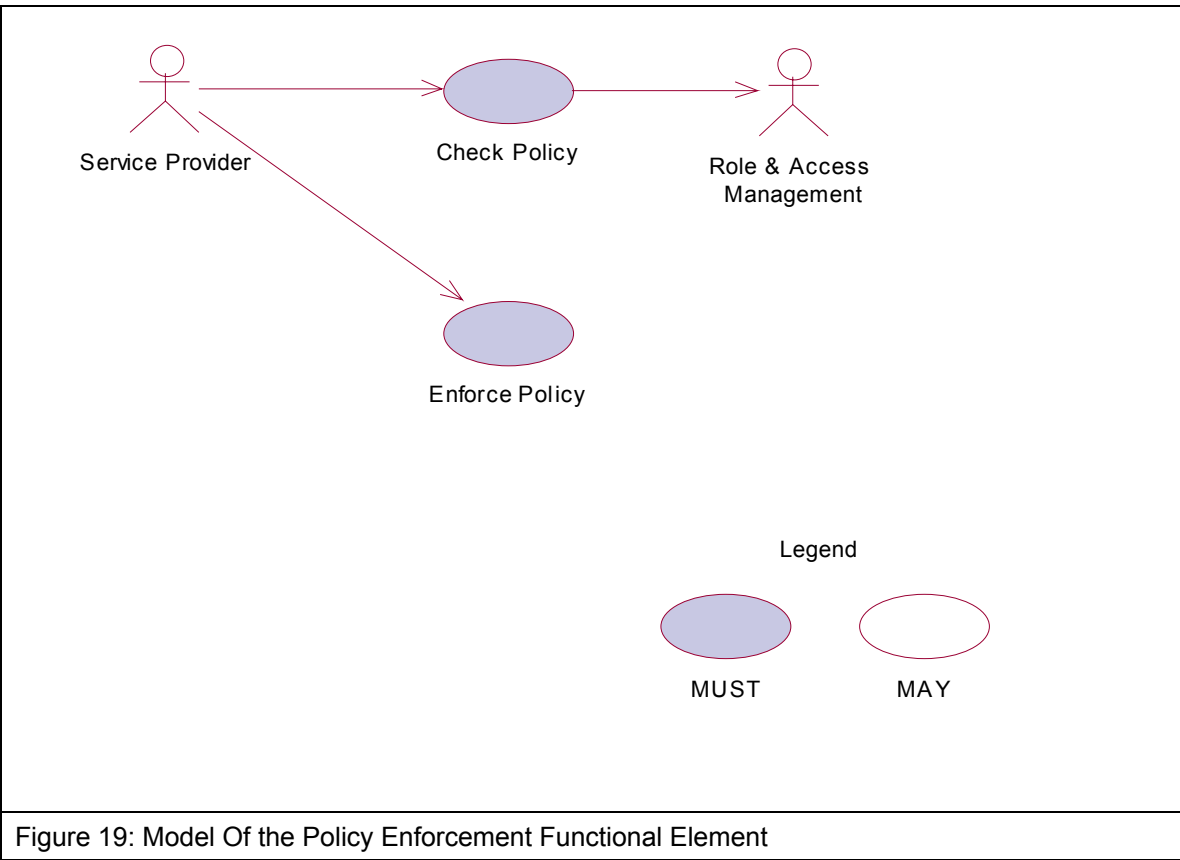
3902     **2.13.4     Interdependency**

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.

3903  
3904     **2.13.5     Related Technologies and Standards**

3905     XACML.

3906  
3907     **2.13.6     Model**



3909     **2.13.7       Usage Scenarios**

3910     **2.13.7.1      Check Policy**

3911     **2.13.7.1.1    Description**

3912     This use case allows the service provider to check policy.

3913     **2.13.7.1.2    Flow of Events**

3914     **2.13.7.1.2.1   Basic Flow**

3915     The use case begins when the service provider wants to check a policy.

3916     1: The service provider sends a request to check a policy.

3917     2: The Functional Element gets the policy and the requested service names.

3918     3: The Functional Element checks whether the policy and the requested service exist.

3919     4: The Functional Element evaluates the policy.

3920     5: The Functional Element resolves any policy conflict.

3921     6: The Functional Element returns the outcome to the service provider and the use case ends.

3922     **2.13.7.1.2.2   Alternative Flows**

3923     1: Policy Not Found.

3924         1.1: If in the basic flow 3, the policy does not exist, Functional Element returns an error  
3925             message and the use case ends.

3926     2: Requested Service Not Found.

3927         2.1: If in the basic flow 3, the requested service does not exist, Functional Element returns an  
3928             error message and the use case ends.

3929     3: Cannot Evaluate Policy.

3930         3.1: If in the basic flow 4, the policy cannot be evaluated, Functional Element returns an error  
3931             message and the use case ends.

3932     4: Cannot Resolve Policy Conflict.

3933         4.1: If in the basic flow 5, the policy conflict cannot be resolved, Functional Element returns  
3934             an error message and the use case ends.

3935     **2.13.7.1.3    Special Requirements**

3936     None.

3937     **2.13.7.1.4    Pre-Conditions**

3938     None.

3939     **2.13.7.1.5     Post-Conditions**

3940     None.

3941

3942     **2.13.7.2       Enforce Policy**

3943     **2.13.7.2.1     Description**

3944     This use case allows the service provider to enforce policy based on the pre-identified access  
3945     structure.

3946     **2.13.7.2.2     Flow of Events**

3947     **2.13.7.2.2.1    Basic Flow**

3948     The use case begins when the service provider wants to enforce policy on a specific service.

3949     1: The service provider sends a request to enforce a policy.

3950     2: The Functional Element gets the policy name and the service activated.

3951     3: The Functional Element checks whether the policy and the service exist.

3952     4: The Functional Element gets the decision outcome.

3953     5: The Functional Element enforces the policy to the service and the use case ends.

3954     **2.13.7.2.2.2    Alternative Flows**

3955     1: Policy Not Found.

3956         1.1: If in the basic flow 3, the policy does not exist, Functional Element returns an error  
3957         message and the use case ends.

3958     2: Service Not Found.

3959         2.1: If in the basic flow 3, the service does not exist, Functional Element returns an error  
3960         message and the use case ends.

3961     3: Cannot Make Decision.

3962         3.1: If in the basic flow 4, decision cannot be made based on the information provided,  
3963         Functional Element returns an error message and the use case ends.

3964     **2.13.7.2.3     Special Requirements**

3965     None.

3966     **2.13.7.2.4     Pre-Conditions**

3967     None.

3968     **2.13.7.2.5     Post-Conditions**

3969     None.

3970

3971 **2.14 Presentation Transformer Functional Element (Deprecated)**

3972

3973 This Functional Element has been deprecated in this version. Please refer to its replacement,  
3974 2.26 Transformer Functional Element (new) for further details.

## 2.15 QoS Functional Element (new)

### 2.15.1 Motivation

With the widespread of Web Services, Quality of Service (QoS) becomes a significant factor in distinguishing the success of service providers. On the other hand poor QoS translates into frustrated customers, which can lead to lost business opportunities. QoS determines the service usability and utility, both of which influence the popularity of the service.

This Functional Element fulfills the following requirements from the Functional Elements Requirements:

#### Primary Requirements

- MANAGEMENT-320,
- MANAGEMENT-321,
- MANAGEMENT-323,
- MANAGEMENT-324,
- [MANAGEMENT-325 and
- MANAGEMENT-312.

#### Secondary Requirements

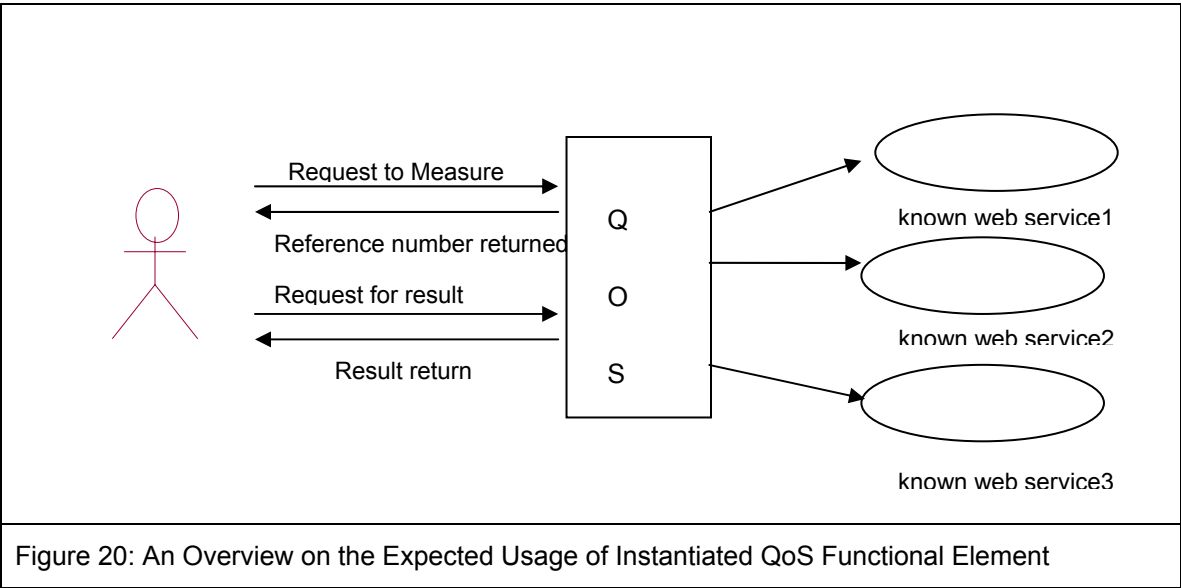
- MANAGEMENT-311 and
- MANAGEMENT-310.

### 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

Figure 20 depicts the basic concepts of 2 steps approach of QoS Functional Element. Step 1 begins when the user (service requester) requests to measure QOS of a known web service. The

Function Element then returns a Reference ID once it receives that request. It also takes necessary measurements and logs them. Step 2 begins when the user requests for the result of measurement. The user provides the Functional Element a Reference ID. With this Reference ID, the Functional Element calculates and returns the result to the user. The measurements used in this Functional Element are designed with the requirements from WSQM Specs (working draft) Version 2.0, dated September 2005 taken into considerations.



### 2.15.3 Key Features

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

1. The Functional Element MUST provide the capability to measure Availability.
2. The Functional Element MUST provide the capability to measure Performance.
3. The Functional Element MUST provide the capability to measure Reliability.
4. The Functional Element MUST provide the capability to measure Accessibility.

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

1. The Functional Element MAY provide confidentiality and non-repudiation by authenticating the parties involved, encrypting messages and providing access control as in the Security aspect.

### 2.15.4 Interdependencies

Direct Dependency	
Log Utility Functional Element	The Log Utility Functional Element is used to record the data.

4024  
4025

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

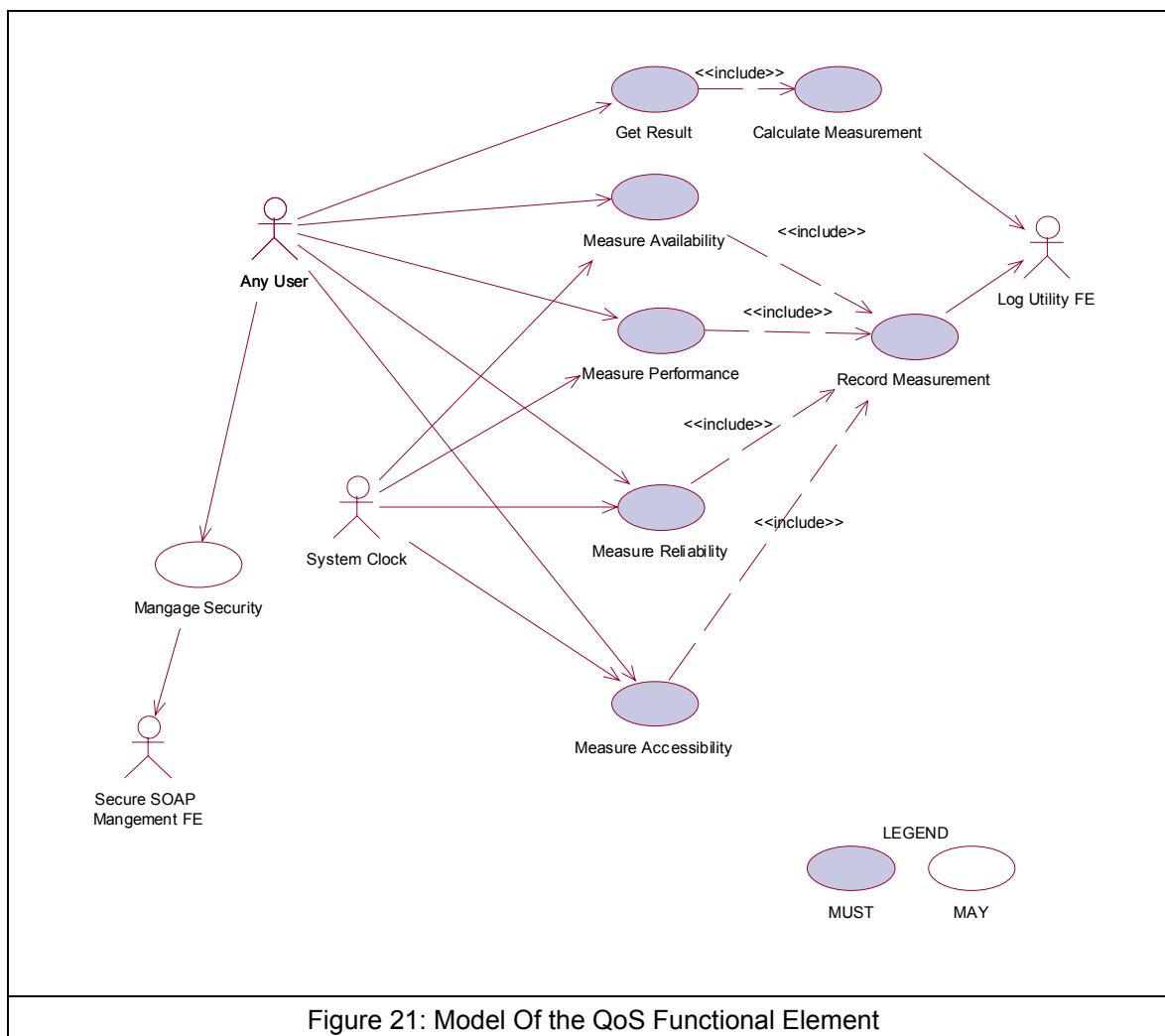
4026

4027 **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.

4028

4029 **2.15.6 Model**





4030

## 4031 **2.15.7 Usage Scenarios**

### 4032 **2.15.7.1 Measure Availability**

#### 4033 **2.15.7.1.1 Description**

4034 This use case allows the user to measure the availability of a known Web Service. User sets a  
4035 period of measurement and the frequency of invocation. The result of this measure is in  
4036 percentage. It is derived by using the successful invocations divided by total number of  
4037 invocations for the given period of measurement.

4038 
$$\text{Total uptime} - \text{downtime} / \text{Total uptime} \times 100 \%$$

4039 
$$= ((\text{number of successful invocation} \times \text{frequency of invocation}) / \text{period of measurement}) \times 100\%$$

4040 
$$= (\text{number of successful invocations} / \text{total invocations}) \times 100\%$$

4041

#### 4042 **2.15.7.1.2 Flow of Events**

##### 4043 **2.15.7.1.2.1 Basic Flow**

4044 This use case starts when the user wants to measure the availability of a Web Service.

- 4045 1. User sets a period of measurement.
- 4046 2. User determines the acceptable invocation interval.
- 4047 3. User submits the WSDL of a known web service.
- 4048 4. Functional Element parses the URL of the WSDL document and extracts the necessary  
4049 information.
- 4050 5. Functional Element generates client base on the extracted information.
- 4051 6. Functional Element invokes the known web service using the generated client
- 4052 7. Functional Element generates a Reference ID.
- 4053 8. Functional Element returns Reference ID to the user.
- 4054 9. Functional Element logs the Reference ID to the Record Measurement Use Case.
- 4055 10. Functional Element logs the Measurement Type to the Record Measurement Use Case.
- 4056 11. Functional Element logs each invocation at every interval to the Record Measurement Use  
4057 Case.
- 4058 12. Functional Element logs successful invocation at every interval to the Record Measurement  
4059 Use Case.
- 4060 13. Functional Element continues to invoke the known web service at every interval until the  
4061 period of measurement is reached and the use case ends.

##### 4062 **2.15.7.1.2.2 Alternative Flows**

- 4063 1. If the structure of the WSDL does not comply with the standard, the Functional Element  
4064 returns an error message and the use case ends.

- 4065 2. If the Functional Element fails to generate the client, the Functional Element returns an error  
4066 message and the use case ends.
- 4067 3. If the Functional Element fails to find the known web service, the Functional Element returns  
4068 an error message and the use case ends.
- 4069 4. If the Functional Element fails to invoke the known web service, the Functional Element  
4070 returns an error message and the use case ends.
- 4071 5. If the Functional Element fails to return a reference ID, the Functional Element returns an  
4072 error message and the use case ends.
- 4073 6. If the Functional Element gets a wrong a reference ID, the Functional Element returns an  
4074 error message and the use case ends.

4075 **2.15.7.1.3 Special Requirements**

4076 None.

4077 **2.15.7.1.4 Pre-Conditions**

4078 None

4079 **2.15.7.1.5 Post-Conditions**

4080 None.

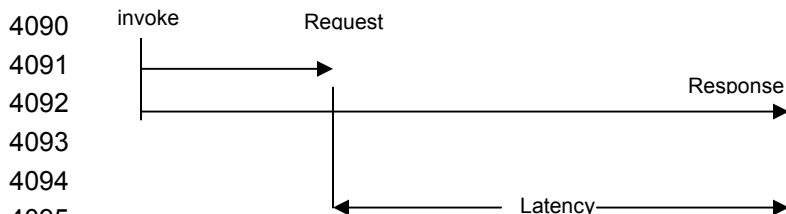
4081

4082 **2.15.7.2 Measure Performance**

4083 **2.15.7.2.1 Description**

4084 This use case allows the user to measure the performance of a Web Service. In Performance  
4085 both Latency and Throughput are measured. For throughput, user sets a period of measurement.  
4086 Throughput is derived as the total number of invocations for the given period of measurement.  
4087 For Latency, user logs the request time and response time of invocation. Latency is derived by  
4088 the response time minus the request time of the invocation, as indicated below:

4089



4097 **2.15.7.2.2 Flow of Events**

4098 **2.15.7.2.2.1 Basic Flow**

4099 This use case starts when a user wants to measure the Performance of a Web Service.

4100 1. Based on the operation it specified, one of the following sub-flows is expected

- 4101
- If the operation is 'Measure Throughput', then sub-flow 1.1 is executed.

- 4102       • If the operation is 'Measure Latency', then sub-flow 1.2 is executed.
- 4103       1.1. Measure Throughput
- 4104           This use case starts when a user wants to measure the Throughput of a Web  
4105           Service.
- 4106           1.1.1. User sets a period of measurement.
- 4107           1.1.2. User submits the WSDL of a known web service.
- 4108           1.1.3. Functional Element parses the URL of the WSDL document and extracts the  
4109           necessary information.
- 4110           1.1.4. Functional Element generates a Reference ID.
- 4111           1.1.5. Functional Element returns a Reference ID to user.
- 4112           1.1.6. Functional Element logs the Reference ID to the Record Measurement Use  
4113           Case.
- 4114           1.1.7. Functional Element logs the measurement type to the Record Measurement  
4115           Use Case.
- 4116           1.1.8. Functional Element waits and logs any invocation to this WSDL until the  
4117           period of measurement is reached and the use case ends.
- 4118       1.2. Measure Latency
- 4119           1.2.1. User submits the WSDL of a known web service.
- 4120           1.2.2. Functional Element parses URL of the WSDL document and extracts the  
4121           necessary information.
- 4122           1.2.3. Functional Element invokes the known web service.
- 4123           1.2.4. Functional Element generates a Reference ID.
- 4124           1.2.5. Functional Element returns a Reference ID to user.
- 4125           1.2.6. Functional Element logs the Reference ID to the Record Measurement Use  
4126           Case.
- 4127           1.2.7. Functional Element logs the measurement type to the Record Measurement  
4128           Use Case.
- 4129           1.2.8. Functional Element logs the request time to the Record Measurement Use  
4130           Case.
- 4131           1.2.9 Functional Element logs the response time to the Record Measurement Use  
4132           Case and the use case ends.
- 4133       **2.15.7.2.2.2 Alternative Flows**
- 4134       1. If the structure of the WSDL does not comply with the standard, the Functional Element  
4135       returns an error message and the use case ends.
- 4136       2. If the Functional Element fails to generate the client, the Functional Element returns an error  
4137       message and the use case ends.

- 4138 3. If the Functional Element fails to find the known web service, the Functional Element returns  
4139 an error message and the use case ends.
- 4140 4. If the Functional Element fails to invoke the known web service, the Functional Element  
4141 returns an error message and the use case ends.
- 4142 5. If the Functional Element fails to return a reference ID, the Functional Element returns an  
4143 error message and the use case ends.
- 4144 6. If the Functional Element gets a wrong a reference ID, the Functional Element returns an  
4145 error message and the use case ends.

4146 **2.15.7.2.3 Special Requirements**

4147 None.

4148 **2.15.7.2.4 Pre-Conditions**

4149 None.

4150 **2.15.7.2.5 Post-Conditions**

4151 None.

4152

4153 **2.15.7.3 Measure Reliability**

4154 **2.15.7.3.1 Description**

4155 This use case allows the user to measure the reliability of a known Web Service. User sets a  
4156 period of measurement. The number of failures over a period of time is the measure of Reliability.  
4157 It is derived as the unsuccessful invocations for the given period of measurement.

4158 **2.15.7.3.2 Flow of Events**

4159 **2.15.7.3.2.1 Basic Flow**

- 4160 1. User sets a period of measurement.
- 4161 2. User submits the WSDL of a known web service.
- 4162 3. Functional Element parses the URL of the WSDL document and extracts the necessary  
4163 information.
- 4164 4. Functional Element generates a Reference ID.
- 4165 5. Functional Element returns a Reference ID to user.
- 4166 6. Functional Element logs the Reference ID to the Record Measurement Use Case.
- 4167 7. Functional Element logs measurement type to the Record Measurement Use Case.
- 4168 8. Functional Element waits for any invocation to the known WSDL.
- 4169 9. Functional Element logs unsuccessful invocations to this WSDL until the period of  
4170 measurement is reached and the use case ends.

4171 **2.15.7.3.2.2 Alternative Flows**

- 4172 1. If the structure of the WSDL does not comply with the standard, the Functional Element  
4173 returns an error message and the use case ends.
- 4174 2. If the Functional Element fails to generate the client, the Functional Element returns an error  
4175 message and the use case ends.
- 4176 3. If the Functional Element fails to find the known web service, the Functional Element returns  
4177 an error message and the use case ends.
- 4178 4. If the Functional Element fails to invoke the known web service, the Functional Element  
4179 returns an error message and the use case ends.
- 4180 5. If the Functional Element fails to return a reference ID, the Functional Element returns an  
4181 error message and the use case ends.
- 4182 6. If the Functional Element gets a wrong a reference ID, the Functional Element returns an  
4183 error message and the use case ends.

4184 **2.15.7.3.3 Special Requirements**

4185 None.

4186 **2.15.7.3.4 Pre-Conditions**

4187 None.

4188 **2.15.7.3.5 Post-Conditions**

4189 None.

4190 **2.15.7.4 Measure Accessibility**

4191 **2.15.7.4.1 Description**

4192 This use case allows the user to measure the accessibility of a known Web Service. It is a  
4193 measure denoting the success rate or chance of a successful service instantiation at a point of  
4194 time. User sets the number of times of invocations. User invokes the known web service at the  
4195 number of times set by the user at one go. The result of this measure is in percentage. It is  
4196 derived by using the successful invocations divided by total invocations for the given period of  
4197 measurement:

4198 
$$\text{success rate} = \text{successful invocations} / \text{total invocations} \times 100\% \text{ (invocations are fired simultaneously)}$$

4199

4200 **2.15.7.4.2 Flow of Events**

4201 **2.15.7.4.2.1 Basic Flow**

- 4202 1. User sets number of invocations.
- 4203 2. User submits the WSDL of a known web service.
- 4204 3. Functional Element parses the URL of the WSDL document and extracts the necessary  
4205 information.
- 4206 4. Functional Element generates client base on the extracted information.

- 4207 5. Functional Element invokes the known web service simultaneously at the number of times set  
4208 by the user using the generated client.
- 4209 6. Functional Element generates a Reference ID.
- 4210 7. Functional Element returns a Reference ID to user.
- 4211 8. Functional Element logs the Reference ID to the Record Measurement Use Case.
- 4212 9. Functional Element logs measurement type to the Record Measurement Use Case.
- 4213 10. Functional Element logs each invocation to the Record Measurement Use Case.
- 4214 11. Functional Element logs successful invocation and the use case ends.

#### 4215 **2.15.7.4.2.2 Alternative Flows**

- 4216 1. If the structure of the WSDL does not comply with the standard, the Functional Element  
4217 returns an error message and the use case ends.
- 4218 2. If the Functional Element fails to generate the client, the Functional Element returns an error  
4219 message and the use case ends.
- 4220 3. If the Functional Element fails to find the known web service, the Functional Element returns  
4221 an error message and the use case ends.
- 4222 4. If the Functional Element fails to invoke the known web service, the Functional Element  
4223 returns an error message and the use case ends.
- 4224 5. If the Functional Element fails to return a reference ID, the Functional Element returns an  
4225 error message and the use case ends.
- 4226 6. If the Functional Element gets a wrong a reference ID, the Functional Element returns an  
4227 error message and the use case ends.

#### 4228 **2.15.7.4.3 Special Requirements**

4229 None.

#### 4230 **2.15.7.4.4 Pre-Conditions**

4231 None.

#### 4232 **2.15.7.4.5 Post-Conditions**

4233 None.

4234

4235

### 4236 **2.15.7.5 Record Measurement**

#### 4237 **2.15.7.5.1 Description**

4238 This use case records the Measurement taken. It records type of Measurement, Reference ID,  
4239 and the invocation data (invocation status (Successful or Unsuccessful), request time and  
4240 response time)

4241 **2.15.7.5.2 Flow of Events**

4242 **2.15.7.5.2.1 Basic Flow**

4243 This use case starts when the user record the Measurement.

- 4244 1. The Functional Element logs Reference ID into a log file using Log Utility FE.  
4245 2. The Functional Element logs Measurement type into a log file using Log Utility FE.  
4246 3. The Functional Element logs the invocation data into a log file using Log Utility FE.

4247 **2.15.7.5.2.2 Alternate Flow**

- 4248 1. Log file not available, the Functional Element returns an error and the user case ends.  
4249 2. If the Functional Element fails to get a reference ID, the Functional Element returns an error  
4250 message and the use case ends.

4251 **2.15.7.5.3 Special Requirements**

4252 None.

4253 **2.15.7.5.4 Pre-Conditions**

4254 None.

4255 **2.15.7.5.5 Post-Conditions**

4256 None.

4257

4258 **2.15.7.6 Calculate Measurement**

4259 **2.15.7.6.1 Description**

4260 This use case calculates the Measurement.

4261 **2.15.7.6.2 Flow of Events**

4262 **2.15.7.6.2.1 Basic Flow**

4263 This use case starts when user wants to calculate Measurement.

- 4264 1. The Functional Element gets the Reference ID.  
4265 2. The Functional Element opens up the log file.  
4266 3. The Functional Element reads the data in the log file base on Reference ID given.  
4267 4. The Functional Element calculates the measurement using the data read from the log file.  
4268 5. The Functional Element sends the calculated result to the user.

4269 **2.15.7.6.2.2 Alternative Flows**

- 4270 1. Log file not available, the Functional Element returns an error and the user case ends.

4271 2. If the Functional Element fails to get a reference ID, the Functional Element returns an error  
4272 message and the use case ends.

### 4273 **2.15.7.6.3 Special Requirements**

4274 None.

### 4275 **2.15.7.6.4 Pre-Conditions**

4276 None.

### 4277 **2.15.7.6.5 Post-Conditions**

4278 None.

4279

## 4280 **2.15.7.7 Get Result**

### 4281 **2.15.7.7.1 Description**

4282 This use case calculates the Measurement logged.

### 4283 **2.15.7.7.2 Flow of Events**

#### 4284 **2.15.7.7.2.1 Basic Flow**

4285 This use case starts when user wanted to get result base on the Reference ID.

4286 1. The Functional Element gets the Reference ID from user

4287 2. The Functional Element passes the Reference ID to Calculate Measurement Use Case.

4288 3. The Functional Element gets calculated result.

4289 4. The Functional Element returns the result to the user.

#### 4290 **2.15.7.7.2.2 Alternative Flows**

4291 1. Log file not available, the Functional Element returns an error and the user case ends.

4292 2. If the Functional Element fails to get a reference ID, the Functional Element returns an error  
4293 message and the use case ends.

### 4294 **2.15.7.7.3 Special Requirements**

4295 None.

### 4296 **2.15.7.7.4 Pre-Conditions**

4297 None.

### 4298 **2.15.7.7.5 Post-Conditions**

4299 None.

4300



4301     **2.15.7.8       Manage Security**

4302     **2.15.7.8.1    Description**

4303     This use case allows user to check that the known web service is securely managed.

4304     **2.15.7.8.2    Flow of Events**

4305     **2.15.7.8.2.1   Basic Flow**

- 4306     1.   The service provider sends a request to check security of the known web service.
- 4307     2.   User submits the WSDL of a known web service.
- 4308     3.   Functional Element parses the URL of the WSDL document and extracts the necessary  
4309         information.
- 4310     4.   Functional Element generates client base on the extracted information.
- 4311     5.   Functional Element invokes the known web service with a username.
- 4312     6.   User sends a message to the known web service.
- 4313     7.   The Functional Element checks whether username is authenticated.
- 4314     8.   The Functional Element checks whether message is encrypted.
- 4315     9.   The Functional Element checks whether the whole process is access controlled.
- 4316     10.  The Functional Element returns the outcome to the user and the use case ends.

4317     **2.15.7.8.2.2   Alternative Flows**

- 4318     1.   If the structure of the WSDL does not comply with the standard, the Functional Element  
4319         returns an error message and the use case ends.
- 4320     2.   If the Functional Element fails to generate the client, the Functional Element returns an error  
4321         message and the use case ends.
- 4322     3.   If the Functional Element fails to find the known web service, the Functional Element returns  
4323         an error message and the use case ends.
- 4324     4.   If the Functional Element fails to invoke the known web service, the Functional Element  
4325         returns an error message and the use case ends.
- 4326     5.   If the web service fails to return result, the Functional Element returns an error message and  
4327         the use case ends.

4328     **2.15.7.8.3    Special Requirements**

4329     None.

4330     **2.15.7.8.4    Pre-Conditions**

4331     None.

4332     **2.15.7.8.5    Post-Conditions**

4333     None.

## 2.16 Role and Access Management Functional Element

### 2.16.1 Motivation

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

Management of roles and access privileges, and

Assignment of roles to entities that will be accessing the resources that is being managed.

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

Primary Requirements

- MANAGEMENT-030 to MANAGEMENT-034, and
- MANAGEMENT-200 to MANAGEMENT-205.

Secondary Requirements

- SECURITY-040 to SECURITY-041.

### 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	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>

4353

### 4354 2.16.3 Key Features

4355 Implementations of the Secure SOAP Functional Element are expected to provide the following  
4356 key features:

- 4357 1. The Functional Element MUST provide the capability to manage the creation and deletion of  
4358 instances of the following concepts based on a pre-defined structure:
  - 4359 1.1. Role,
  - 4360 1.2. Access, and
  - 4361 1.3. Resource
- 4362 16. The Functional Element MUST provide the capability to manage all the information (attribute  
4363 values) stored in such concepts. This includes the capability to retrieve and update attribute's  
4364 values belonging to a concept like Role, Access or Resource.
- 4365 17. The Functional Element MUST provide the capability to associate a Role to its access  
4366 privileges through the Access structure.
- 4367 18. The Functional Element MUST provide the capability to determine a Role's accessibility to  
4368 Resources based on the access privileges that have been assigned.
- 4369 19. The Functional Element MUST provide the ability to manage the association of users to  
4370 Roles via assignments of Roles to users. This will include:
  - 4371 1.4. Assignment/Un-assignment of Roles to individual Users, and
  - 4372 1.5. Assignment/Un-assignment of Roles to Groups.

4373 This will provide an indirect linkage between the accessibility of specific Users to Resources  
4374 through the concept of Role and Access.
- 4375 20. The Functional Element MUST provide a mechanism for managing the concepts of Role,  
4376 Access and Resource across different application domains.

4377 *Example: Namespace control mechanism*

4378

4379 In addition, the following key features could be provided to enhance the Functional Element  
4380 further:

- 4381 1. The Functional Element MAY provide a mechanism to enable different Access instances to  
4382 be related to one another.
- 4383 2. The Functional Element MAY also provide a mechanism to enable hierarchical  
4384 relationships between Access instances.

4385 *Example: Parent and Child Relationship*

4386 3. The Functional Element MAY provide the ability for Roles to be temporal sensitive.

4387 *Example: A Role is assigned to a particular Phase in a Lifecycle.*

4388

#### 4389 **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.

#### 4390 **2.16.5 Related Technologies and Standards**

4391 None

4392

4393      **2.16.6      Model**

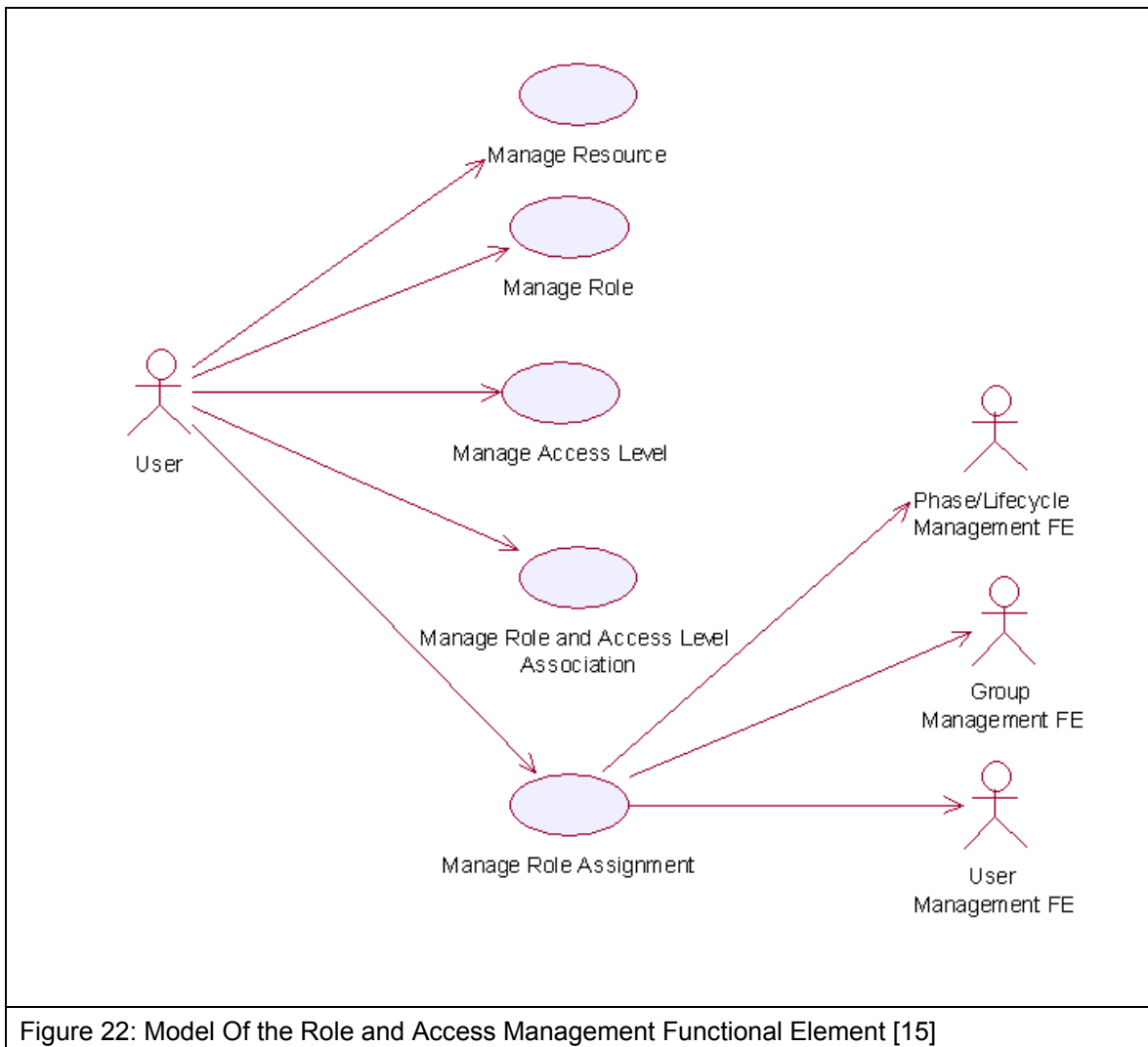


Figure 22: Model Of the Role and Access Management Functional Element [15]

4394

4395      **2.16.7      Usage Scenario**

4396      **2.16.7.1      Manage Role**

4397      **2.16.7.1.1      Description**

4398      This use case allows the service user to manipulate the role information such as adding,  
4399      changing and deleting role information in the Functional Element.

4400      **2.16.7.1.2      Flow of Events**

4401      **2.16.7.1.2.1      Basic Flow**

4402      This use case starts when any user wants to create, change or delete a role.

4403 1: Service user specifies the function it would like to perform (either create a role, update a role or  
4404 delete a role).

4405 2: Once the service user provides the requested information, one of the sub-flows is executed.

4406 If the service user provides '**Create a Role**', then sub-flow 2.1 is executed.

4407 If the service user provides '**Retrieve a Role**', then sub-flow 2.2 is executed.

4408 If the service user provides '**Update a Role**', then sub-flow 2.3 is executed.

4409 If the service user provides '**Delete a Role**', then sub-flow 2.4 is executed.

4410 2.1: Create a Role.

4411 2.1.1: The service user specifies role information such as the role name and description.

4412 2.1.2: The Functional Element connects to the data storage.

4413 2.1.3: The Functional Element checks whether the role exists in the Functional Element  
4414 or not, saves the role information in the data storage and the use case ends.

4415 2.2: Retrieve a Role.

4416 2.2.1: The service user specifies the role name for retrieval.

4417 2.2.2: The Functional Element connects to the data storage.

4418 2.2.3: The Functional Element retrieves the role information in the data storage and the  
4419 use case ends.

4420 2.3: Update a Role.

4421 2.3.1: The service user specifies the role name to update.

4422 2.3.2: The service user specifies the target field name and value of the role.

4423 2.3.3: The Functional Element connects to the data storage.

4424 2.3.4: The Functional Element updates the role information in the data storage and the  
4425 use case ends.

4426 2.4: Delete a Role.

4427 2.4.1: The service user specifies the role name to delete.

4428 2.4.2: The Functional Element connects to the data storage.

4429 2.4.3: The Functional Element removes the record of the role in the data storage and the  
4430 use case ends.

4431 **2.16.7.1.2.2 Alternative Flows**

4432 1: Data Storage Not Available.

4433 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  
4434 available, an error message is returned and the use case ends.

4435 2: Role Already Exists.

4436 2.1: If in basic flow 2.1.3, the Functional Element checks that the role already exists in the  
4437 data storage, an error message is returned and the use case ends.

4438 3: Role Does Not Exist.

4439 3.1: If in basic flow 2.2.3, 2.3.4 and 2.4.3, the Functional Element checks that the role does  
4440 not exist in the data storage, an error message is returned and the use case ends.

4441 4: Role Cannot Be Deleted.

4442 4.1: If in basic flow 2.4.3, the other information associated with the role, such as any access  
4443 level assigned, still exists, the role information may not be removed. An error message is  
4444 returned and the use case ends.

4445 **2.16.7.1.3 Special Requirements**

4446 None

4447 **2.16.7.1.4 Pre-Conditions**

4448 None.

4449 **2.16.7.1.5 Post-Conditions**

4450 If the use case was successful, the role is saved/updated/removed in the Functional Element.  
4451 Otherwise, the Functional Element state is unchanged.

4452 **2.16.7.2 Manage Resource**

4453 **2.16.7.2.1 Description**

4454 This use case allows the service user to manipulate the resource information such as adding,  
4455 changing and deleting resource information in the Functional Element.

4456 **2.16.7.2.2 Flow of Events**

4457 **2.16.7.2.2.1 Basic Flow**

4458 This use case starts when any user wants to create, change or delete a resource.

4459 1: The user specifies the function it would like to perform.

4460 2: The user provides the requested information, one of the sub-flows is executed.

4461 If the user provides '**Create a Resource**', then sub-flow 2.1 is executed.

4462 If the user provides '**Retrieve a Resource**', then sub-flow 2.2 is executed.

4463 If the user provides '**Update a Resource**', then sub-flow 2.3 is executed.

4464 If the user provides '**Delete a Resource**', then sub-flow 2.4 is executed.

4465 2.1: Create a Resource.

4466 2.1.1: The user specifies resource information such as the resource name and  
4467 description.

4468 2.1.2: The Functional Element connects to the data storage.

4469            2.1.3: The Functional Element checks whether the resource exists in the Functional  
4470            Element, saves the resource information in the data storage and the use case ends.

4471            2.2: Retrieve a Resource.

4472            2.2.1: The service user specifies the resource name for retrieval.

4473            2.2.2: The Functional Element connects to the data storage.

4474            2.2.3: The Functional Element retrieves the resource information in the data storage and  
4475            the use case ends.

4476            2.3: Update a Resource.

4477            2.3.1: The service user specifies the resource name to update.

4478            2.3.2: The Functional Element connects to the data storage.

4479            2.3.3: The Functional Element updates the resource information in the data storage and  
4480            the use case ends.

4481            2.4: Delete a Resource.

4482            2.4.1: The service user specifies the resource name to delete.

4483            2.4.2: The Functional Element connects to the data storage.

4484            2.4.3: The Functional Element removes the record of the resource in the data storage  
4485            and the use case ends.

4486            **2.16.7.2.2.2    Alternative Flows**

4487            1: Data Storage Not Available.

4488            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  
4489            is not available, an error message is returned and the use case ends.

4490            2: Resource Already Exists.

4491            2.1: If in basic flow 2.1.3, the Functional Element checks that the resource already exists in  
4492            the data storage, an error message is returned and the use case ends.

4493            3: Resource Does Not Exist.

4494            3.1: If in basic flow 2.2.3, 2.3.3 and 2.4.3, the Functional Element checks that the resource  
4495            does not exist in the data storage, an error message is returned and the use case ends.

4496            **2.16.7.2.3    Special Requirements**

4497            None

4498            **2.16.7.2.4    Pre-Conditions**

4499            None.

4500            **2.16.7.2.5    Post-Conditions**

4501            None



### 4502    **2.16.7.3      Manage Access Level**

#### 4503    **2.16.7.3.1    Description**

4504    This use case allows service user to manage the creation/retrieval/modification/deletion of access  
4505    level.

#### 4506    **2.16.7.3.2    Flow of Events**

##### 4507    **2.16.7.3.2.1   Basic Flow**

4508    This use case starts when service user wants to manage the access levels.

4509    1: The service user specifies the function it would like to perform (add, update or delete an  
4510    access level).

4511    2: Once the service user provides the requested information, one of the sub-flows is executed.

4512    If the service user provides '**Add an Access Level**', then sub-flow 2.1 is executed.

4513    If the service user provides '**Retrieve an Access Level**', then sub-flow 2.2 is activated.

4514    If the service user provides '**Update an Access Level**', then sub-flow 2.3 is activated.

4515    If the service user provides '**Delete an Access Level**', then sub-flow 2.4 is executed.

4516        2.1: Add an Access Level.

4517            2.1.1: The service user specifies the access level information, which includes: name,  
4518            description, name of parent access level and group of resources that the access level is  
4519            associated with.

4520            2.1.2: The Functional Element connects to the data storage.

4521            2.1.3: The Functional Element check whether the access level and its parent access level  
4522            exist in the Functional Element, saves the access level information in the data storage  
4523            and the use case ends.

4524        2.2: Retrieve an Access Level.

4525            2.2.1: The service user specifies the access level name to retrieve.

4526            2.2.2: The Functional Element connects to the data storage.

4527            2.2.3: The Functional Element gets access level information from the data storage and  
4528            returns to the service user and the use case ends.

4529        2.3: Update an Access Level.

4530            2.3.1: The service user specifies the access level name.

4531            2.3.2: The service user specifies the field(s) and new value(s) to update.

4532            2.3.3: The Functional Element connects to the data storage.

4533            2.3.4: The Functional Element updates the access level information in the data storage  
4534            with the value specified in 2.3.2 and the use case ends.

4535        2.4: Delete an Access Level.

4536 2.4.1: The service user specifies the access level name to delete.  
4537 2.4.2: The Functional Element connects to the data storage.  
4538 2.4.3: The Functional Element removes the record of the access level in the data storage  
4539 and the use case ends.

#### 4540 **2.16.7.3.2.2 Alternative Flows**

4541 1: Data Storage Not Available.

4542 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  
4543 information is not available, an error message is returned and the use case ends.

4544 2: Access Level Already Exists.

4545 2.1: If in basic flow 2.1.3, the Functional Element checks that the access level already exists  
4546 in the data storage, an error message is returned and the use case ends.

4547 3: Access Level Cannot Be Deleted.

4548 3.1: If in basic flow 2.4.3, the other information associated with the Access Level, such as  
4549 roles to which the access level is assigned and the parent access level still exists, the access  
4550 level information may not be removed. An error message is returned and the use case ends.

4551 4: Parent Access Level Not Exist.

4552 4.1: If in basic flow 2.1.3, the parent access level does not exist, an error message is returned  
4553 and the use case ends.

#### 4554 **2.16.7.3.3 Special Requirements**

4555 None

#### 4556 **2.16.7.3.4 Pre-Conditions**

4557 None.

#### 4558 **2.16.7.3.5 Post-Conditions**

4559 None

### 4560 **2.16.7.4 Manage Role and Access Level Association**

#### 4561 **2.16.7.4.1 Description**

4562 This use case allows service user to assign, update and remove the access level assigned to  
4563 role.

#### 4564 **2.16.7.4.2 Flow of Events**

##### 4565 **2.16.7.4.2.1 Basic Flow**

4566 This use case starts when service user wants to manage the relationship between access level  
4567 and role.

4568 1: The service user specifies a role and the function he/she would like to perform on the role  
4569 (either assign an access level to role, update role access level, or delete role access level).

4570 2: Once the service user provides the requested information, one of the sub-flows is executed.

4571 If the user provides '**Assign an Access Level to Role**', then sub-flow 2.1 is executed.

4572 If the user provides '**Update Access Level for Role**', then sub-flow 2.2 is executed.

4573 If the user provides '**Delete Access Level for Role**', then sub-flow 2.3 is executed.

4574 If the user provides '**Retrieve Access Level for Role**', then sub-flow 2.4 is executed.

4575 If the service user provides '**Retrieve Role for Access Level**', then sub-flow 2.5 is executed.

4576 2.1: Assign an Access Level to Role.

4577 2.1.1: The service user specifies access level that will be assigned to the role.

4578 2.1.2: The Functional Element connects to the data storage.

4579 2.1.3: The Functional Element checks whether the access level has been assigned to the

4580 role. Functional Element saves the access level reference in the role record in the data

4581 storage and the use case ends.

4582 2.2: Update Access Level for Role.

4583 2.2.1: The service user specifies the access level to update and the new access level

4584 information.

4585 2.2.2: The Functional Element connects to the data storage.

4586 2.2.3: The Functional Element updates the access level reference in the role record in the

4587 data storage and the use case ends.

4588 2.3: Delete Access Level to Role.

4589 2.3.1: The service user specifies the access level to delete.

4590 2.3.2: The Functional Element connects to the data storage.

4591 2.3.3: The Functional Element removes the access level reference from the record of the

4592 role in the data storage and the use case ends.

4593 2.4: Retrieve Access Level for Role.

4594 2.4.1: The service user specifies the role to retrieve the access levels associated with it.

4595 2.4.2: The Functional Element connects to the data storage.

4596 2.4.3: The Functional Element retrieves the access level assigned to the role in the data

4597 storage and the use case ends.

4598 2.5: Retrieve Role for Access Level.

4599 2.5.1: The service user specifies the access level to retrieve roles associated to it.

4600 2.5.2: The Functional Element connects to the data storage.

4601 2.5.3: The Functional Element retrieves roles associated to the access level in the data

4602 storage and the use case ends.

#### 4603 **2.16.7.4.2.2 Alternative Flows**

4604 1: Data Storage Not Available.

4605 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  
4606 not available, an error message is returned and the use case ends.

4607 2: Access Level Assignment Already Exists.

4608 2.1: If in basic flow 2.1.3, the Functional Element checks that the access level already exists  
4609 in the role record in the data storage, an error message is returned and the use case ends.

4610 3: Access Level Assignment Not Exist.

4611 3.1: If in basic flow 2.3.3, the access level assignment does not exist, an error message is  
4612 returned and the use case ends.

4613 4: Access Level Not Exist.

4614 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  
4615 error message is returned and the use case ends.

4616 5: Role Not Exist.

4617 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  
4618 message is returned and the use case ends.

4619 **2.16.7.4.3 Special Requirements**

4620 None.

4621 **2.16.7.4.4 Pre-Conditions**

4622 None.

4623 **2.16.7.4.5 Post-Conditions**

4624 None.

4625 **2.16.7.5 Manage Role Assignment**

4626 **2.16.7.5.1 Description**

4627 The use case allows service user to assign a role to a user, a group, a phase in a lifecycle, to  
4628 change or to delete such assignment.

4629 **2.16.7.5.2 Flow of Events**

4630 **2.16.7.5.2.1 Basic Flow**

4631 This use case starts when the service user wants to manage the assignment of a role. This role  
4632 can be assigned to a user, group, phase and lifecycle.

4633 1: Service user specifies a role and an operation to perform on the role.

4634 2: Once the service user provides the requested information, one of the sub-flows is executed.

4635 If the user provides 'Assign Role', then sub-flow 2.1 is executed.

4636 If the user provides 'Retrieve Role', then sub-flow 2.2 is executed.

4637 If the user provides 'Un-assign Role', then user sub-flow 2.3 is executed.

4638 2.1: Assign Role.

4639 2.1.1: The service user specifies a user/group/phase/lifecycle to which the role will be  
4640 assigned.

4641 2.1.2: Depending of target of the assignment, the Functional Element will check for the  
4642 presence of one of the following Functional Elements.

4643 User Management Functional Element

4644 Group Management Functional Element

4645 Phase and Lifecycle Management Functional Element

4646 2.1.3: The Functional Element checks whether the role has been assigned to the  
4647 intended target

4648 2.1.4: The Functional Element saves the relationship between the role and the target and  
4649 the use case ends.

4650 2.2: Retrieve Role.

4651 2.2.1: The service user specifies a user/group/phase/lifecycle to retrieve all roles  
4652 assigned

4653 2.2.2: Depending of target of the assignment, the Functional Element will check for the  
4654 presence of one of the following Functional Elements.

4655 User Management Functional Element

4656 Group Management Functional Element

4657 Phase and Lifecycle Management Functional Element

4658 2.2.3: The Functional Element gets the roles that are assigned to the target.

4659 2.2.4: The Functional Element returns the results to the service user and the use case  
4660 ends.

4661 2.3: Un-assign Role.

4662 2.3.1: The service user specifies a user/group/phase/lifecycle and the role that is to be  
4663 un-assigned.

4664 2.3.2: Depending of target of this un-assignment, the Functional Element will check for  
4665 the presence of one of the following Functional Elements.

4666 User Management Functional Element

4667 Group Management Functional Element

4668 Phase and Lifecycle Management Functional Element

4669 2.3.3: The Functional Element checks if the roles have been assigned to the target in the  
4670 first place.

4671 2.3.4: The Functional Element removes the role assigned and the use case ends.

4672 **2.16.7.5.2.2 Alternative Flows**

4673 1: Dependent Functional Element not available.

4674 1.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the dependent Functional Elements are not  
4675 available, an error message is returned and the use case ends.

4676 2: Invalid User/Group/Phase/Lifecycle Account.

4677 2.1: If in basic flow 2.1.2, 2.2.2 and 2.3.2, the dependent Functional Elements are available  
4678 but an invalid account is provided, an error message is returned and the use case ends.

4679 3: Data Storage Not Available.

4680 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  
4681 storage, an error message is provided and the use case ends.

4682

4683 **2.16.7.5.3 Special Requirements**

4684 None.

4685 **2.16.7.5.4 Pre-Conditions**

4686 None.

4687 **2.16.7.5.5 Post-Conditions**

4688 None.

## 2.17 Search Functional Element

### 2.17.1 Motivation

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

Providing the capability for configuration of different types of data sources for information search,

Providing the facility to provide a concrete definition of data source classification for information search,

Providing the ability to define different search scopes for various data source classification,

Performing information search on those pre-configured different types of data sources and

Providing the provision to consolidate the return result arising from the search operation.

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

Primary Requirements

- MANAGEMENT-009,
- PROCESS-030 to PROCESS-031, and
- PROCESS-034.

Secondary Requirements

- None

### 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
------	---------------------------

### 4712 **2.17.3 Key Features**

4713 Implementations of the Search Functional Element are expected to provide the following key  
4714 features:

- 4715 1. The Functional Element MUST provide a mechanism to define and manage Search  
4716 Categories.
- 4717 2. The Functional Element MUST provide the capability to configure and store information  
4718 about targeted data sources for a particular Search Category.  
4719 *Example: Some of the stored information would include Location, Type, Name, Data Fields*  
4720 *(of interest to the search) and access control (typically username and password) of the*  
4721 *targeted data source.*
- 4722 3. As part of Key Feature (2), the Functional Element MUST also provide the ability to  
4723 configure the scope of search and returned results.
- 4724 4. The Functional Element MUST also provide a mechanism to link the Search Categories to  
4725 configured target data sources.
- 4726 5. The Functional Element MUST provide the ability to search multiple data sources for a  
4727 defined Search Category.  
4728 *Example: Some of the common data sources would include RDBMS, XML DB, LDAP*  
4729 *servers and flat files like XML files, text files and HTML files*
- 4730 21. The Functional Element MUST provide the ability to perform searches based on a given set  
4731 of keyword(s).

4732

4733 In addition, the following key features could be provided to enhance the Functional Element  
4734 further:

- 4735 1. The Functional Element MAY also provide the ability to perform conditional and parametric  
4736 searches.
- 4737 22. The Functional Element MAY also provide the ability to restrict the scope of a search.  
4738 *Example: By providing a particular Search Category or types of data sources for the*  
4739 *search.*

4740

### 4741 **2.17.4 Interdependencies**

4742 None

4743

### 4744 **2.17.5 Related Technologies and Standards**

4745 None



4746 **2.17.6 Model**

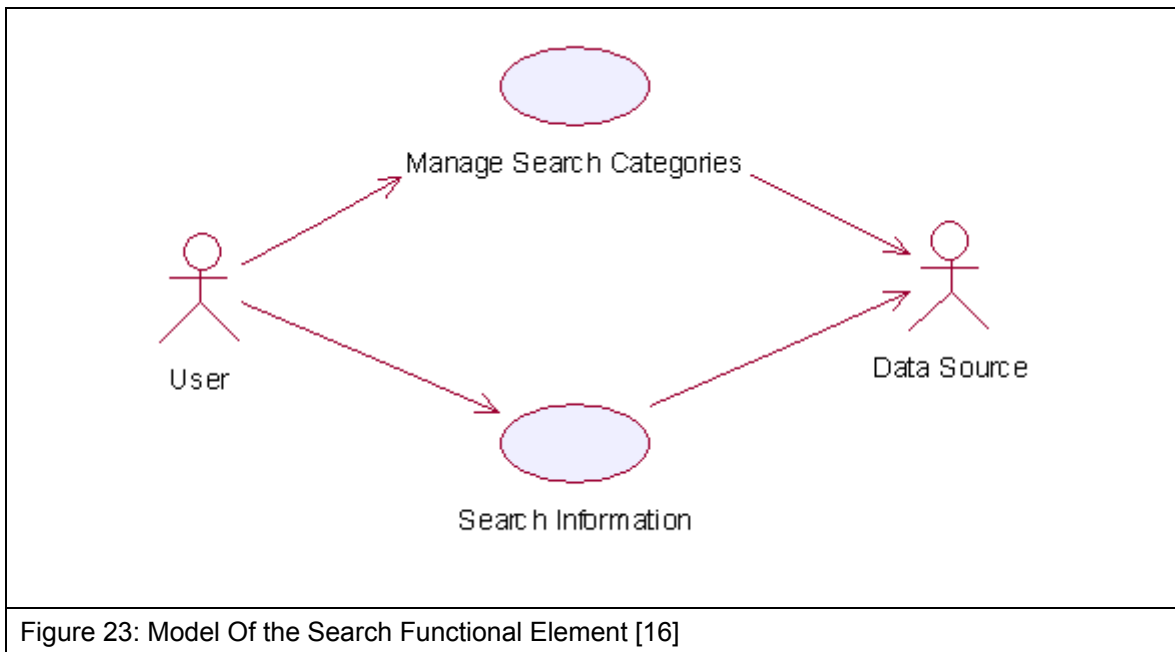


Figure 23: Model Of the Search Functional Element [16]

4747 **2.17.7 Usage Scenario**

4748 **2.17.7.1 Manage Search Categories**

4749 **2.17.7.1.1 Description**

4750 This use case allows the users to manage the different search categories.

4751 **2.17.7.1.2 Flow of Events**

4752 **2.17.7.1.2.1 Basic Flow**

4753 This use case starts when the user wishes to manage the different data sources for search to be  
4754 performed on it.

4755 1: The users initiates a request to configure data source(s) and type(s) by providing the data  
4756 source information and type to be added, removed or retrieved.

4757 2: The Functional Element checks whether the data source configuration file exists.

4758 3: The Functional Element checks the request. Based on the type of request, one of the sub-  
4759 flows is executed.

4760 If the request is to '**Create Data Source And Type**', then sub-flow 3.1 is executed.

4761 If the request is to '**View Data Sources And Types**', then sub-flow 3.2 is executed.

4762 If the request is to '**Delete Data Source And Type**', then sub-flow 3.3 is executed.

4763 3.1: Create Data Source and Type.

4764 3.1.1: The Functional Element checks whether the same data source and type has been  
4765 created.

4766 3.1.2: The Functional Element appends the new data source and type in the data source  
4767 configuration file specified.

4768 3.2: View Data Source and Type.

4769 3.2.1: The Functional Element retrieves all the data source and type information from the  
4770 data source configuration file.

4771 3.2.2: The Functional Element returns the data source(s) and type(s).

4772 3.3: Delete Data Source and Type.

4773 3.3.1: The Functional Element checks whether the data source and type exist in the data  
4774 source configuration based on data source id from the data source configuration file.

4775 3.3.2: The Functional Element removes the old data source and type from the data  
4776 source configuration file.

4777 4: The Functional Element returns a success or failure flag indicating the status of the operation  
4778 being performed and use case ends.

4779 **2.17.7.1.2.2 Alternative Flows**

4780 1: Data Source Configuration File Not Found.

4781 1.1: If in basic flow 2, the data source configuration file does not exist, the Functional Element  
4782 creates an empty data source configuration file.

4783 2: Duplicate Data Source and Type.

4784 2.1: If in basic flow 3.1.1, the same data source and type have been configured, the  
4785 Functional Element returns an error message and the use case end.

4786 3: Data Source and Type Do Not Exist.

4787 3.1: If in basic flow 3.2.1 and 3.3.1, a particular data source and type cannot be found in the  
4788 specified data source configuration file, the Functional Element returns an error message and  
4789 the use case end.

4790 **2.17.7.1.3 Special Requirements**

4791 None.

4792 **2.17.7.1.4 Pre-Conditions**

4793 None.

4794 **2.17.7.1.5 Post-Conditions**

4795 None.

4796 **2.17.7.2 Search Information**

4797 **2.17.7.2.1 Description**

4798 This use case allows any users to perform search on various disparate data sources and types  
4799 configured to be searched and returns the matching results.

4800     **2.17.7.2.2     Flow of Events**

4801     **2.17.7.2.2.1    Basic Flow**

4802     This use case starts when users wishes to perform information search on a data source.

4803     1: Users initiates a request to perform information search on a given data source by providing  
4804     information to be searched, location of the data source(s) and the data source type(s).

4805     2: The Functional Element checks for the existence of the specified data source(s).

4806     3: The Functional Element validates the data source type(s) against the set of supported data  
4807     type(s) configured within the Functional Element that are available for information search.

4808     4: The Functional Element performs information search based on the search parameters given by  
4809     the users or the other Functional Elements.

4810     5: The Functional Element returns the result of the information search performed to the users or  
4811     other Functional Elements and use case ends.

4812     **2.17.7.2.2.2    Alternative Flows**

4813     1: Data Source(s) Are Not Available.

4814         1.1: In basic flow 2, if the identified data source is not available, the Functional Element  
4815         returns an error message and the use case ends.

4816     2: Invalid Configuration Instructions

4817         2.1: In basic flow 2, if the input inform by the user is incomplete, the Functional Element  
4818         returns an error message and the use case ends.

4819     3: Invalid Data Source Type.

4820         3.1: In basic flow 3, if the data source type is invalid, the Functional Element returns an error  
4821         message and the use case ends.

4822     4: No Matching Result.

4823         4.1: In basic flow 4, if the search results in no matching results, the Functional Element  
4824         returns an error message and the use case ends..

4825     **2.17.7.2.3     Special Requirements**

4826     None

4827     **2.17.7.2.4     Pre-Conditions**

4828     None.

4829     **2.17.7.2.5     Post-Conditions**

4830     None.

4831

## 2.18 Secure SOAP Management Functional Element

### 2.18.1 Motivation

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

The facility of digitally signing SOAP message,

The facility of encrypting SOAP message, and

The capability to generate the original SOAP message after signing or encrypting the message.

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

#### Primary Requirements

- SECURITY-003 (SECURITY-003-3 only),
- SECURITY-020 (all), and
- SECURITY-022, and
- SECURITY-026.

#### Secondary Requirements

- None

### 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.

### 2.18.3 Key Features

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

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

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

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

### 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.

### 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 <sup>th</sup> 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 <sup>th</sup> Dec 2002 [18]	<p>This specification addresses data privacy by defining a process for encrypting data and representing the result in XML document.</p> <p>In this Functional Element, the encryption and decryption of SOAP messages are done based on this specification.</p>
---	---

4877

4878

## 4879 2.18.6 Model

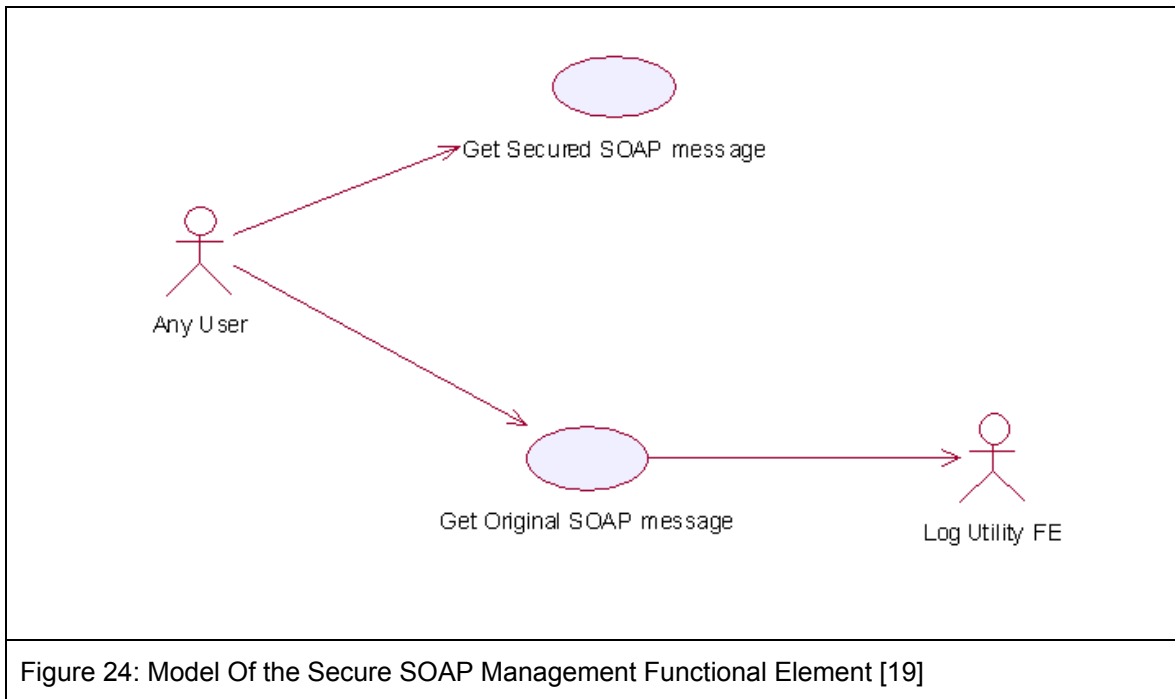


Figure 24: Model Of the Secure SOAP Management Functional Element [19]

## 4880 2.18.7 Usage Scenarios

### 4881 2.18.7.1 Get Secured SOAP message

#### 4882 2.18.7.1.1 Description

4883 This Functional Element describes the process to generate secured SOAP message.

#### 4884 2.18.7.1.2 Flow of Events

##### 4885 2.18.7.1.2.1 Basic Flow

4886 This use case starts when the user wants to secure the SOAP message.

4887 If user wants to '**Sign SOAP message**', then basic flow 1 is executed.

4888 If user wants to '**Encrypt and Sign the SOAP message**', then basic flow 2 is executed.

4889 1: Sign SOAP Message.

4890 1.1: User sends the SOAP message, digital certificate and specifies the element name that  
4891 needs to be signed.

4892 1.2: Functional Element gets the key information from the digital certificate.

4893 *Note: The private key will be used to sign the SOAP message and the public key will be*  
4894 *added to the SOAP message after the signing.*

4895 1.3: Functional Element signs the element.

4896 *Note: The digital signature format is expected to be based on XML-Digital Signature Syntax*  
4897 *mentioned in section 3.10.5.*

4898 1.4: Functional Element parses the secure SOAP message and regenerates the SOAP  
4899 message.

4900 1.5: Functional Element returns the secured SOAP message to user and the use case ends.

4901 2: Encrypt And Sign SOAP Message.

4902 2.1: User sends the SOAP message, digital certificate and specify the element name that  
4903 needs to be encrypted.

4904 2.2: User sends the receiver's public key information to Functional Element.

4905 *Note: Receiver's public key will be used to encrypt the session key, which was then used to*  
4906 *encrypt the content of the element in the SOAP message.*

4907 2.3: Functional Element gets key information from the user's digital certificate.

4908 *Note: Private key is used to sign the SOAP message and public key is used to add into the*  
4909 *SOAP message after the signing.*

4910 2.4: Functional Element generates the session key.

4911 *Note: Session key is used to encrypt the content of the element.*

4912 2.5: Functional Element encrypts the content of element with the session key.

4913 2.6: Functional Element encrypts session key with the receiver's public key.

4914 2.7: Functional Element signs the SOAP message after encryption.

4915 2.8: Functional Element regenerates the SOAP message.

4916 *Note: Functional Element adds the encrypted content of the element, encrypted session key*  
4917 *information, the receiver's public key information and the signature to the SOAP message.*

4918 2.9: Functional Element returns the SOAP message and the use case ends.

#### 4919 **2.18.7.1.2.2 Alternative Flows**

4920 1: Cannot Get Key.

4921 1.1: In basic flow 1.2 and 2.3, Functional Element cannot get the key information from the  
4922 digital certificate. The Functional Element returns an error message and the use case ends.

4923 2: Cannot Sign

4924 2.1: In basic flow 1.3, Functional Element cannot sign the SOAP message. The Functional  
4925 Element returns an error message and the use case ends.

4926 3: Cannot Encrypt

4927 3.1: In basic flow 2.5, Functional Element cannot encrypt the SOAP message. The Functional  
4928 Element returns an error message and the use case ends.

4929 **2.18.7.1.3 Special Requirements**

4930 None.

4931 **2.18.7.1.4 Pre-Conditions**

4932 None.

4933 **2.18.7.1.5 Post-Conditions**

4934 None.

4935 **2.18.7.2 Get Original SOAP Message**

4936 **2.18.7.2.1 Description**

4937 This use case allows users to get original SOAP message.

4938 **2.18.7.2.2 Flow of Events**

4939 **2.18.7.2.2.1 Basic Flow**

4940 This use case starts when the user wants to get the original SOAP message.

4941 If the user wants to '**Verify the SOAP message**', then basic flow 1 is executed.

4942 If the user wants to '**Decrypt and Verify the SOAP message**', then basic flow 2 is executed.

4943 1: Verify SOAP Message.

4944 1.1: User sends the SOAP message and sender's digital certificate.

4945 1.2: Functional Element verifies the SOAP message.

4946 *Note: The sender's certificate information will be used to verify the signature.*

4947 1.3: Functional Element gets the original SOAP message, returns to user and the use case  
4948 ends.

4949 2: Decrypt And Verify The SOAP Message.

4950 2.1: User sends the SOAP message, user's digital certificate and sender's certificate.

4951 2.2: Functional Element verifies the SOAP message.

4952 *Note: The sender's certificate information will be used to verify the signature.*

4953 2.3: Functional Element gets the user's key information from the user's digital certificate.

4954 *Note: The user's private key will be used to decrypt the session key.*



- 4955 2.4: Functional Element decrypts the session key.
- 4956 2.5: Functional Element decrypts the content of the element with the session key.
- 4957 2.6: Functional Element regenerates the SOAP message.
- 4958 *Note: Functional Element removes the session key information and the digital signature*  
4959 *information from the SOAP message and gets the original one.*
- 4960 2.7: Functional Element returns the original SOAP message to user and the use case ends.
- 4961 **2.18.7.2.2.2 Alternative Flows**
- 4962 1: Verification Fails.
- 4963 1.1: In basic flow 1.3 and 2.3, if verification fails, the Functional Element returns an error  
4964 message and the use case ends.
- 4965 2: Decryption of Content Fails.
- 4966 2.1: In basic flow 2.5, the Functional Element cannot decrypt the content of the element. The  
4967 Functional Element returns an error message and the use case ends.
- 4968 **2.18.7.2.3 Special Requirements**
- 4969 None
- 4970 **2.18.7.2.4 Pre-Conditions**
- 4971 None.
- 4972 **2.18.7.2.5 Post-Conditions**
- 4973 None.

## 2.19 Sensory Functional Element

### 2.19.1 Motivation

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

Determining the presentation capabilities by inspecting incoming headers, and

Determining the presentation capabilities by extracting MIME information from the relevant headers.

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

Primary Requirements

- DELIVERY-001,
- DELIVERY-005 to DELIVERY-006, and
- DELIVERY-009.

Secondary Requirements

- MANAGEMENT-011, and
- MANAGEMENT-096.

### 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.

### 2.19.3 Key Features

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

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

5001 24. The Functional Element MUST provide the mechanism to manage MIME types, including the  
 5002 ability to add, delete and retrieve supported MIME types.

5003

5004 In addition, the following key features could be provided to enhance the Functional Element  
 5005 further:

5006 1. The Functional Element MAY provide a mechanism to enable Location Based Services  
 5007 (LBS).

## 5008 2.19.4 Interdependencies

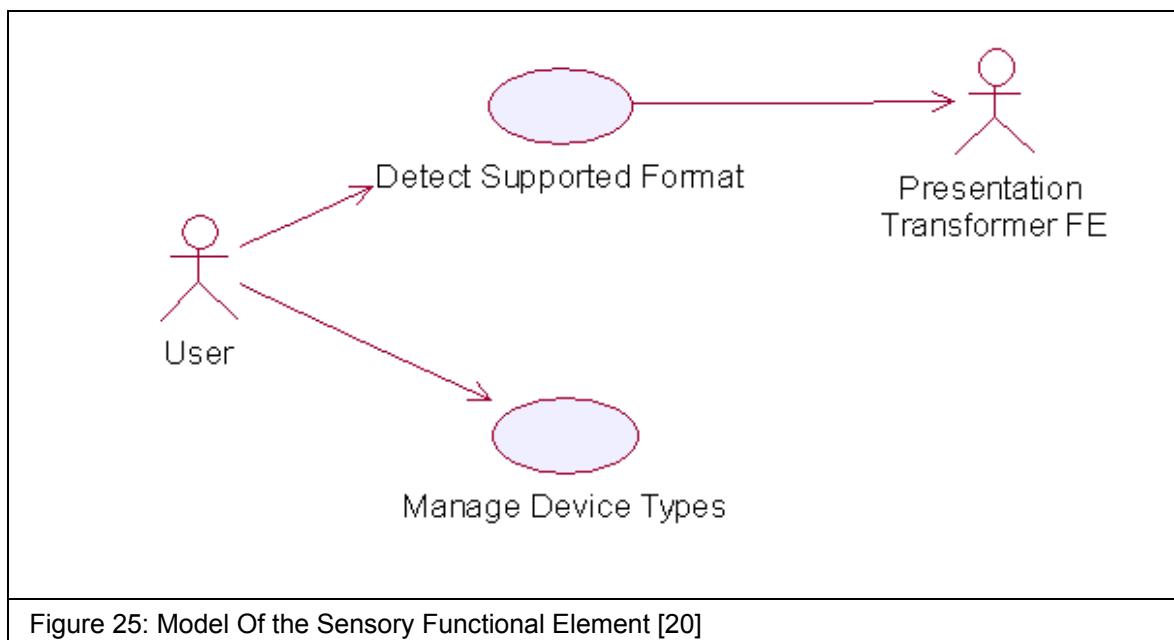
Interaction Dependency	
Presentation Transformer Functional Element	The Presentation Transformer Functional Element may be used to generate the appropriate output for the targeted devices.

## 5009 2.19.5 Related Technologies and Standards

5010 None.

5011

## 5012 2.19.6 Model



## 5013 2.19.7 Usage Scenarios

### 5014 2.19.7.1 Detect Supported Format

#### 5015 2.19.7.1.1 Description

5016 This use case allows the service user (user/other service) to make request and based on that  
 5017 request it detects service user's device capabilities.

5018     **2.19.7.1.2     Flow of Events**

5019     **2.19.7.1.2.1    Basic Flow**

5020     This use case starts when the service user wishes to use any service provided by the service  
5021     provider.

5022     1: The Functional Element receives the request from the service user.

5023     2: The Functional Element extracts MIME name and MIME type from the service user's HTTP  
5024     request (even from SOAP request).

5025     3: The Functional Element uses MIME name and MIME TYPE to check with the pre-registered  
5026     MIME type.

5027     4: The Functional Element sends device capabilities to service user and ends the use case.

5028     **2.19.7.1.2.2    Alternative Flows**

5029     1: Unsupported Device.

5030         1.1 If in the basic flow 2, the Functional Element is unable to detect the service user' device  
5031         capability, the Functional Element returns a error message and the use case ends.

5032     **2.19.7.1.3     Special Requirements**

5033     None

5034     **2.19.7.1.3.1    Supportability**

5035     The edge devices must be able to support the HTTP request.

5036     **2.19.7.1.4     Pre-Conditions**

5037     None.

5038     **2.19.7.1.5     Post-Conditions**

5039     None.

5040     **2.19.7.2        Manage Device Types**

5041     **2.19.7.2.1     Description**

5042     This use case allows the service user to maintain the device (MIME Type information). This  
5043     includes adding, changing and deleting device information from the Functional Element.

5044     **2.19.7.2.2     Flow of Events**

5045     **2.19.7.2.2.1    Basic Flow**

5046     This use case starts when the service user wishes to add or delete either device or service  
5047     information from the Functional Element.

5048     1: The Functional Element requests that the service user specify the function to perform (either  
5049     add, update or delete device or service).

5050 2: Once the service user provides the requested information, one of the sub-flows is executed.

5051 If the service user provides '**Register Device Types**', then sub-flow 2.1 is executed.

5052 If the service user provides '**Delete Device Types**', then sub-flow 2.2 is executed.

5053 2.1: Register Device Type.

5054 2.1.1: The Functional Element requests that the service user provide the device  
5055 information. This includes: MIME Name, MIME Description, Supported MIME type.

5056 2.1.2: Once the service user provides the requested information, the Functional Element  
5057 generates and assigns a unique MIME Id number to the device.

5058 2.2: Delete Device Type.

5059 2.2.1: The Functional Element requests that the service user provide the Device ID.

5060 2.2.2: The Functional Element retrieves the existing device information based on the  
5061 Device ID.

5062 2.2.3: The service user provides the delete device information and the Functional  
5063 Element deletes the device record from the Functional Element.

5064 3: The use case ends when the service user provides the requested information or decided to  
5065 end use case.

5066 **2.19.7.2.2.2 Alternative Flows**

5067 1: Invalid Device Information.

5068 1.1: If in the sub-flow 2.1.2, the requested information provided by the user is invalid, the  
5069 Functional Element returns an error message and the use case ends

5070 2: Device Not Found.

5071 2.1 If in the basic flows 2.2.2, the device information with the specified device is not found or  
5072 does not exist, the Functional Element returns an error message and the use case ends.

5073 **2.19.7.2.3 Special Requirements**

5074 **2.19.7.2.3.1 Supportability**

5075 Manage Device Types supports the most widespread MIME types used today.

5076 **2.19.7.2.4 Pre-Conditions**

5077 None.

5078 **2.19.7.2.5 Post-Conditions**

5079 If the use case was successful, the device information is added, updated or deleted from the  
5080 Functional Element. Otherwise, the Functional Element's state is unchanged.

## 2.20 Service Level Management Functional Element (new)

### 2.20.1 Motivation

The Service Level Management Functional Element enables the management of Service Level Agreements (SLAs), each of which represents a joint agreement between the service customer and provider based on a set of service offerings. The service offerings typically expressed as SLA templates, but still can be customized to cater to various services and customers. The Service Level Management Functional Element also manages the lifecycle of a SLA which could be broadly classified into: SLA creation; SLA deployment and provisioning; SLA enforcement and SLA termination.

This Functional Element fulfills the following requirements from the Functional Elements Requirements:

Primary Requirement

- MANAGEMENT-300.

### 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.

### 2.20.3 Key Features

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

1. The Functional Element MUST provide the ability to create Service Offering and associated service levels.
2. The Functional Element MUST provide the ability to manage defined Service Offerings, including the ability to retrieve, modify and delete.
3. The Functional Element MUST provide the ability to create of a SLA via customer subscription based on defined Service Offerings.
4. The Functional Element MUST provide the ability to generate billing & service level reports based on defined SLAs.
5. The Functional Element MUST provide the ability to notify subscribers of SLA termination.
6. The Functional Element MUST provide the ability to delete SLAs upon termination.

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

1. The Functional Element MAY provide the ability to customize SLAs. This includes the capability to:
  - 1.1. Alter service offerings parameters.
  - 1.2. Add and delete different service offerings into a SLA.

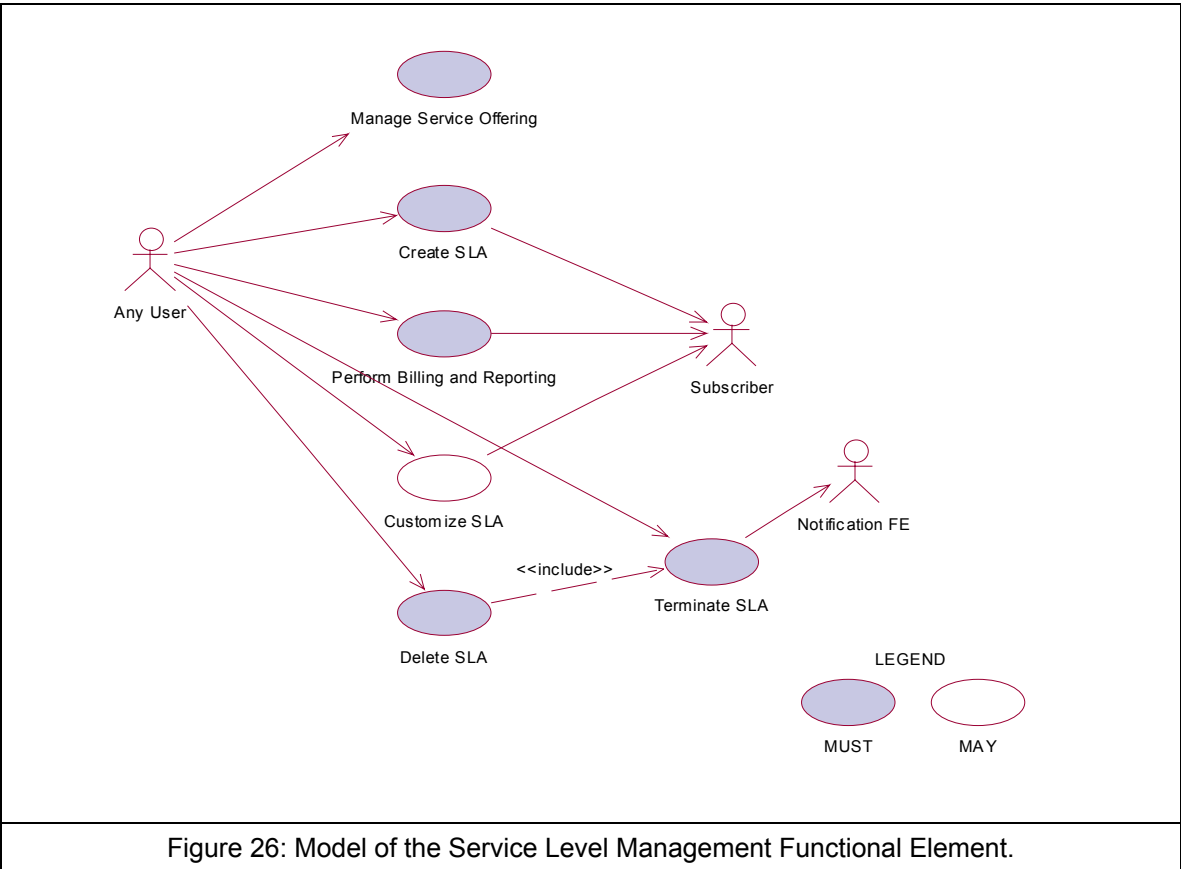
5119 **2.20.4 Interdependencies**

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.

5120  
5121 **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.

5122  
5123 **2.20.6 Model**



## 5126    **2.20.7        Usage Scenarios**

### 5127    **2.20.7.1        Manage Service Offering**

#### 5128    **2.20.7.1.1      Description**

5129    This use case allows any user to manage service offering, which enables any user to create,  
5130    retrieve, update and delete a service offering.

#### 5131    **2.20.7.1.2      Flow of Events**

##### 5132    **2.20.7.1.2.1    Basic Flow**

5133    This use case starts when any user wants to manage service offerings.

5134    1: The user sends Manage Service Offering request to the system together with the specified  
5135    operation.

5136    2: On receipt of the request from the user, the functional element will execute one of the sub-  
5137    flows. If the service user provides "**Create Service Offering**", the Create Service Offering sub-  
5138    flow (**2.1**) is executed. If the service user provides "**Update Service Offering**", the Update  
5139    Service Offering sub-flow (**2.2**) is activated. If the service user provides "**Retrieve Service**  
5140    **Offering**", the Retrieve Service Offering sub-flow (**2.3**) is activated. If the service user provides  
5141    "**Delete Service Offering**", the Delete Service Offering sub-flow (**2.4**) is executed.

5142

5143        2.1: Create Service Offering.

5144            2.1.1: The service user specifies details of a service offering.

5145            2.1.2: The system checks the existing service offering.

5146            *2.1.3: The system generates service offering information and adds to the system*  
5147            *and the use case ends.*

5148        2.2: Update Service Offering.

5149            2.2.1: The service user specifies the service offering to update.

5150            2.2.2: The system retrieves the existing service offering information.

5151            2.2.3: The service user provides the update service offering information.

5152            *2.2.4: The system updates the service offering with the updated information and*  
5153            *ends use case.*

5154        2.3: Retrieve Service Offering.

5155            2.3.1: The service user specifies the service offering to retrieve.

5156            2.3.2: The system retrieves the existing service offering information and ends the use  
5157            case.

5158        2.4: Delete Service Offering.

5159            2.4.1: The service user specifies the service offering to delete.

5160            2.4.2: The system retrieves the existing service offering information.

5161            *2.4.3: The system deletes the service offering from the system and the use case*  
5162            *ends.*

##### 5163    **2.20.7.1.2.2    Alternative Flows**

5164    1: Invalid Service Offering.



5165 1.1: If in the Basic Flow 2.1.1, system detects any invalid description, system returns  
5166 general error message and ends the use case.

5167 2: Service Offering Already Exists.

5168 2.1: If in the Basic Flow 2.1.2, the system checks the existing service offering and finds the  
5169 service offering already exists. The system returns an error and ends the use case.

5170 3: Service Offering Not Exist.

5171 3.1: If in the Basic Flow 2.2.2, 2.3.2, 2.4.2, the system checks the existing service  
5172 offering and finds the service offering doesn't exist. The system returns an error  
5173 and ends the use case.

5174 **2.20.7.1.3 Special Requirements**

5175 **2.20.7.1.4 Pre-Conditions**

5176 None.

5177 **2.20.7.1.5 Post-Conditions**

5178 None.

5179

5180 **2.20.7.2 Create SLA**

5181 **2.20.7.2.1 Description**

5182 This use case allows any user to create Service Level Agreement.

5183 **2.20.7.2.2 Flow of Events**

5184 **2.20.7.2.2.1 Basic Flow**

5185 This use case starts when any user wants to create SLA.

5186 1: The user sends a request to create SLA to the Functional Element which includes the  
5187 arrangement of the defined service offerings.

5188 2: The Functional Element will dispatch the SLA information to the subscribers.

5189 3: The subscribers accept the SLA arrangement and the use case ends.

5190 **2.20.7.2.3 Alternative Flows**

5191 1: Service Offering Not Available.

5192 1.1: If in the Basic Flow 1, Functional Element detects the service offering provided by the  
5193 user is not available, the Functional Element returns general error message and ends the use  
5194 case.

5195 2: Subscriber Not Available.

5196 2.1: If in the Basic Flow 2, the Functional Element checks that the subscriber is not available,  
5197 the Functional Element returns an error and ends the use case.

5198 3: Subscriber Don't Agree.

5199 3.1: If in the Basic Flow 3, the subscriber does not agree with the arrangement defined in  
5200 SLA, the Functional Element returns an error and ends the use case.

#### 5201 **2.20.7.2.4 Special Requirements**

5202 None.

#### 5203 **2.20.7.2.5 Pre-Conditions**

5204 None.

#### 5205 **2.20.7.2.6 Post-Conditions**

5206 If the use case is successful, a SLA is added into the Functional Element.

5207

### 5208 **2.20.7.3 Perform Billing and Reporting**

5209 This use case allows any user to do billing and reporting of the information related to SLA.

#### 5210 **2.20.7.3.1 Flow of Events**

##### 5211 **2.20.7.3.1.1 Basic Flow**

5212 This use case starts when any user wants to do SLA related billing and report.

5213 1: The user sends a request to conduct billing and reporting by providing information, which  
5214 enables to identify the SLA and its service offering and associated subscribers.

5215 2: On receipt of request of performing billing and reporting from the user, the Functional Element  
5216 retrieves the billing and report information according to the definition of SLA and internally  
5217 recorded information.

5218 3: The Functional Element passes the generated information to the subscribers.

5219 4: The Functional Element passes the response to the user and the use case ends.

##### 5220 **2.20.7.3.1.2 Alternative Flows**

5221 1: Information Not Enough.

5222 1.1: If in the Basic Flow 1, Functional Element detects the information provided by the user is  
5223 not enough to form identify the SLA and its associated service offerings and subscribers,  
5224 Functional Element returns general error message and ends the use case.

5225 2: No Data Available.

5226 2.1: If in the Basic Flow 2, the Functional Element retrieves the recorded information and  
5227 finds it is unavailable or incomplete, the Functional Element returns an error and ends the use  
5228 case.

5229 3: Subscriber Not Available.

5230 3.1: If in the Basic Flow 3, the subscriber is not available, the Functional Element returns an  
5231 error and ends the use case.

5232     **2.20.7.3.2     Special Requirements**

5233     None.

5234     **2.20.7.3.3     Pre-Conditions**

5235     None.

5236     **2.20.7.3.4     Post-Conditions**

5237     None.

5238

5239     **2.20.7.4        Customize SLA**

5240     **2.20.7.4.1     Description**

5241     This use case allows users to customize a SLA.

5242     **2.20.7.4.1.1   Basic Flow**

5243     This use case starts when any user wants to customize a SLA.

5244     1: The user sends request to customize a SLA by providing the information what will be  
5245     customized in a SLA. There are two ways to customize a SLA, to modify the parameters of  
5246     service offerings in a SLA and to add or delete service offerings in a SLA.

5247     2: On receipt of a customizing SLA request from the user, the Functional Element checks the  
5248     validity of the customized SLA.

5249     3: The Functional Element passes the customized SLA to the subscribers.

5250     4: The subscribers accept the customized SLA.

5251     5: The Functional Element passes the response from the service to the user and the use case  
5252     ends.

5253     **2.20.7.4.1.2   Alternative Flows**

5254     1: SLA Not Available.

5255         1.1: If in the Basic Flow 1, the SLA that the user wants to customize does not exist,  
5256         Functional Element returns general error message and ends the use case.

5257     2: Information Not Valid.

5258         2.1: If in the Basic Flow 2, Functional Element detects the information provided by the user is  
5259         not valid to form a SLA, Functional Element returns general error message and ends the use  
5260         case.

5261     3: Subscriber Not Available.

5262         3.1: If in the Basic Flow 3, the subscriber is not available, Functional Element returns general  
5263         error message and ends the use case.

5264     4: Subscriber Does Not Accept.

5265 4.1: If in the Basic Flow 4, the subscriber does not accept the customized SLA, Functional  
5266 Element returns general error message and ends the use case.

#### 5267 **2.20.7.4.2 Special Requirements**

5268 None.

#### 5269 **2.20.7.4.3 Pre-Conditions**

5270 None.

#### 5271 **2.20.7.4.4 Post-Conditions**

5272 If the use case is successful, a customized SLA is added into the functional element.

5273

#### 5274 **2.20.7.5 Terminate SLA**

5275 This use case enables the user to terminate a SLA.

#### 5276 **2.20.7.5.1 Flow of Events**

##### 5277 **2.20.7.5.1.1 Basic Flow**

5278 This use case starts when the user wants to terminate a SLA.

5279 1: The user sends a request to terminate a SLA to the Functional Element by providing related  
5280 information.

5281 2: On receipt of a terminating SLA request from the user, the Functional Element terminates the  
5282 operations related to the SLA.

5283 3: The Functional Element notifies the subscribers about the termination of the SLA through  
5284 Notification Functional Element.

5285 4: The Functional Element passes the response from the service to the user and the use case  
5286 ends.

##### 5287 **2.20.7.5.1.2 Alternative Flows**

5288 1: SLA Not Exist.

5289 1.1: If in the Basic Flow 2, Functional Element detects the SLA that the user wants to  
5290 terminate does not exist, Functional Element returns general error message and ends the use  
5291 case.

5292 2: Notification FE Not Available.

5293 2.1: If in Basic Flow 3, Functional Element detects the Notification Functional Element is not  
5294 available, Functional Element returns general error message and ends the use case.

#### 5295 **2.20.7.5.2 Special Requirements**

5296 None.

#### 5297 **2.20.7.5.3 Pre-Conditions**

5298 None.

5299     **2.20.7.5.4     Post-Conditions**

5300     If the use case is successful, the Functional Element stops all the operations related to the SLA.  
5301

5302     **2.20.7.6       Delete SLA**

5303     This use case enables the user to remove a SLA from the Functional Element.

5304     **2.20.7.6.1     Flow of Events**

5305     **2.20.7.6.1.1   Basic Flow**

5306     This use case starts when the user wants to delete a SLA from the Functional Element.

5307     1: The user sends a request to delete a SLA providing related information.

5308     2: On receipt of request of deleting SLA from the user, the Functional Element validates the  
5309     provided information and invokes the use case Terminate SLA.

5310     3: The Functional Element deletes the SLA.

5311     4: The Functional Element passes the response from the service to the user and the use case  
5312     ends.

5313     **2.20.7.6.1.2   Alternative Flows**

5314     1: SLA Does Not Exist.

5315         1.1: If in the Basic Flow 2, Functional Element detects the SLA that the user wants to delete  
5316         does not exist, Functional Element returns general error message and ends the use case.

5317     2: Terminate SLA Error.

5318         2.1: If in the Basic Flow 2, use case Terminate SLA returns error, Functional Element returns  
5319         general error message and ends the use case.

5320     **2.20.7.6.2     Special Requirements**

5321     None.

5322     **2.20.7.6.3     Pre-Conditions**

5323     None.

5324     **2.20.7.6.4     Post-Conditions**

5325     If the use case is successful, a SLA is deleted from the Functional Element.

## 2.21 Service Level Enforcement Functional Element (new)

### 2.21.1 Motivation

The Service Level Enforcement Functional Element enables monitoring the compliance of SLA and enforcing SLA through load management.

This Functional Element fulfills the following requirements from the Functional Elements Requirements:

- Primary Requirements
  - MANAGEMENT-301 and
  - MANAGEMENT-302.

### 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.

### 2.21.3 Key Features

Implementations of the Service Level Enforcement Functional Element are expected to provide the following key features:

1. The Functional Element MUST provide the ability to monitor SLA compliance based on measured data.
2. The Functional Element MUST provide the ability to detect any violation of SLA.
3. The Functional Element MUST provide the ability to enforce a SLA via through load management.

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

1. The Functional Element MAY provide the ability to manage load. This include the capability to:
  - 1.1. Control admission of service.
  - 1.2. Prioritize requests.

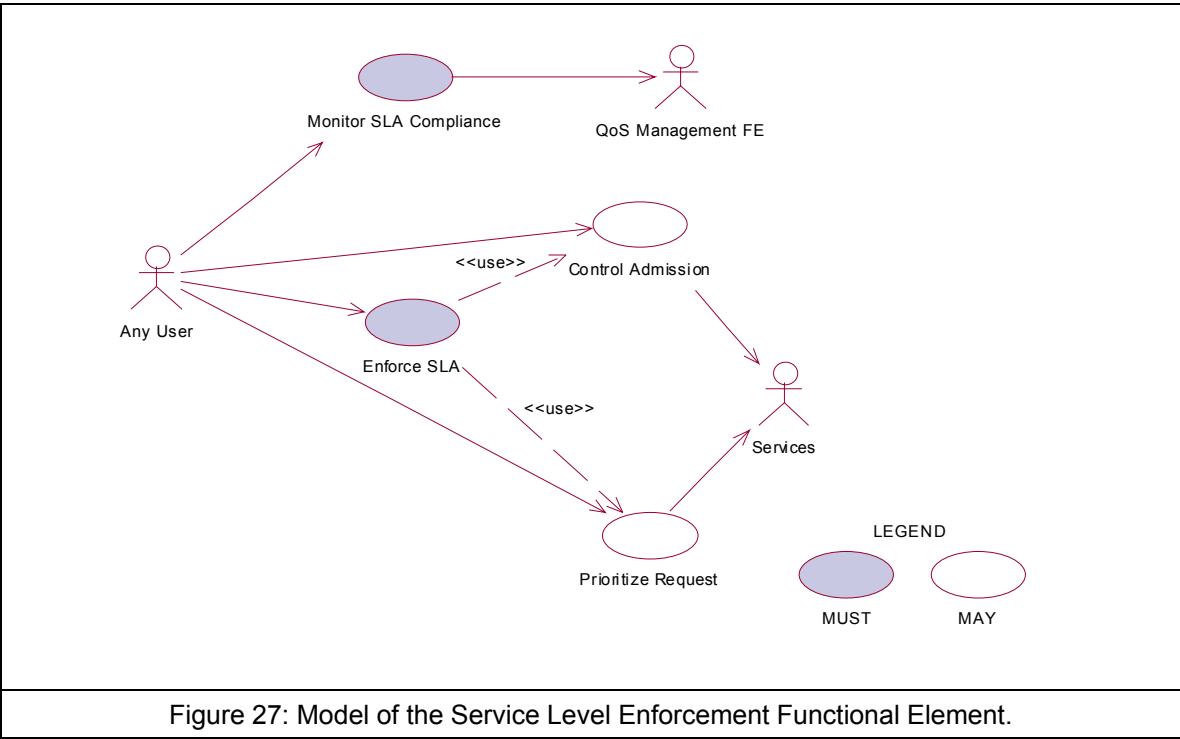
### 2.21.4 Interdependencies

Interaction Dependencies	
QoS Management	The Service Level Enforcement Functional Element may make use the metrics and metering results to monitor compliance of SLA.

5357      **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.

5358      **2.21.6      Model**



5360

5361      **2.21.7      Usage Scenarios**

5362      **2.21.7.1      Monitor SLA Compliance**

5363      **2.21.7.1.1      Description**

5364      This use case allows any user to monitor and check the SLA is compliant or not at the run time.

5365      **2.21.7.1.2      Flow of Events**

5366      **2.21.7.1.2.1      Basic Flow**

5367      This use case starts when any user wants to monitor the SLA compliance.

5368      1: The user sends Monitor SLA Compliance request to the Functional Element together with the  
5369      specified SLA information.

5370      2: On receipt of the request from the user, the Functional Element will retrieve the SLA  
5371      information.

5372

5373 3: The Functional Element extracts the measured data through QoS Management Functional  
5374 Element.

5375 4: The Functional Element checks the compliance of SLA.

5376 5: The Functional Element returns response to the user and the use case ends.

5377 **2.21.7.1.2.2 Alternative Flows**

5378 1: SLA Not Exist.

5379 1.1: If in the Basic Flow 2, the Functional Element detects that the SLA to monitor does not  
5380 exists, system returns general error message and ends the use case.

5381 2: Measured Data Not Available.

5382 2.1: If in the Basic Flow 3, the Functional Element retrieves measured data through QoS  
5383 Management Functional Element and the latter is not ready, the Functional Element returns  
5384 an error and ends the use case.

5385 3: SLA Not Compliant.

5386 3.1: If in the Basic Flow 4, the Functional Element checks the measured data against SLA  
5387 and the violation exists, the Functional Element returns an error and ends the use case.

5388 **2.21.7.1.3 Special Requirements**

5389 **2.21.7.1.4 Pre-Conditions**

5390 None

5391 **2.21.7.1.5 Post-Conditions**

5392 None

5393

5394 **2.21.7.2 Control Admission**

5395 **2.21.7.2.1 Description**

5396 As a means of manage load to enforce SLA, the use case allows any user to control admission  
5397 toward services.

5398 **2.21.7.2.2 Flow of Events**

5399 **2.21.7.2.2.1 Basic Flow**

5400 This use case starts when any user wants to control admission toward services.

5401 1: The user sends request to control admission to certain services to the Functional Element  
5402 which includes the option of admission and the targeted services.

5403 2: The Functional Element will manage the control of admission to the services at run time.

5404 3: The Functional Element returns response to the user and the use case ends.



5405     **2.21.7.2.3     Alternative Flows**

5406     1: Service Not Available.

5407         1.1: If in the Basic Flow 1, Functional Element detects the targeted service provided by the  
5408         user is not available, Functional Element returns general error message and ends the use  
5409         case.

5410     2: Control Admission Failed.

5411         2.1: If in the Basic Flow 2, the Functional Element fails to control admission to the services at  
5412         run time, Functional Element returns an error and ends the use case.

5413     **2.21.7.2.4     Special Requirements**

5414     None.

5415     **2.21.7.2.5     Pre-Conditions**

5416     The services are manageable to the user.

5417     **2.21.7.2.6     Post-Conditions**

5418     If the use case is successful, the load of the monitored services is changed thus the SLA is  
5419     enforced through load management.

5420

5421     **2.21.7.3         Prioritize Request**

5422     As a means of load management to enable SLA enforcement, the use case allows any user to  
5423     prioritize request to the targeted services according to the requirements of SLA.

5424     **2.21.7.3.1     Flow of Events**

5425     **2.21.7.3.1.1    Basic Flow**

5426     This use case starts when any user wants to prioritize various requests to targeted services.

5427     1: The user sends request to prioritize request to the Functional Element, which include  
5428     information of the targeted services, the priority of the request and so on.

5429     2: On receipt of the request from the user, the Functional Element controls the processing of the  
5430     request according to the priority given at the run time.

5431     3: The Functional Element passes the response to the user and the use case ends.

5432     **2.21.7.3.1.2    Alternative Flows**

5433     1: Services Not Exist.

5434         1.1: If in the Basic Flow 1, Functional Element detects the targeted service provided by the  
5435         user does not exist, Functional Element returns general error message and ends the use  
5436         case.

5437     2: Prioritize Request Fails.

5438 2.1: If in the Basic Flow 2, the Functional Element fails to control the requests of the services  
5439 according to the priority given the user, the Functional Element returns an error and ends the  
5440 use case.

#### 5441 **2.21.7.3.2 Special Requirements**

5442 None.

#### 5443 **2.21.7.3.3 Pre-Conditions**

5444 The services are manageable to the user.

#### 5445 **2.21.7.3.4 Post-Conditions**

5446 If the use case is successful, the load of the monitored services is changed thus the SLA is  
5447 enforced through load management.

5448

#### 5449 **2.21.7.4 Enforce SLA**

##### 5450 **2.21.7.4.1 Description**

5451 This use case allows users to enforce a SLA in a run time environment.

##### 5452 **2.21.7.4.1.1 Basic Flow**

5453 This use case starts when any user wants to enforce a SLA in the run time environment.

5454 1: The user sends a request to enforce a SLA to the Functional Element by providing the SLA  
5455 and its associated services and the option of the means of enforcement through load  
5456 management.

5457 2: On receipt of the request from the user, the Functional Element checks the SLA and decides  
5458 the means of enforcement, i.e. by taking advantage of load management.

5459 3: The Functional Element dispatches its request of load management and invokes use case  
5460 Control Admission or use case Prioritize Request.

5461 4: The Functional Element returns the response to the user and the use case ends.

##### 5462 **2.21.7.4.1.2 Alternative Flows**

5463 1: SLA Not Available.

5464 1.1: If in the Basic Flow 1, the SLA that the user wants to enforce does not exist, Functional  
5465 Element returns general error message and ends the use case.

5466 2: Services Not Exist.

5467 2.1: If in the Basic Flow 1, Functional Element detects the services that the user wants to  
5468 enforce SLA do not exist, Functional Element returns general error message and ends the  
5469 use case.

5470 3: Control Admission Not Working.

5471 3.1: If in the Basic Flow 3, Functional Element fails to invoke use case control admission,  
5472 Functional Element returns general error message and ends the use case.

5473 4: Prioritize Request Not Working.

5474 4.1: If in the Basic Flow 3, Functional Element fails to invoke use case Prioritize Request,  
5475 Functional Element returns general error message and ends the use case.

5476 **2.21.7.4.2 Special Requirements**

5477 None.

5478 **2.21.7.4.3 Pre-Conditions**

5479 The services targeted are manageable.

5480 **2.21.7.4.4 Post-Conditions**

5481 None.

5482

## 2.22 Service Management Functional Element

### 2.22.1 Motivation

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

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

Primary Requirements

- MANAGEMENT-090, and
- MANAGEMENT-093 to MANAGEMENT-096.

Secondary Requirements

- None

### 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

### 2.22.3 Key Features

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

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)*

25. The Functional Element MUST provide the capability to discover services that are under the Management Domain.

26. The Functional Element MUST provide the capability to configure Performance Parameters 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.*

5510 27. The Functional Element MUST provide a means to log Performance Parameters.

5511

5512 In addition, the following key feature could be provided to enhance the Functional Element  
5513 further:

5514 1. The Functional Element MAY provide the capability to configure additional attributes that is  
5515 tagged along with a particular Web Service.

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

5516 2. The Functional Element MAY provide verification services to block unauthorized Web  
5517 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.*

#### 5518 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

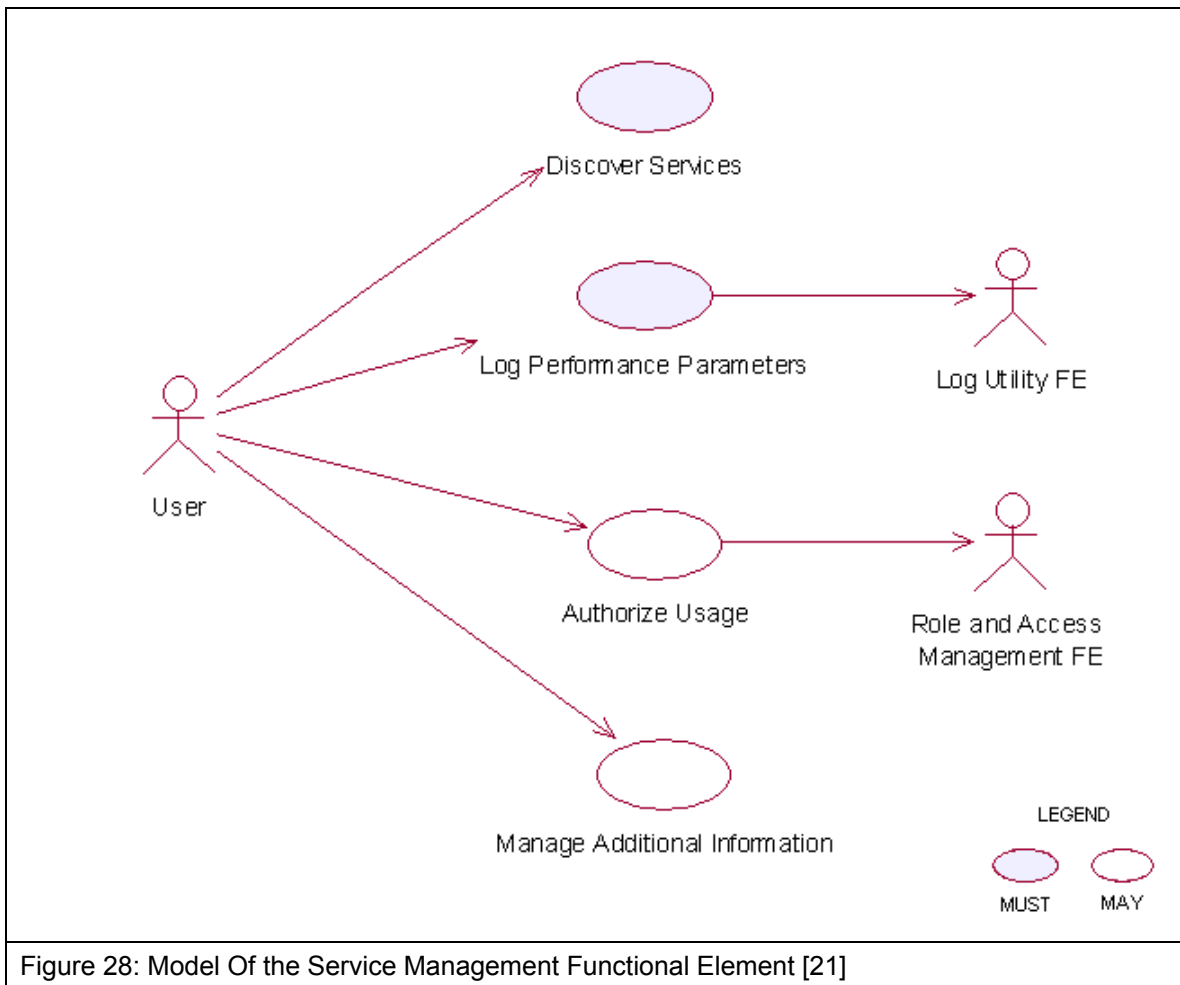
5519

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.

#### 5520 2.22.5 Related Technologies and Standards

5521 None

5522 **2.22.6 Model**



5523 **2.22.7 Usage Scenarios**

5524 **2.22.7.1 Discover Services**

5525 **2.22.7.1.1 Description**

5526 This use case describes the scenario surrounding the automatic discovery of services hosted in  
5527 the Management Domain.

5528 **2.22.7.1.2 Flow of Events**

5529 **2.22.7.1.2.1 Basic Flow**

5530 The use case begins when the user wants to retrieve a list of services URLs from the  
5531 Management Domain.

5532 1: The user sends a request to retrieve the list of services URLs from the Management Domain.

5533 2: The Functional Element reads from a configuration file to so as to determine the exact  
5534 boundaries of the Management Domain.

5535 3: The Functional Element retrieves from each of the servers as stated in the configuration file a  
5536 list of service URLs that it is hosting

5537 4: The Functional Element returns the list of service URLs back to the user and the use case  
5538 ends.

#### 5539 **2.22.7.1.2.2 Alternative Flows**

5540 1: Configuration File Does Not Exist

5541 1.1: In basic flow 2, the Functional Element fails to read boundaries from the configuration  
5542 file. The Functional Element in turn return an error message and the use case end.

5543 2: Fail To Communicate With the Server

5544 2.1: In basic flow 3, the Functional Element fails to communicate with the servers hosting the  
5545 services. The Functional Element in turn return an error message and the use case end.

#### 5546 **2.22.7.1.3 Special Requirements**

5547 The protocol of communicating with a server hosting the services is not standardized. Each  
5548 server may offer different mechanism for retrieving the list of services hosted and as such, the  
5549 extensibility this approach is severely limited.

#### 5550 **2.22.7.1.4 Pre-Conditions**

5551 None.

#### 5552 **2.22.7.1.5 Post-Conditions**

5553 None

5554

### 5555 **2.22.7.2 Log Performance Parameters**

#### 5556 **2.22.7.2.1 Description**

5557 This use case allows the user to log the performance parameters of all the Web Services that is  
5558 being hosted by an application that contains the Service Management Functional Element.

#### 5559 **2.22.7.2.2 Flow of Events**

##### 5560 **2.22.7.2.2.1 Basic Flow**

5561 The use case begins when the user wants to log the performance parameters of all the Web  
5562 Services that is being hosted by an application that contains the Service Management Functional  
5563 Element.

5564 1: The user sends a request to log the performance parameters of all the Web Services hosted.

5565 2: The Functional Element reads from a configuration file the performance parameter to be  
5566 logged.

5567 3: The Functional Element extracts the performance parameters for the incoming message and  
5568 stores them into the data store

5569 4: The Functional Element next extracts the performance parameters for the outgoing message  
5570 and stores them into the data store

5571 5: The Functional Element stores the necessary information into the data store.

5572 **2.22.7.2.2.2 Alternative Flows**

5573 1: No Performance Parameter Found.

5574 1.1: In basic flow 2, the Functional Element discovers that the performance parameter to be  
5575 logged is not configured. The Functional Element returns an error message and the use case  
5576 ends.

5577 2: Data Store Not Available.

5578 2.1: In basic flow 5, the Functional Element detects that the data store is not available. The  
5579 Functional Element returns an error message and the use case ends.

5580 **2.22.7.2.3 Special Requirements**

5581 None.

5582 **2.22.7.2.4 Pre-Conditions**

5583 None.

5584 **2.22.7.2.5 Post-Conditions**

5585 None.

5586

5587 **2.22.7.3 Authorize Usage**

5588 **2.22.7.3.1 Description**

5589 This use case describes the authentication process for invoking a Web Service that is being  
5590 hosted by an application that contains the Service Management Functional Element.

5591 **2.22.7.3.2 Flow of Events**

5592 **2.22.7.3.2.1 Basic Flow**

5593 The use case starts when a user accesses a service.

5594 1: The user sends a request to invoke a particular Web Service.

5595 2: The Functional Element extracts the following information from the incoming message

5596 2.1: The username attribute that resides in the header of the incoming message

5597 3: The Functional Element extracts the access privilege associated with the service from the data  
5598 store

5599 4: The Functional Element uses the Role and Access Management Functional Element to retrieve  
5600 the role of the user.

5601 5: The Functional Element looks up the data store to determine if the user is authorized to access  
5602 the service



5603 6: The Functional Element allows the request to be process and the use case ends.

5604 **2.22.7.3.2.2 Alternative Flow**

5605 1: Username header not found.

5606 1.1: In basic flow 2, the username attribute is not found in the header.

5607 1.2: The Functional Element denies access to the requested Web Service and returns an  
5608 error message.

5609 2: Web Service access privilege not set.

5610 2.1: In basic flow 3, the Functional Element could not find the access privilege for the Web  
5611 Service.

5612 2.2: The Functional Element denies access to the requested Web Service and returns an  
5613 error message.

5614 3: Role and Access Management Functional Element not available

5615 3.1: In basic flow 4, the Functional Element could not find the Role and Access Management  
5616 Functional Element.

5617 3.2: The Functional Element denies access to the requested Web Service and returns an  
5618 error message.

5619 4: User not authorize

5620 4.1: In basic flow 5, the Functional Element looks up the data source and determines that the  
5621 user does not have the required privilege to access the service.

5622 4.2: The Functional Element denies access to the requested Web Service and returns an  
5623 error message.

5624 **2.22.7.3.3 Special Requirements**

5625 None.

5626 **2.22.7.3.4 Pre-Conditions**

5627 None.

5628 **2.22.7.3.5 Post-Conditions**

5629 None.

5630

5631 **2.22.7.4 Manage Additional Information**

5632 **2.22.7.4.1 Description**

5633 This use case helps to maintain the following attributes of a Web Service that is useful in  
5634 determining if a particular user has the privilege to invoke it.

5635 Service Name. This is the name of the service to monitor

5636 Access level. This refers to the access level of the Web Services hosted

5637 Role Names. If a user's role matches any of the roles contained here, then he/she has the  
5638 privilege to access the Web Service.

#### 5639 **2.22.7.4.2 Flow of Events**

##### 5640 **2.22.7.4.2.1 Basic Flow**

5641 This use case starts when user wants to manage services.

5642 1: The user specifies the additional information that he wants to create/update/delete/retrieve.

5643 2: Once the user provides the requested information, one of the sub-flows is executed.

5644 If the user provides '**Create Service Parameter**', then sub-flow 2.1 is executed.

5645 If the user provides '**Update Service Parameter**', then sub-flow 2.2 is executed.

5646 If the user provides '**Delete Service Parameter**', then sub-flow 2.3 is executed.

5647 If the user provides '**Retrieve Service Parameter**', then sub-flow 2.4 is executed.

5648 2.1: Create Service Parameter.

5649 2.1.1: The user specifies the service to create with the appropriate additional information.

5650 2.1.2: The Functional Element connects to the data store.

5651 2.1.3: The Functional Element saves the new service in the data store and the use case  
5652 ends.

5653 2.2: Update Service Parameter.

5654 2.2.1: The user specifies the service to update with the appropriate additional information.

5655 2.2.2: The Functional Element connects to the data store.

5656 2.2.3: The Functional Element updates the service in the data store and the use case  
5657 ends.

5658 2.3: Delete Service Parameter.

5659 2.3.1: The user specifies the service to delete.

5660 2.3.2: The Functional Element connects to the data store.

5661 2.3.3: The Functional Element deletes the service in the data store and the use case  
5662 ends.

5663 2.4: Retrieve Service Parameter.

5664 2.4.1: The user specifies the service to retrieve.

5665 2.4.2: The Functional Element connects to the data store.

5666 2.4.3: The Functional Element retrieves the service from the data store and the use case  
5667 ends.

##### 5668 **2.22.7.4.2.2 Alternative Flows**

5669 1: Data Store Not Available.

5670 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  
5671 is returned and the use case ends.

5672 **2.22.7.4.3 Special Requirements**

5673 None.

5674 **2.22.7.4.4 Pre-Conditions**

5675 None.

5676 **2.22.7.4.5 Post-Conditions**

5677 None.

## 2.23 Service Registry Functional Element

### 2.23.1 Motivation

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

Simplify management of information in a XML registry server like UDDI and ebXML, and

Simplify information publish and query from a XML registry server like UDDI and ebXML.

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

Primary Requirements

- PROCESS-031 to PROCESS-032,
- PROCESS-035, and
- MANAGEMENT-097 to MANAGEMENT-100

Secondary Requirements

- PROCESS-014.

### 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

### 2.23.3 Key Features

Implementations of the Service Registry Functional Element are expected to provide the following key features:

1. The Functional Element MUST provide the capability to facilitate the management of the following information in a UDDI or an ebXML compliant registry server.
  - 1.1. Organisation
  - 1.2. Classification / Taxonomy
  - 1.3. Web Service
  - 1.4. tModel
  - 1.5. Service BindingThe management of this information includes registering, updating, deleting and searching.
2. As part of Key Feature (1), the Functional Element MUST provide the ability to perform the operations specified across multiple registry servers.
3. The Functional Element MUST provide a mechanism to enable single step publishing of services into registry servers

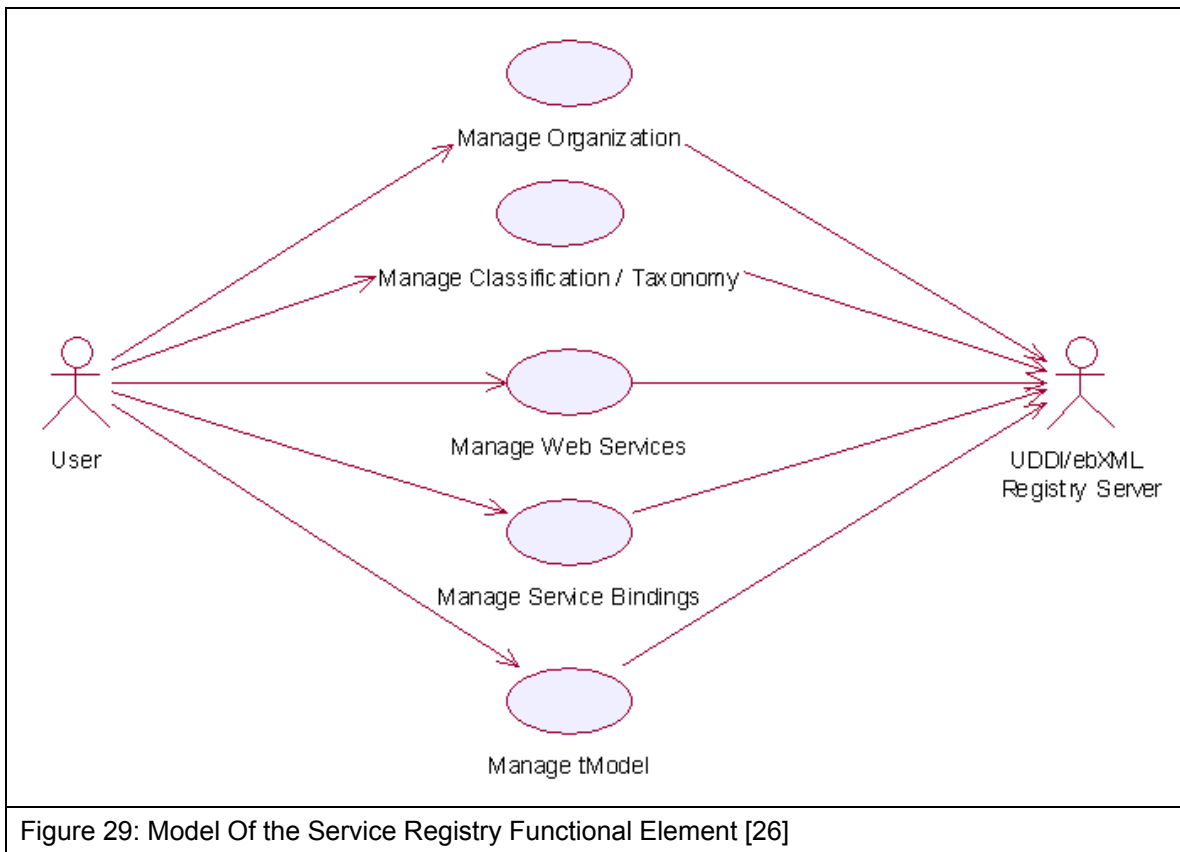
### 2.23.4 Interdependencies

None

### 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.

5718 **2.23.6 Model**



5719 **2.23.7 Usage Scenario**

5720 **2.23.7.1 Manage Classification / Taxonomy**

5721 **2.23.7.1.1 Description**

5722 This use case allows any users to create, remove and view classification/taxonomy in the  
 5723 registry.

5724 **2.23.7.1.2 Flow of Events**

5725 **2.23.7.1.2.1 Basic Flow**

5726 This use case starts when the users of registry server wishes to create, remove or view the  
 5727 classification/taxonomy in the registry server.

5728

5729 1: User initiates a request type to the Functional Element stating whether to create, remove or  
 5730 view classification/taxonomy.

5731 2: The Functional Element checks whether the registry server exists.

5732 3: The Functional Element checks the request. Based on the type of request, one of the sub-  
 5733 flows is executed.

5734 If the request is to '**Create Classification/Taxonomy**', then sub-flow 3.1 is executed.

5735 If the request is to '**View Classification/Taxonomy**', then sub-flow 3.2 is executed.

5736 If the request is to '**Remove Classification/Taxonomy**', then sub-flow 3.3 is executed.

5737 3.1: Create Classification/Taxonomy.

5738 3.1.1: Other Functional Element provides username, password and registry server URL

5739 to the Functional Element for authentication.

5740 3.1.2: The Functional Element checks for the user validity in the identified registry server.

5741 3.1.3: Other Functional Element provides classification/taxonomy information to be

5742 created in the registry server.

5743 3.1.4: The Functional Element checks for the duplicate classification/taxonomy name.

5744 3.1.5: The Functional Element creates the classification/taxonomy information in the

5745 private (default) or the public UDDI registry server according to the URL provided by

5746 other Functional Element, if it does not exist.

5747 3.2: View Classification/Taxonomy.

5748 3.2.1: The Functional Element retrieves all the classification/taxonomy from the identified

5749 registry server, which may be private (default) or public.

5750 3.2.2: The Functional Element returns the classification/taxonomy information from the

5751 identified registry server to other Functional Element.

5752 3.3: Remove Classification/Taxonomy.

5753 3.3.1: Other Functional Element provides username, password and registry server URL

5754 to the Functional Element for authentication.

5755 3.3.2: The Functional Element checks for the user validity in the identified registry server.

5756 3.3.3: Other Functional Element provides classification/taxonomy key (i.e. UUID) to be

5757 removed from the identified registry server.

5758 3.3.4: The Functional Element removes the classification/taxonomy information from the

5759 private (default) or the public UDDI registry server according to the URL provided by the

5760 user.

5761 4: The Functional Element returns the status of the operation and the use case ends.

## 5762 **2.23.7.1.2.2 Alternative Flows**

5763 1: Registry Server Down.

5764 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element

5765 returns an error message and the use case ends.

5766 2: Invalid Username And Password.

5767 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional

5768 Element returns an error message and the use case ends.

5769 3: Classification/Taxonomy Key Not Found.

5770 3.1: In the basic flow 3.3.3, if the classification/taxonomy key cannot be found in the

5771 specified registry server, the Functional Element returns an error message and the use

5772 case ends.

5773 4: Duplicate Classification/Taxonomy.  
5774 4.1: In the basic flow 3.1.4, If the same classification/taxonomy name has been defined in  
5775 the registry server, the Functional Element returns an error message and the use case  
5776 ends.

5777 **2.23.7.1.3 Special Requirements**  
5778 None

5779 **2.23.7.1.4 Pre-Conditions**  
5780 In order to manage the classification/taxonomy in the registry server, users must be registered  
5781 with the registry server. Username and password will be given when a user registers with a  
5782 registry server.

5783 **2.23.7.1.5 Post-Conditions**  
5784 None.

5785 **2.23.7.2 Manage Web Services**

5786 **2.23.7.2.1 Description**  
5787 This use case allows any users to register, remove and view Web Services in the private (default)  
5788 as well as the public UDDI Registry Server.

5789 **2.23.7.2.2 Flow of Events**

5790 **2.23.7.2.2.1 Basic Flow**  
5791 This use case starts when the users of registry server wishes to create, remove and view Web  
5792 Services.

5793 1: User initiates a request type to the Functional Element stating whether to create, remove or  
5794 view Web Services in the identified private or public registry server.

5795 2: The Functional Element checks whether the registry server exists.

5796 3: The Functional Element checks the request. Based on the type of request, one of the sub-  
5797 flows is executed.

5798 If the request is to '**Create Web Service**', then sub-flow 3.1 is executed.

5799 If the request is to '**View Web Services**', then sub-flow 3.2 is executed.

5800 If the request is to '**Remove Web Service**', then sub-flow 3.3 is executed.

5801 3.1: Create Web Service.

5802 3.1.1: User provides username, password and registry server URL to the Functional  
5803 Element for authentication.

5804 3.1.2: The Functional Element checks for the user validity in the identified registry server.

5805 3.1.3: Other Functional Element provides Web Service information to be created in the  
5806 registry server.



5807 3.1.4: The Functional Element creates the Web Service information in the private  
5808 (default) or the public UDDI registry server according to the URL provided by other  
5809 Functional Element.

5810 3.2: View Web Services.

5811 3.2.1: The Functional Element retrieves all the Web Services from the identified registry  
5812 server for specific stated conditions like service name search, business name search,  
5813 etc.

5814 3.2.2: The Functional Element displays the Web Services information search results from  
5815 the identified registry server to other Functional Element.

5816 3.3: Remove Web Service

5817 3.3.1 User provides username, password and registry server URL to the Functional  
5818 Element for authentication.

5819 3.3.2: The Functional Element checks for the user validity in the identified registry server.

5820 3.3.3: Other Functional Element provides Web Service key (i.e. UUID) to be removed  
5821 from the identified registry server.

5822 3.3.4: The Functional Element removes the Web Service information from the private  
5823 (default) or the public UDDI registry server according to the URL provided by other  
5824 Functional Element.

5825 4: The Functional Element returns the results of the operation and the use case ends.

#### 5826 **2.23.7.2.2.2 Alternative Flows**

5827 1: Registry Server Down.

5828 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element  
5829 returns an error message and the use case ends.

5830 2: Invalid Username And Password.

5831 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional  
5832 Element returns an error message and the use case ends.

5833 3: Web Service Key Not Found.

5834 3.1: In the basic flow 3.3.3, if the Web Service key cannot be found in the specified registry  
5835 server, the Functional Element returns an error message and the use case ends.

#### 5836 **2.23.7.2.3 Special Requirements**

#### 5837 **2.23.7.2.4 Pre-Conditions**

5838 In order to manage Web Services in the registry server, the users must be registered with the  
5839 registry server. Username and password will be given when a user registers with a registry  
5840 server.

#### 5841 **2.23.7.2.5 Post-Conditions**

5842 None.

### 5843    **2.23.7.3      Manage Organization**

#### 5844    **2.23.7.3.1    Description**

5845    This use case allows any users to create, remove and view organization in the registry.

#### 5846    **2.23.7.3.2    Flow of Events**

##### 5847    **2.23.7.3.2.1   Basic Flow**

5848    This use case starts when the users of registry server wishes to create, remove or view  
5849    Organization.

5850    1: User initiates a request type to the Functional Element stating whether to create, remove or  
5851    view Organization.

5852    2: The Functional Element checks whether the registry server exists.

5853    3: The Functional Element checks the request. Based on the type of request, one of the sub-  
5854    flows is executed.

5855    If the request is to '**Create Organization**', then sub-flow 3.1 is executed.

5856    If the request is to '**View Organizations**', then sub-flow 3.2 is executed.

5857    If the request is to '**Remove Organization**', then sub-flow 3.3 is executed.

5858        3.1: Create Organization.

5859            3.1.1: Other Functional Element provides username, password and registry server URL  
5860            to the Functional Element for authentication.

5861            3.1.2: The Functional Element checks for the user validity in the identified registry server.

5862            3.1.3: Other Functional Element provides organization information to be created in the  
5863            registry server.

5864            3.1.4: The Functional Element checks for the duplicate organization name.

5865            3.1.5: The Functional Element creates the organization information in the private (default)  
5866            or the public UDDI registry server according to the URL provided by other Functional  
5867            Element, if it does not exist.

5868        3.2: View Organizations.

5869            3.2.1: The Functional Element retrieves all the organizations from the identified registry  
5870            server for specific stated conditions like organization name, key, etc.

5871            3.2.2: The Functional Element returns the organization information from the identified  
5872            registry server to other Functional Element.

5873        3.3: Remove Organization.

5874            3.3.1: Other Functional Element provides username, password and registry server URL  
5875            to the Functional Element for authentication.

5876            3.3.2: The Functional Element checks for the user validity in the identified registry server.

5877            3.3.3: Other Functional Element provides Organization key (i.e. UUID) to be removed  
5878            from the identified registry server.

5879            3.3.4: The Functional Element removes the Organization information from the private  
5880            (default) or the public UDDI registry server according to the URL provided by the user.

5881    4: The Functional Element returns the status of the operation and the use case ends.

5882    **2.23.7.3.2.2    Alternative Flows**

5883    1: Registry Server Down.

5884            1.1: In the basic flow 2, if the identified registry server is down, the Functional Element  
5885            returns an error message and the use case ends.

5886    2: Invalid Username And Password.

5887            2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional  
5888            Element returns an error message and the use case ends.

5889    3: Organization Key Not Found.

5890            3.1: In the basic flow 3.3.3, if the Organization key cannot be found in the specified registry  
5891            server, the Functional Element returns an error message and the use case ends.

5892    4: Duplicate Organization.

5893            4.1: In the basic flow 3.1.4, If the same Organization name has been defined in the registry  
5894            server the Functional Element returns an error message and the use case ends.

5895    **2.23.7.3.3    Special Requirements**

5896    None

5897    **2.23.7.3.4    Pre-Conditions**

5898    In order to manage Organization in the registry server, users must be registered with the registry  
5899    server. Username and password will be given when a user registers with a registry server.

5900    **2.23.7.3.5    Post-Conditions**

5901    None.

5902    **2.23.7.4    Manage Service Binding**

5903    **2.23.7.4.1    Description**

5904    This use case allows any users to register, remove and view Service Binding in the private  
5905    (default) as well as the public UDDI Registry Server.

5906    **2.23.7.4.2    Flow of Events**

5907    **2.23.7.4.2.1    Basic Flow**

5908    This use case starts when the users of registry server wishes to create, remove and view Service  
5909    Binding.

5910    1: User initiates a request type to the Functional Element stating whether to create, remove or  
5911    view Service Binding in the identified private or public registry server.

5912    2: The Functional Element checks whether the registry server exists.

5913    3: The Functional Element checks the request. Based on the type of request, one of the sub-  
5914    flows is executed.

5915 If the request is to '**Create Service Binding**', then sub-flow 3.1 is executed.

5916 If the request is to '**View Service Bindings**', then sub-flow 3.2 is executed.

5917 If the request is to '**Remove Service Binding**', then sub-flow 3.3 is executed.

5918 3.1: Create Service Binding.

5919 3.1.1: User provides username, password and registry server URL to the Functional  
5920 Element for authentication.

5921 3.1.2: The Functional Element checks for the user validity in the identified registry server.

5922 3.1.3: Other Functional Element provides Service Binding information to be created in the  
5923 registry server.

5924 3.1.4: The Functional Element creates the Service Binding information in the private  
5925 (default) or the public UDDI registry server according to the URL provided by other  
5926 Functional Element.

5927 3.2: View Service Bindings.

5928 3.2.1: The Functional Element retrieves all the Service Bindings from the identified  
5929 registry server for specific stated conditions like service binding key search, etc.

5930 3.2.2: The Functional Element displays the Service Bindings information search results  
5931 from the identified registry server to other Functional Element.

5932 3.3: Remove Service Binding

5933 3.3.1 User provides username, password and registry server URL to the Functional  
5934 Element for authentication.

5935 3.3.2: The Functional Element checks for the user validity in the identified registry server.

5936 3.3.3: Other Functional Element provides Service Binding key (i.e. UUID) to be removed  
5937 from the identified registry server.

5938 3.3.4: The Functional Element removes the Service Binding information from the private  
5939 (default) or the public UDDI registry server according to the URL provided by other  
5940 Functional Element.

5941 4: The Functional Element returns the results of the operation and the use case ends.

#### 5942 **2.23.7.4.2.2 Alternative Flows**

5943 1: Registry Server Down.

5944 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element returns  
5945 an error message and the use case ends.

5946 2: Invalid Username And Password.

5947 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional  
5948 Element returns an error message and the use case ends.

5949 3: Service Binding Key Not Found.

5950 3.1: In the basic flow 3.3.3, if the Service Binding key cannot be found in the specified registry  
5951 server, the Functional Element returns an error message and the use case ends.

5952     **2.23.7.4.3     Special Requirements**

5953     **2.23.7.4.4     Pre-Conditions**

5954     In order to manage Service Binding in the registry server, the users must be registered with the  
5955     registry server. Username and password will be given when a user registers with a registry  
5956     server.

5957     **2.23.7.4.5     Post-Conditions**

5958     None.

5959     **2.23.7.5        Manage tModel**

5960     **2.23.7.5.1     Description**

5961     This use case allows any users to register, remove and view tModel in the private (default) as  
5962     well as the public UDDI Registry Server.

5963     **2.23.7.5.2     Flow of Events**

5964     **2.23.7.5.2.1    Basic Flow**

5965     This use case starts when the users of registry server wishes to create, remove and view tModel.

5966     1: User initiates a request type to the Functional Element stating whether to create, remove or  
5967     view tModel in the identified private or public registry server.

5968     2: The Functional Element checks whether the registry server exists.

5969     3: The Functional Element checks the request. Based on the type of request, one of the sub-  
5970     flows is executed.

5971     If the request is to '**Create tModel**', then sub-flow 3.1 is executed.

5972     If the request is to '**View tModels**', then sub-flow 3.2 is executed.

5973     If the request is to '**Remove tModel**', then sub-flow 3.3 is executed.

5974         3.1: Create tModel.

5975             3.1.1: User provides username, password and registry server URL to the Functional  
5976             Element for authentication.

5977             3.1.2: The Functional Element checks for the user validity in the identified registry server.

5978             3.1.3: Other Functional Element provides tModel information to be created in the registry  
5979             server.

5980             3.1.4: The Functional Element creates the tModel information in the private (default) or  
5981             the public UDDI registry server according to the URL provided by other Functional  
5982             Element.

5983         3.2: View tModels.

5984             3.2.1: The Functional Element retrieves all the tModels from the identified registry server  
5985             for specific stated conditions like tModel name search, tModel key search, etc.

5986 3.2.2: The Functional Element displays the tModel information search results from the  
5987 identified registry server to other Functional Element.

5988 3.3: Remove tModel.

5989 3.3.1 User provides username, password and registry server URL to the Functional  
5990 Element for authentication.

5991 3.3.2: The Functional Element checks for the user validity in the identified registry server.

5992 3.3.3: Other Functional Element provides tModel key (i.e. UUID) to be removed from the  
5993 identified registry server.

5994 3.3.4: The Functional Element removes the tModel information from the private (default)  
5995 or the public UDDI registry server according to the URL provided by other Functional  
5996 Element.

5997 4: The Functional Element returns the results of the operation and the use case ends.

5998 **2.23.7.5.2.2 Alternative Flows**

5999 1: Registry Server Down.

6000 1.1: In the basic flow 2, if the identified registry server is down, the Functional Element returns  
6001 an error message and the use case ends.

6002 2: Invalid Username And Password.

6003 2.1: In the basic flow 3.1.2 and 3.3.2, if the username or password is invalid, the Functional  
6004 Element returns an error message and the use case ends.

6005 3: tModel Key Not Found.

6006 3.1: In the basic flow 3.3.3, if the tModel key cannot be found in the specified registry server,  
6007 the Functional Element returns an error message and the use case ends.

6008 **2.23.7.5.3 Special Requirements**

6009 **2.23.7.5.4 Pre-Conditions**

6010 In order to manage tModel in the registry server, the users must be registered with the registry  
6011 server. Username and password will be given when a user registers with a registry server.

6012 **2.23.7.5.5 Post-Conditions**

6013 None.

## 2.24 Service Router Functional Element (new)

### 2.24.1 Motivation

Enable capability for easy and simple mechanisms for invoking web services by:

- Providing a façade to service requesters for services location transparency, services reliability.
- Performing pre- and post- processing before and after web services invocation.

This Functional Element fulfills the following requirements from the Functional Elements Requirements:

Primary Requirements

1.3 PROCESS-250 to PROCESS-260.

Secondary Requirements

1.4 None

### 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.

Figure 30 depicts the basic concepts of how the participating entities collaborate together in the Service Router Functional Element. All the invocations from service client come to the Service router which servers as façade. The Service Router routes the invocation the actual web services. Functional handlers could be incorporated in the Functional Element or other Functional Elements. The functional handlers can be invoked before or after the actual web services are invoked.

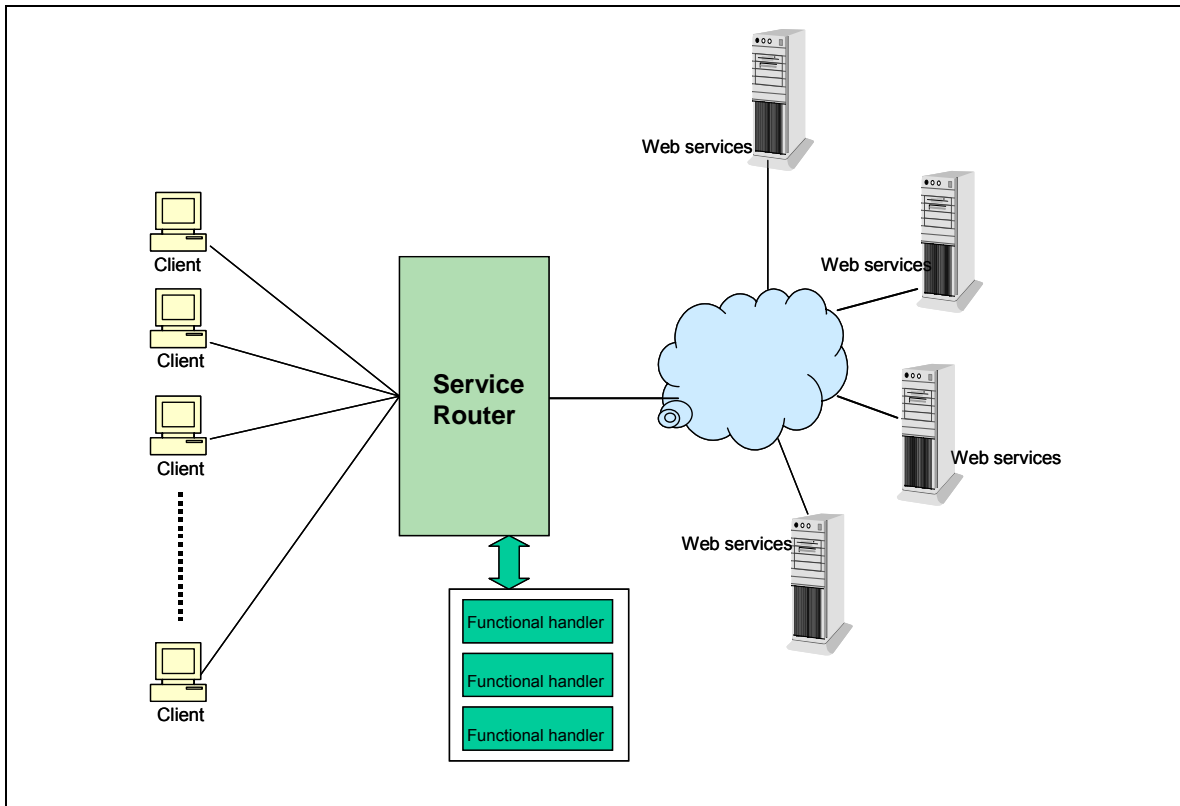


Figure 30: An Overview of the Service Router Functional Element

6039

### 6040 2.24.3 Key Features

6041 Implementations of the Service Router Functional Element are expected to provide the following  
6042 key features:

- 6043 1. The Functional Element MUST provide mechanism as façade for web services invocations.  
6044 This mechanism has the following capabilities:  
6045 1.1. Provide a single access point for web service invocation.  
6046 1.2. Provide the location transparency of actual web services.
- 6047 2. The Functional Element MUST provide capability to route web services invocation on  
6048 behalf of service requesters to the specified actual web services.
- 6049 3. The Functional Element MUST provide capability to manage web services invocation in the  
6050 aspects of invocation time-out, transaction management.
- 6051 4. The Functional Element MUST provide capability to manage the registration of web  
6052 services that are going to be invoked.
- 6053 5. The Functional Element MUST provide capability to deploy registered web services  
6054 automatically into the façade.
- 6055 6. The Functional Element MUST provide mechanism to incorporate functional handlers.
- 6056 7. The Functional Element MUST provide capability to perform processing by invoking  
6057 functional handlers defined for a web services invocation before the web services is really  
6058 invoked.
- 6059 8. The Functional Element MUST provide capability to perform processing by invoking  
6060 functional handlers for a web services invocation after the web services is invoked.



- 6061 9. The Functional Element MUST provide capability to manage functional handlers.  
6062 10. The Functional Element MUST provide capability to manage the parameter mappings  
6063 between two adjacent functional handlers and parameter mapping between functional  
6064 handler and web services.

6065

6066 In addition, the following key features could be provided to enhance the Functional Element  
6067 further:

- 6068 1. The Functional Element MAY provide capability to invoke the alternative web services if the  
6069 actual web services that is targeted to invoke is not available. The Functional Element MAY  
6070 provide the capability to define a sequence of functional handlers for a web services for a  
6071 web services invocation.

- 6072 1. The Functional Element MAY provide capability to enable the invocation of functional  
6073 handlers in pre-defined sequence for a web for a web services invocation.

6074

#### 6075 **2.24.4 Interdependencies**

6076 None.

6077

#### 6078 **2.24.5 Related Technologies and Standards**

6079 None.

6080 **2.24.6 Model**

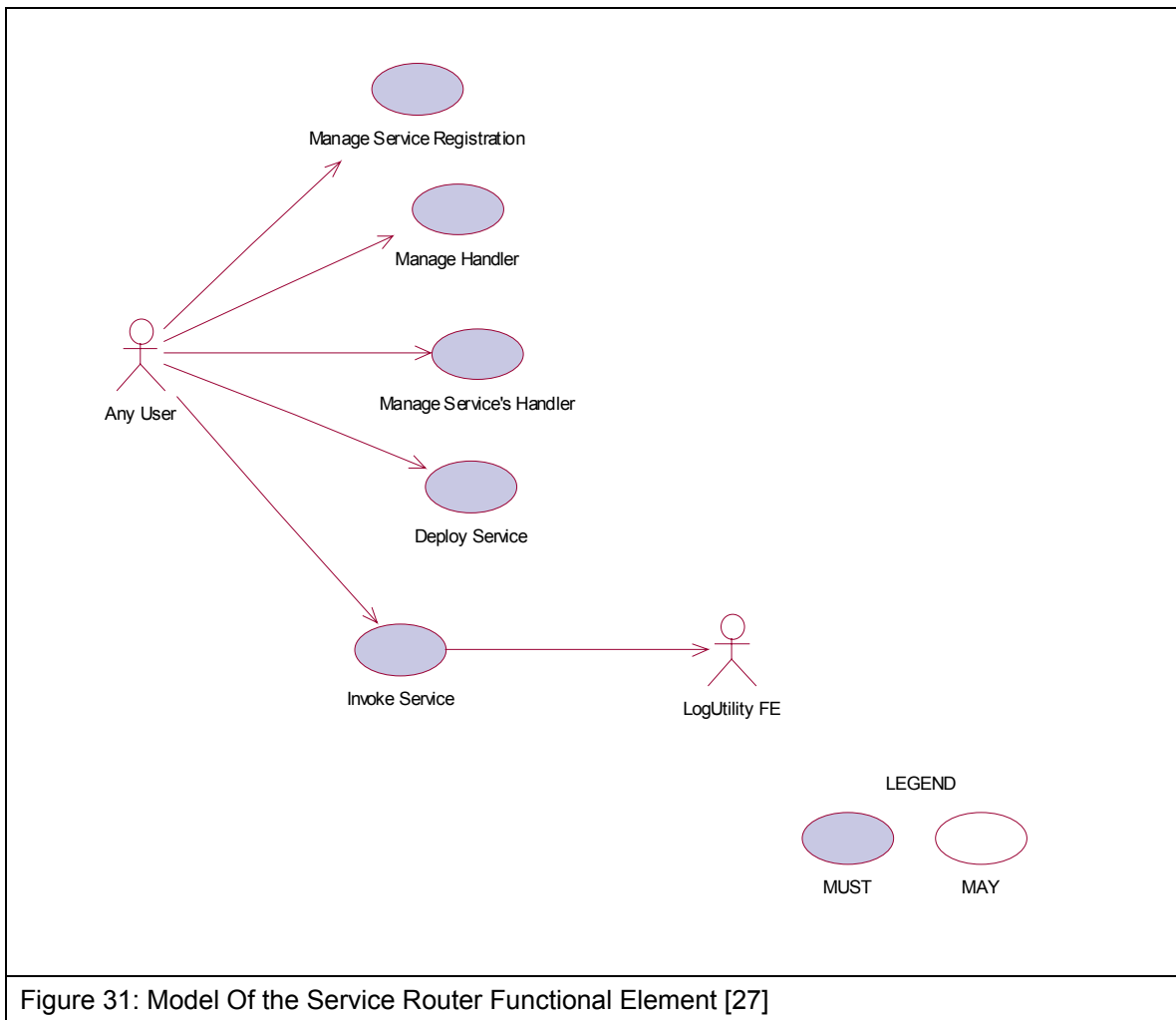


Figure 31: Model Of the Service Router Functional Element [27]

6081

6082 **2.24.7 Usage Scenarios**

6083 **2.24.7.1 Manage Service Registration**

6084 **2.24.7.1.1 Description**

6085 This use case allows the user to register, remove and view web services from or to the service  
6086 router.

- 6087 • Register Web Service

6088 Web services details are registered to the service router.

- 6089 • Delete Web Service

6090 Web services are removed from the service router.

- 6091 • View Web Service

6092 View the registration information of a web service.

## 6093 **2.24.7.1.2 Flow of Events**

### 6094 **2.24.7.1.2.1 Basic Flow**

6095 This use case starts when the user of service router wishes to register, remove and view web  
6096 services registration.

6097 1: The user initiates a request type to the Functional Element stating whether to register, remove  
6098 or view web services registration in the service router.

6099 2: The Functional Element checks the request. Based on the type of request, one of the sub-  
6100 flows is executed. If the request is to register a new web service in the service router, system  
6101 executes 'Register Web Service'. If the request is to view web services from the service router,  
6102 system executes 'View Web Services'. If the request is to remove a web service from the service  
6103 router, system executes 'Remove Web Service'.

6104 2.1: Register Web Service.

6105 2.1.1: The user provides the WSDL of a web service.

6106 2.1.2: The user provides other web service information to be kept in the service  
6107 router.

6108 2.1.3: The Functional Element retrieves web service information from the WSDL and  
6109 keeps them into the registry.

6110 2.2: View Web Services.

6111 2.2.1: The Functional Element retrieves the service from the registry with the specific  
6112 service name.

6113 2.2.2: The Functional Element returns the web services information results to the  
6114 user.

6115 2.3: Remove Web Service

6116 2.3.1: The user provides web service name to be removed from the identified  
6117 registry server.

6118 2.3.2: The Functional Element removes the web service information from the  
6119 registry.

6120 3: The Functional Element responses the status of the operation whether it is successful or failure  
6121 to the user and the use case ends.

### 6122 **2.24.7.1.2.2 Alternative Flows**

6123 1: WSDL error.

6124 1.1: In the Basic Flow 2.1.1, if the WSDL could not be retrieved, "WSDL error" will be sent  
6125 back.

6126 2: Service does not exist

6127 2.1: In the Basic Flow 2.2.1 and 2.3.1, if the service name does not exist, "Service does not  
6128 exist" error will be sent back.

6129     **2.24.7.1.3     Special Requirements**

6130     None.

6131     **2.24.7.1.4     Pre-Conditions**

6132     None.

6133     **2.24.7.1.5     Post-Conditions**

6134     None.

6135

6136     **2.24.7.2        Manage Handler**

6137     **2.24.7.2.1     Description**

6138     This use case allows any user to add, remove and view handler to the service router.

6139     **2.24.7.2.2     Flow of Events**

6140     **2.24.7.2.2.1   Basic Flow**

6141     This use case starts when the user of registry server wishes to add, remove or view web service  
6142     handlers.

6143     1: The user initiates a request type to the Functional Element stating whether to add, remove or  
6144     view web service handlers.

6145     2: The Functional Element checks the request. Based on the type of request, one of the sub-  
6146     flows is executed. If the request is to add a new web service handler to the router, system  
6147     executes 'Add Service Handler'. If the request is to view web service handlers, system executes  
6148     'View Service Handlers'. If the request is to remove a handler from the router, system executes  
6149     'Remove Service Handler'.

6150         2.1: Add Service Handler.

6151             2.1.1: The user provides handler name and location to The Functional Element.

6152             2.1.2: The service adds the information to the registry.

6153         2.2: View Service Handlers.

6154             2.2.1: The Functional Element receives a handler name from the user.

6155             2.2.2: The Functional Element returns the information of the handler to the user.

6156         2.3: Remove Service Handler.

6157             2.3.1: The user provides handler name to be removed from the service router.

6158             2.3.2: The Functional Element removes the service handler from the registry.

6159     3: The Functional Element responses the status of the operation whether it is successful or failure  
6160     to the user and the use case ends.

6161     **2.24.7.2.2   Alternative Flows**

6162     1: Handler name error.

6163         1.1: In the Basic Flow 2.2.1 and 2.3.1, if the handler name does not exist, system displays an  
6164         error message and exits the use case.

6165

6166     **2.24.7.2.3   Special Requirements**

6167     None.

6168     **2.24.7.2.4   Pre-Conditions**

6169     None.

6170     **2.24.7.2.5   Post-Conditions**

6171     None.

6172

6173     **2.24.7.3     Manage Service's Handler**

6174     **2.24.7.3.1    Description**

6175     This use case allows the user to add, remove and view handlers to the services registered in the  
6176     service router.

- 6177
  - Add a handler to a service

6178             New handler is added to a registered service.

- 6179
  - Remove a handler to a service

6180             Existing handler is removed from a registered service.

- 6181
  - View service's handler

6182             Existing handlers of a service could be viewed by the user.

6183     **2.24.7.3.2    Flow of Events**

6184     **2.24.7.3.2.1   Basic Flow**

6185     This use case starts when the user of service router wishes to add, remove or view handlers to a  
6186     service.

6187     1: The user initiates a request type to the Functional Element stating whether to add, remove or  
6188     view handlers to a service.

6189     2: The Functional Element checks the request. Based on the type of request, one of the sub-  
6190     flows is executed. If the request is to add a new web service handler to a registered web service,  
6191     system executes 'Add Service Handler'. If the request is to view web service handlers, system  
6192     executes 'View Service Handlers'. If the request is to remove a handler from a service, system  
6193     executes 'Remove Service Handler'.

6194           2.1: Add Service Handler.

6195           2.1.1: The user provides handler name, service name and parameter mappings to The  
6196           Functional Element.

6197           2.1.2: The service adds the information to the registry.

6198           2.2: View Service Handlers.

6199           2.2.1: The Functional Element receives the service name from the user.

6200           2.2.2: The Functional Element retrieves all the handlers and return to the user.

6201           2.3: Remove Service Handler.

6202           2.3.1: The user provides handler name and service name to be removed from the  
6203           service router.

6204           2.3.2: The Functional Element removes the service handler from the registry.

6205           3: The Functional Element responses the status of the operation whether it is successful or failure  
6206           to the user and the use case ends.

6207           **2.24.7.3.2.2   Alternative Flows**

6208           1: Handler name or service name does not exist.

6209           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  
6210           not exist, system displays an error message and exits the use case.

6211           **2.24.7.3.3   Special Requirements**

6212           None.

6213           **2.24.7.3.4   Pre-Conditions**

6214           None.

6215           **2.24.7.3.5   Post-Conditions**

6216           None.

6217

6218           **2.24.7.4       Deploy Service**

6219           **2.24.7.4.1    Description**

6220           This use case allows the user to deploy registered services to an application server.

6221           •   Add server information to The Functional Element

6222               New server is added to a registered service.

6223           •   Remove server information to The Functional Element

6224               Existing server is removed from a registered service.

6225           •   View server information

6226 Existing server information could be viewed by the user.

- 6227 • Deploy service

6228 Deploy a registered service to a server.

6229 .

## 6230 **2.24.7.4.2 Flow of Events**

### 6231 **2.24.7.4.2.1 Basic Flow**

6232 This use case starts when the user of service router wishes to add, remove, view server  
6233 information or deploy a web service to a server.

6234 1: The user initiates a request type to the Functional Element stating whether to add, remove or  
6235 view server's information or deploy service.

6236 2: The Functional Element checks the request. Based on the type of request, one of the sub-  
6237 flows is executed. If the request is to add a server to the router, system executes 'Add Server'. If  
6238 the request is to view server information, system executes 'View Server'. If the request is to  
6239 remove a server from the router, system executes 'Remove Server'. If the request is to deploy a  
6240 service to a server, system executes 'Deploy Service'.

6241 2.1: Add Server.

6242 2.1.1: The user provides server name and location of the server.

6243 2.1.2: The service adds the information to the registry.

6244 2.2: View Server.

6245 2.2.1: The Functional Element receives the server name from the user.

6246 2.2.2: The Functional Element retrieves the information and return to the user.

6247 2.3: Remove Server.

6248 2.3.1: The user provides the server name from the service router.

6249 2.3.2: The Functional Element removes the server from the registry.

6250 2.4: Deploy Service.

6251 2.4.1: The user provides the server name and service name from the service router.

6252 2.4.2: The Functional Element generate code package the service and deploy it to  
6253 the server.

6254 3: The Functional Element responses the status of the operation whether it is successful or failure  
6255 to the user and the use case ends.

### 6256 **2.24.7.4.2.2 Alternative Flows**

6257 1: Service name or server name does not exist.

6258 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  
6259 exist, system displays an error message and exits the use case.

6260

6261   **2.24.7.4.3   Special Requirements**

6262   None.

6263   **2.24.7.4.4   Pre-Conditions**

6264   None.

6265   **2.24.7.4.5   Post-Conditions**

6266   None.

6267

6268   **2.24.7.5       Invoke Service**

6269   **2.24.7.5.1    Description**

6270   This use case allows the user to invoke registered services through the Service Router. It is  
6271   expected to utilise the Notification FE and Log Util FE in the implementation of this use case.

6272   **2.24.7.5.2    Flow of Events**

6273   **2.24.7.5.2.1   Basic Flow**

6274   This use case starts when the user of service router wishes to invoke a deployed or registered  
6275   service.

6276   1: The user initiates a request to the Service Router.

6277   2: The Functional Element checks the request, and determines if the invoked service has any  
6278   pre-invocation Functional Handlers. If so, the handlers are invoked.

6279   3: The Functional Element then routes the request to the actual service based on registration  
6280   information captured.

6281   4: When the result from the actual service is returned, the Functional Element checks if there is  
6282   any post-invocation Functional Handlers. If so, the handlers are invoked.

6283   5: The Functional Element returns the result of invocation to the user and the use case ends.

6284

6285   **2.24.7.5.2.2   Alternative Flows**

6286   1: Functional Handlers are not available.

6287       1.1: In the Basic Flow 2 and 4, if the Functional Handlers are not available, an error message  
6288       will be returned, and the use case ends.

6289   2: Invoked Service is not available.

6290       2.1: In the Basic Flow 3, if the invoked Service is not available, an error message will be  
6291       returned, and the use case ends.

6292



6293     **2.24.7.5.3     Special Requirements**

6294     None.

6295     **2.24.7.5.4     Pre-Conditions**

6296     None.

6297     **2.24.7.5.5     Post-Conditions**

6298     None.

6299

6300 **2.25 Service Tester Functional Element (Deprecated)**

6301

6302 This Functional Element has been deprecated in this version. Please refer to its replacement,  
6303 2.15 QoS Functional Element (new) for further details.

## 2.26 Transformer Functional Element (new)

### 2.26.1 Motivation

Different applications support different format of files or message. Sometimes same information needs to be represented in different format in different use cases. This element tries to provide a framework to facilitate transformation between files or messages.

This Functional Element fulfills the following requirements from the Functional Elements Requirements:

- Primary Requirements
  - DELIVERY-150,
  - DELIVERY-151,
  - DELIVERY-152,
  - DELIVERY-153,
  - DELIVERY-155, and
  - DELIVERY-157.

### 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

Figure 32 depicts the basic concepts of 2 steps approach of Transformer Functional Element. Step 1 begins when the user (service requester) requests to define supported message, file types, XSLT templates and process handlers. The Function Element persists these definitions the return the results. Step 2 begins when the user requests for file or message transformation. The user provides messages or files to be transformed. The Functional Element will do the transformation and returns the result to the user.

6329

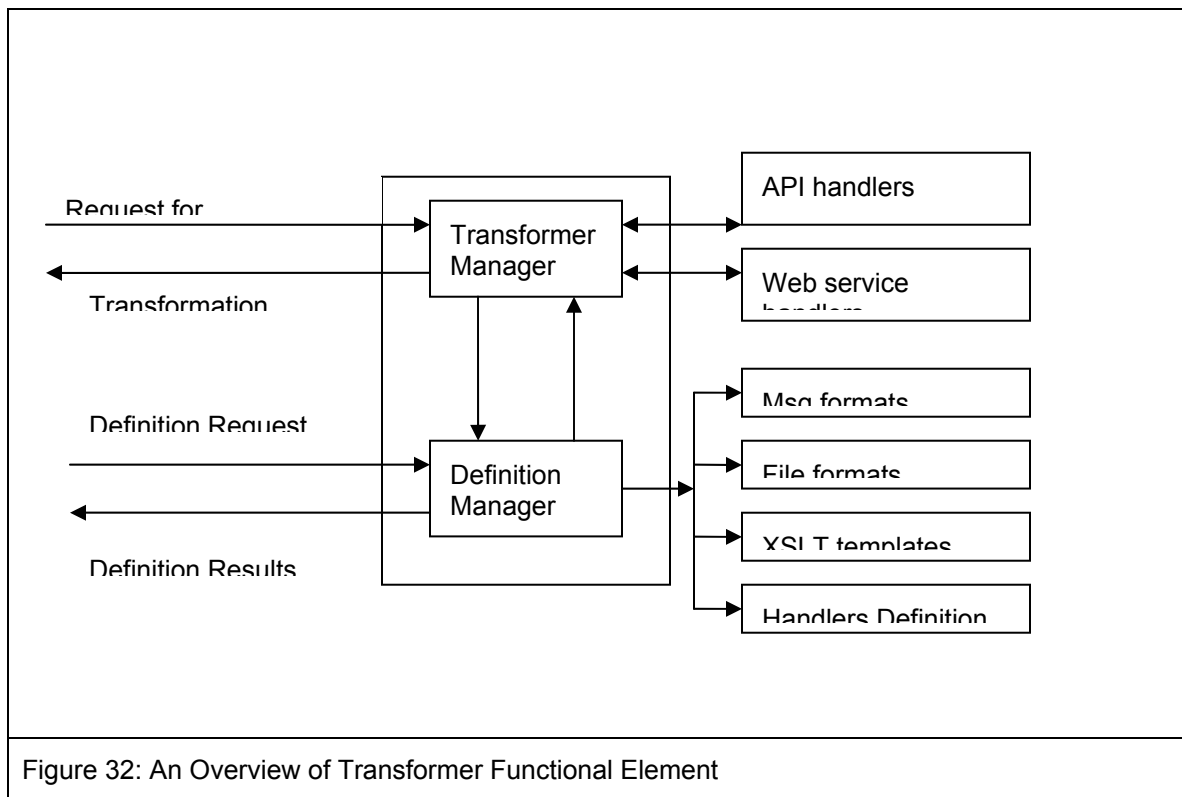


Figure 32: An Overview of Transformer Functional Element

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

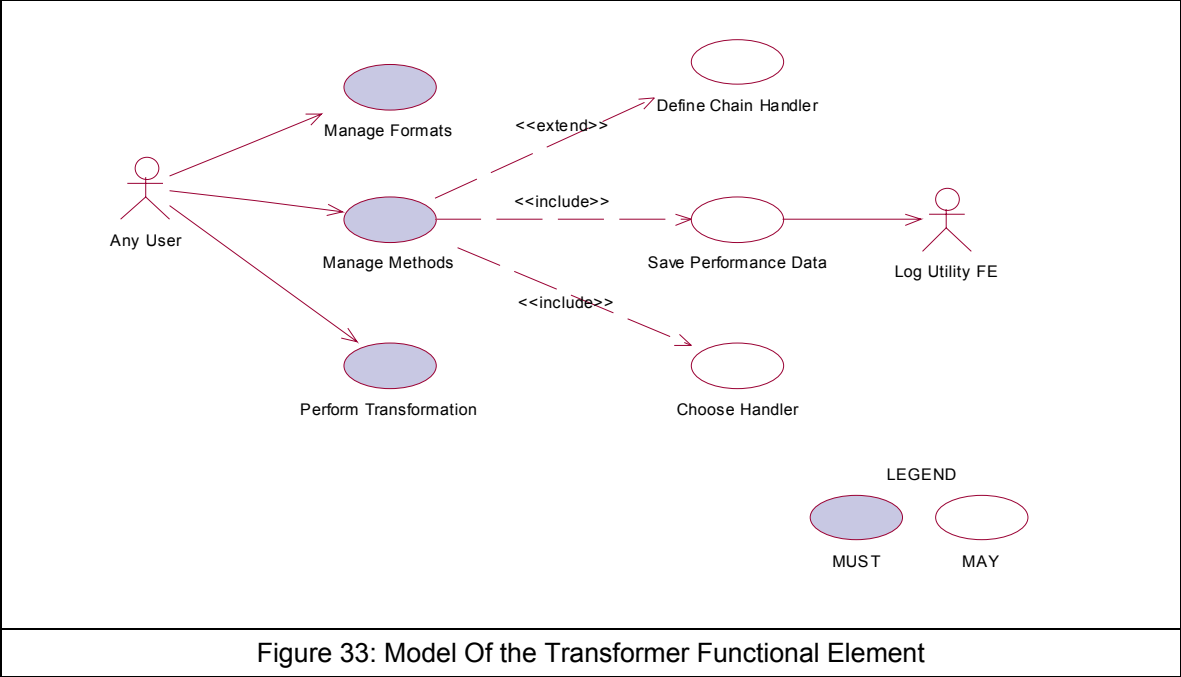


Figure 33: Model Of the Transformer Functional Element

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

## 2.27 User Management Functional Element

### 2.27.1 Motivation

The User Management Functional Element is expected to be an integral part of the user access management (UAM) functionalities that is expected to be needed by a Web Service-enabled implementation. This FE is expected to fulfill the needs arising out of managing resources within an application, with a user-centric viewpoint. As such it will cover aspects that include:

- Basic user accounts management facilities,
- Ability to extend dynamically from the basic set of account information,
- Capability for configurable policies governing account management,
- Providing log trails for user activities, and
- Management of user authentication means, either directly or indirectly.

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

#### Primary Requirements

- MANAGEMENT-001 to MANAGEMENT-003,
- MANAGEMENT-005,
- MANAGEMENT-008,
- MANAGEMENT-012, and
- SECURITY-002 (all).

#### Secondary Requirements

- SECURITY-001.

### 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

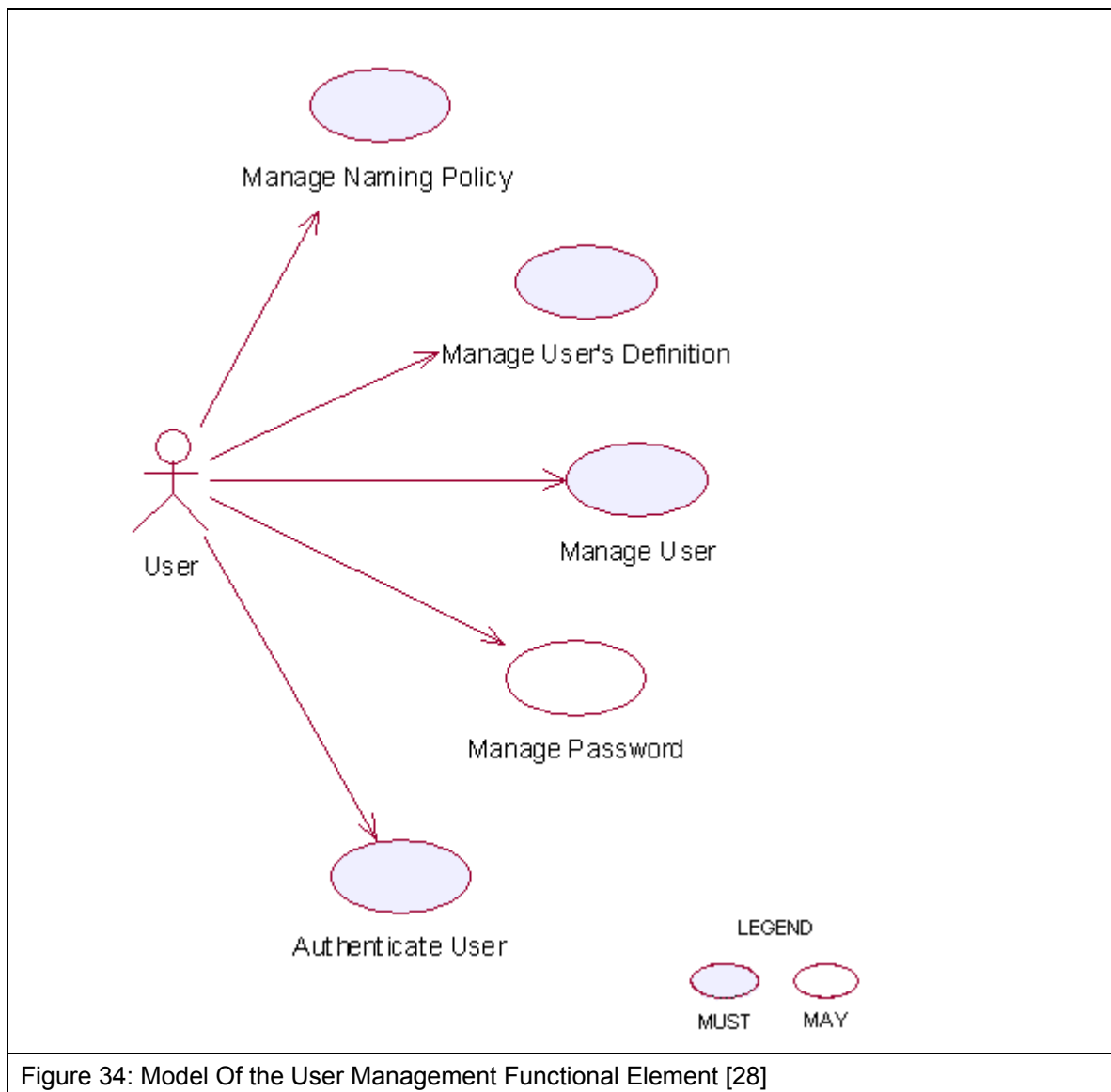


Figure 34: Model Of the User Management Functional Element [28]

## 2.27.7 Usage Scenarios

### 2.27.7.1 Manage Naming Policy

#### 2.27.7.1.1 Description

This use case allows any user to manage naming policy when creating/updating user accounts. The service user may create, update, retrieve and delete a naming policy.

#### 2.27.7.1.2 Flow of Events

##### 2.27.7.1.2.1 Basic Flow

This use case starts when any user wants to manage naming policy for creating/updating user 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.

## 2.28 Web Service Aggregator Functional Element

### 2.28.1 Motivation

In any Web Service-enabled application, it is expected that complex business functions have to be realized via aggregation of multiple Web Services. This Functional Element is expected to fulfill the needs arising out of Web Services composition. As such it will cover aspects that include:

Facilitating the composition of Web Services, and  
Testing of aggregated Web Services.

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

Primary Requirements

- PROCESS-010 to PROCESS-014.

Secondary Requirements

- PROCESS-131

### 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.

The following diagram shows the meaning of the terms in the context of Web Services aggregation.

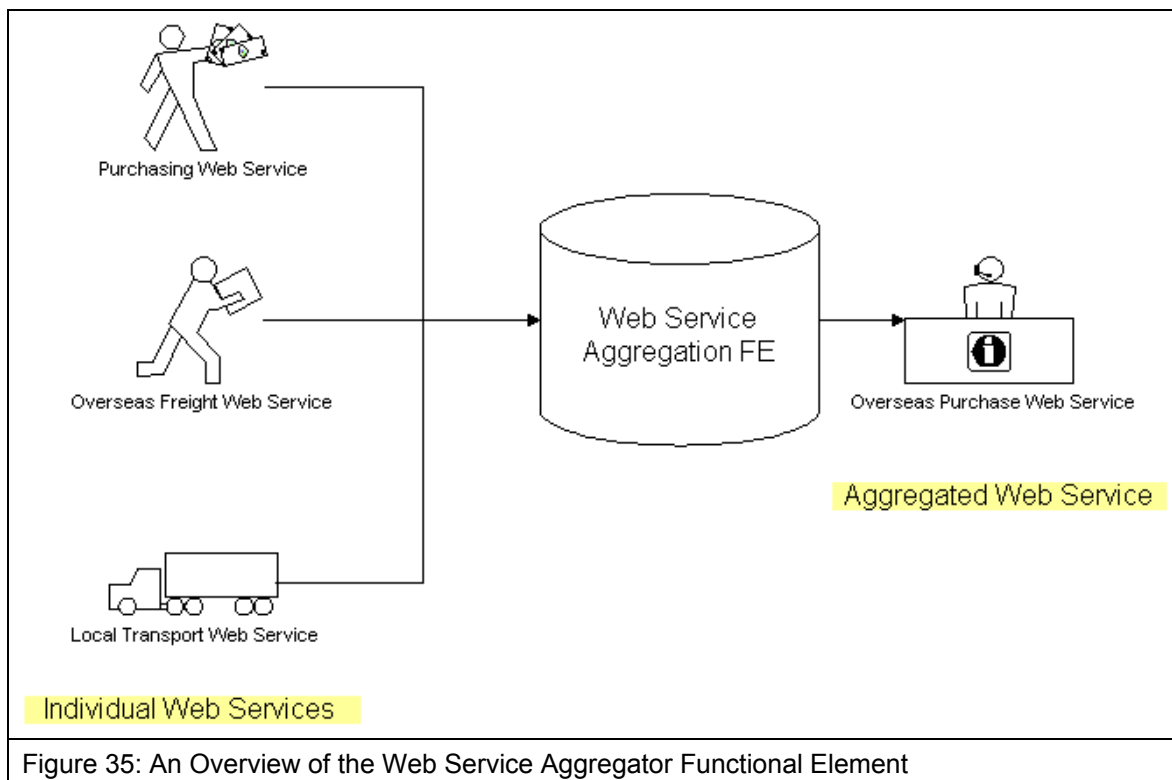


Figure 35: An Overview of the Web Service Aggregator Functional Element

### 2.28.3 Key Features

Implementations of the Web Service Aggregator Functional Element are expected to provide the following key features:

1. The Functional Element MUST provide a mechanism for composing any number of Web Services into single Web Service according to specified Composition Rule(s).
2. Individual web services can reside at any location, but it is expected to be accessible.
3. As part of Key Feature (1), the WSDL of a web service used for composition MUST be available.
4. The Functional Element MUST support the definition, modification and removal of Composition Rules.
5. The Functional Element MUST encapsulate the composition logic used into an interpretable XML-based script based on a particular standard\*.

*Example: BPEL or WSCI. The TC will have to decide on which standard to use*

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

1. The Functional Element MAY provide the capability to transform the interpretable XML-based script into an executable program.
2. If Key Feature (1) is provided, then the Functional Element MAY also have the following capabilities:
  - 2.1 The ability to test the functionality of the aggregated Web Service,
  - 2.2 A WSDL to describe the aggregated Web Service, and
  - 2.3 The capability to publish the aggregated Web Service into an UDDI-compliant registry

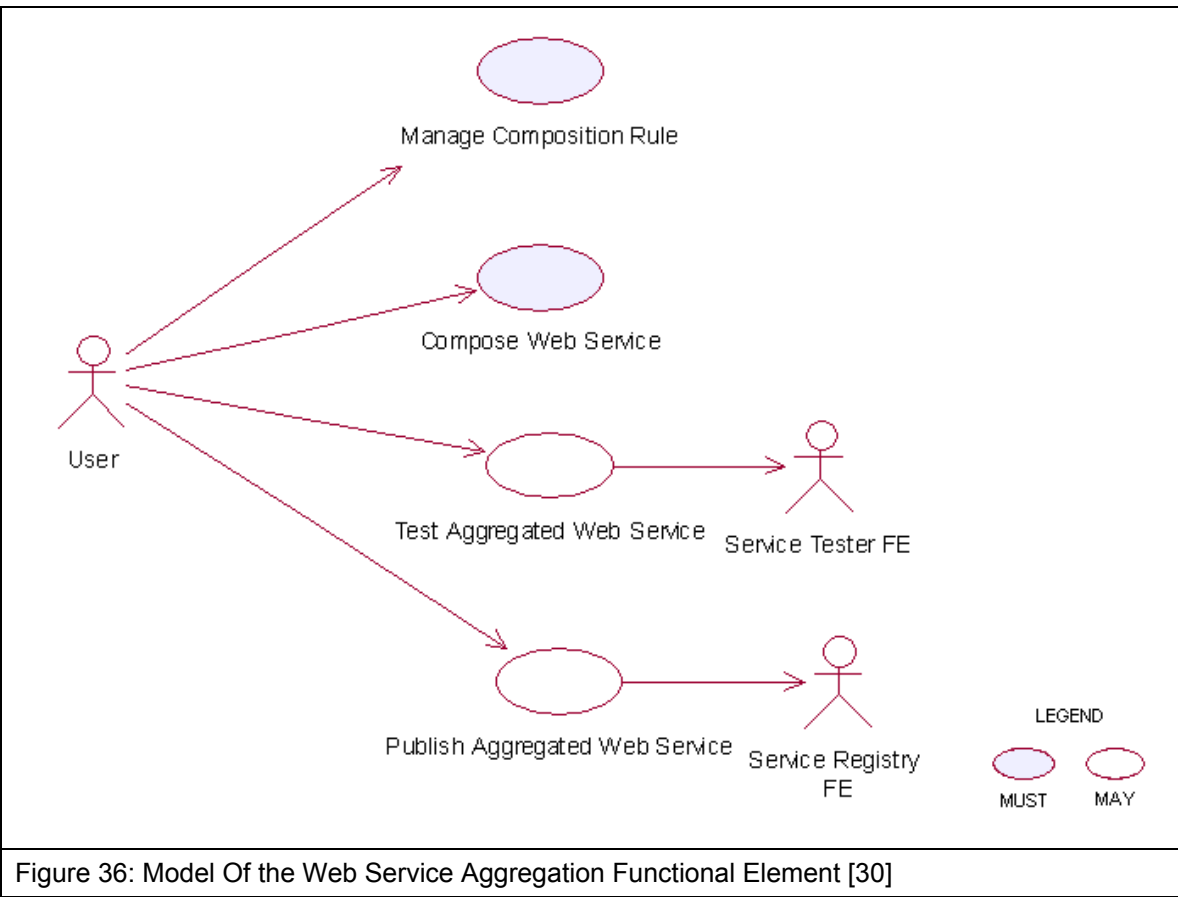
6835     **2.28.4     Interdependencies**

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**



6841	<b>2.28.7</b>	<b>Usage Scenarios</b>
6842	<b>2.28.7.1</b>	<b>Manage composition rule</b>
6843	<b>2.28.7.1.1</b>	<b>Description</b>
6844	This use case allows the user to manage the composition rule used for Web Services aggregation.	
6845		
6846	<b>2.28.7.1.2</b>	<b>Flow of Events</b>
6847	<b>2.28.7.1.2.1</b>	<b>Basic Flow</b>
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 operation.	
6850		
6851	2: Based on the operation it specified, one of the following sub-flows is executed:	
6852	If the operation is ' <b>Define a rule</b> ', then sub-flow 2.1 is executed.	
6853	If the operation is ' <b>Update a rule</b> ', then sub-flow 2.2 is executed.	
6854	If the operation is ' <b>Retrieve a rule</b> ', then sub-flow 2.3 is executed.	
6855	If the operation is ' <b>Remove a rule</b> ', 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, the sequence specification, parameters mapping between Web Services.	
6858		
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 mechanism.	
6864		
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 **2.28.7.2.2.2 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 **2.28.7.3.2.2 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 UDDI.

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

---

### 3 Functional Elements Usage Scenarios

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:

- A service monitoring solution for the management of services in a SOA model
- Enabling security through the Secure SOAP Functional Element
- Decoupled User Access Management with support for multi-domain capabilities in a web service environment
- Single-Sign On for Distributed Services (Applications)

### 3.1 Service Monitoring

In a SOA environment, management of services includes the capability to monitor services within the management domain. These includes:

Monitoring the performance of services invoked

Generating audit trails of services invoked

Monitoring and testing the availability of services on the remote machine (server)

A basic solution can be realised through the aggregation of two Functional Element, namely Service Management and Service Tester, as shown in Figure 19. This solution can be improved with notification capabilities, using the Notification Engine, be it to a remote client, a system administrator or an end user of a particular service. Further enhancement can be added with a Rule Engine that will have the cognitive ability to make decisions. An example of this enhancement would be the ability to decide when should notifications or alerts be sent and in what form.

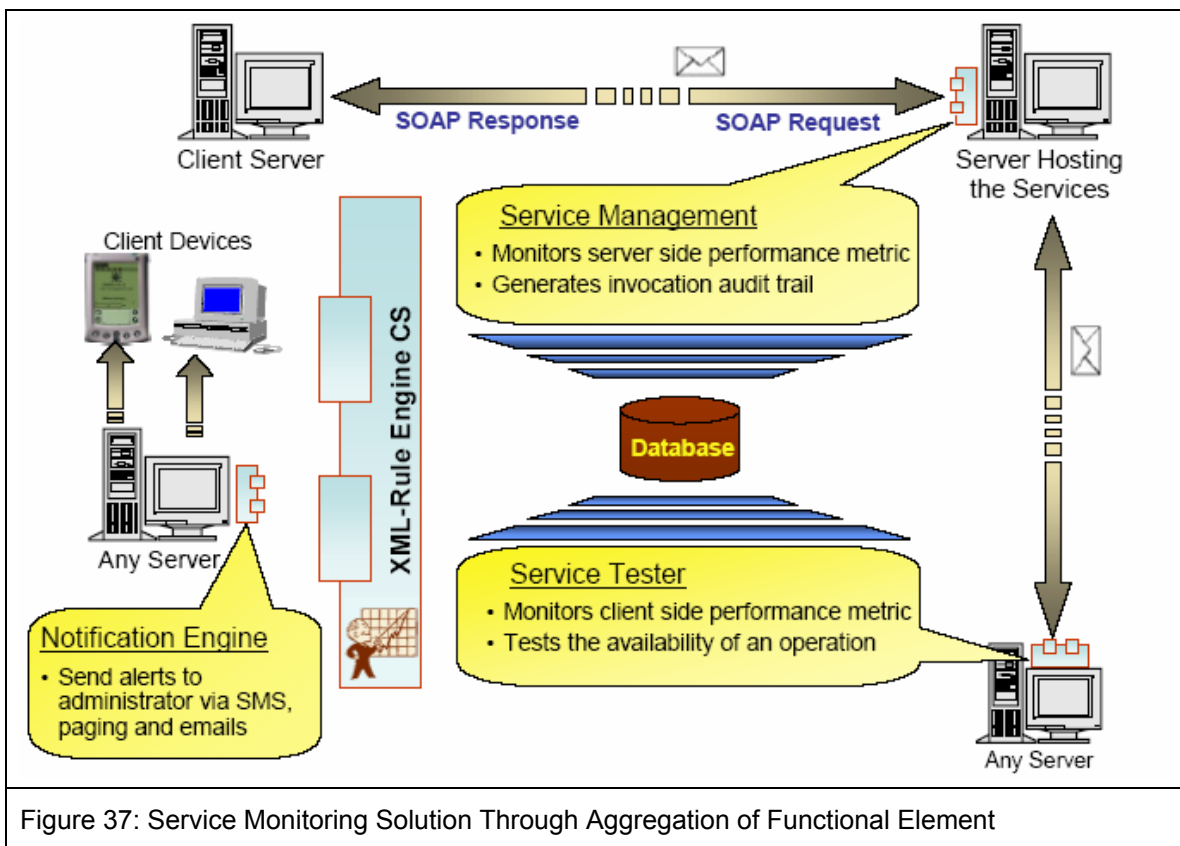


Figure 37: Service Monitoring Solution Through Aggregation of Functional Element

## 3.2 Securing SOAP Messages

SOAP in its pure form does not have any built in security as it is meant to be a simple and lightweight protocol. As such, where security is needed, additional capabilities must be provided. Presently, standards like XML Encryption and XML Signature are available. Making use of these standards, the Secure Soap Functional Element, when deployed on both the sending and receiving parties, will be able to provide encryption and signing of messages as illustrated in Figure 20.

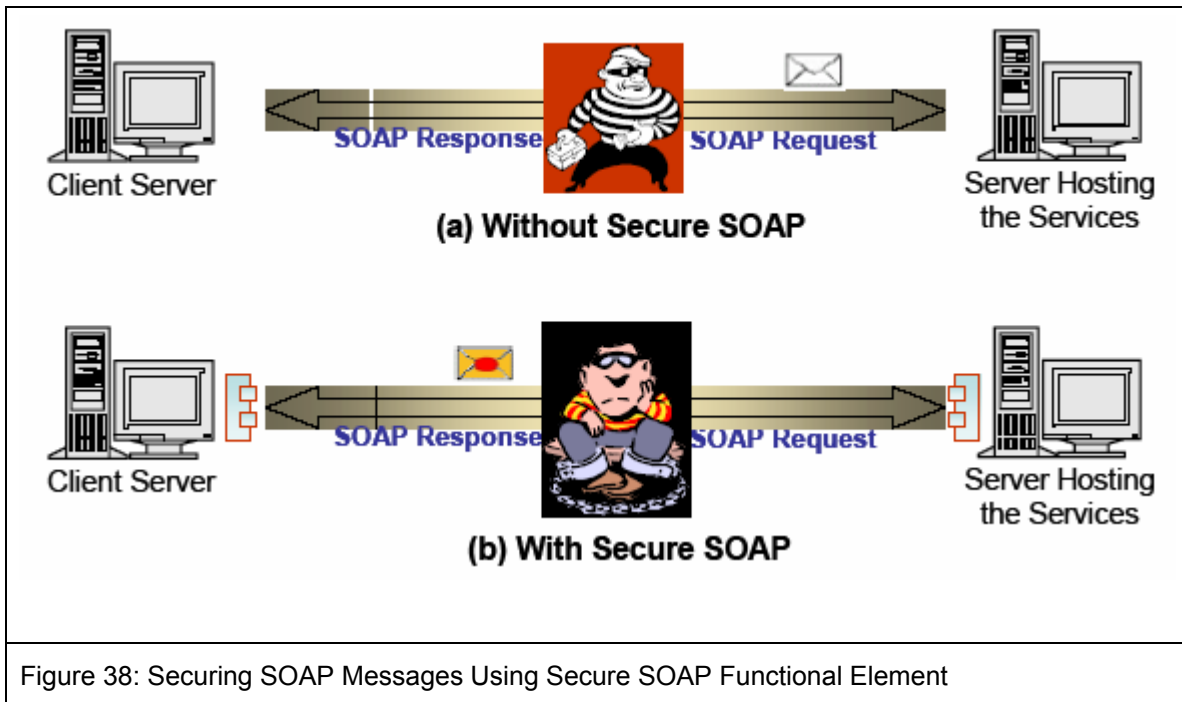


Figure 38: Securing SOAP Messages Using Secure SOAP Functional Element

### 3.3 Decoupled User Access Management

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.

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:

- User Management only, or
- User Management and Group Management, or
- User Management and Role & Access Management, or
- User Management, Group Management and Role & Access Management, or
- All the four Functional Elements in tandem

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.

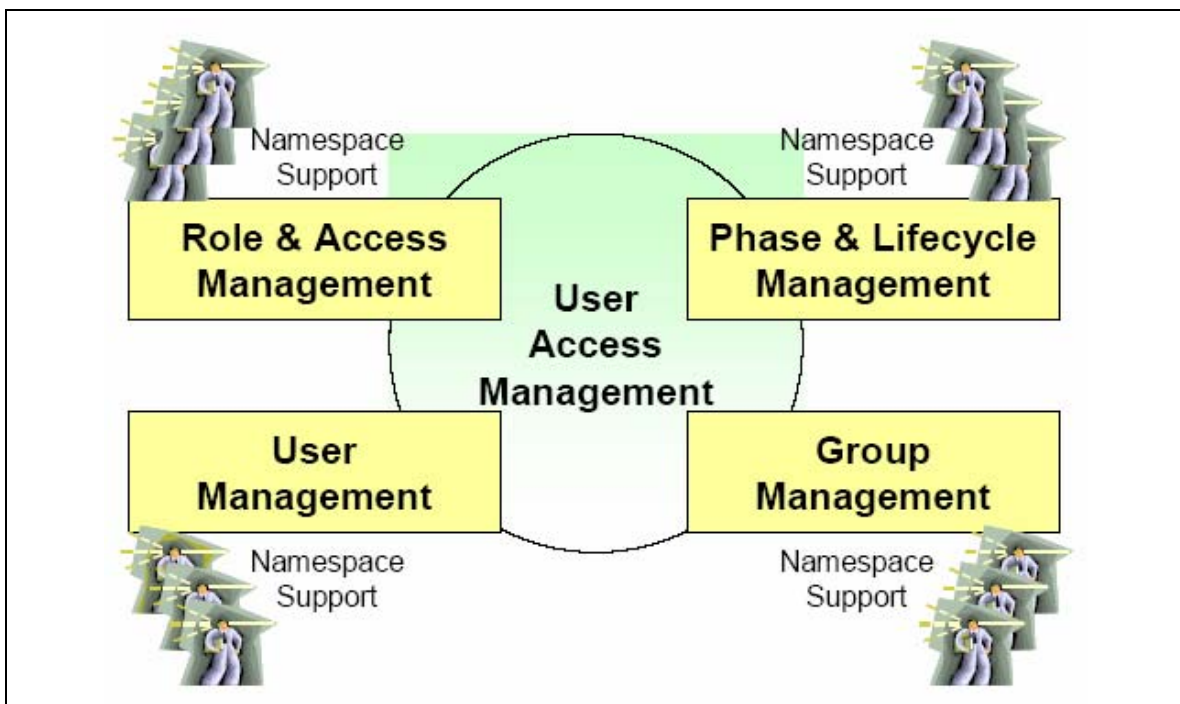


Figure 39: User Access Management via Functional Element



### 3.4 Single-Sign-On for Distributed Services (Applications)

In a SOA world, it is very likely that services for a composite application can be potentially made up of multiple 3<sup>rd</sup> party services from different application domains. It is also very likely that each of these domains will require authentication of the user separately. However, it is not user friendly to enforce re-authentication as the user moves from one domain to another. Using the Identity Management Functional Element, with the potential combination of Secure SOAP Functional Element and other user access management Functional Elements like User Management, a solution for such an environment can be put together to enable Single-Sign-On. In this scenario of use, a Circle of Trust between different application domains can be established using the Identity Management Functional Element, and the exchanges between these domains can be secured using the Secure SOAP Functional Element. Access and authentication to individual domains remain the purview of the distributed applications, and can potentially also leveraged on the Decoupled User Access Management scenario detailed in section 3.3.

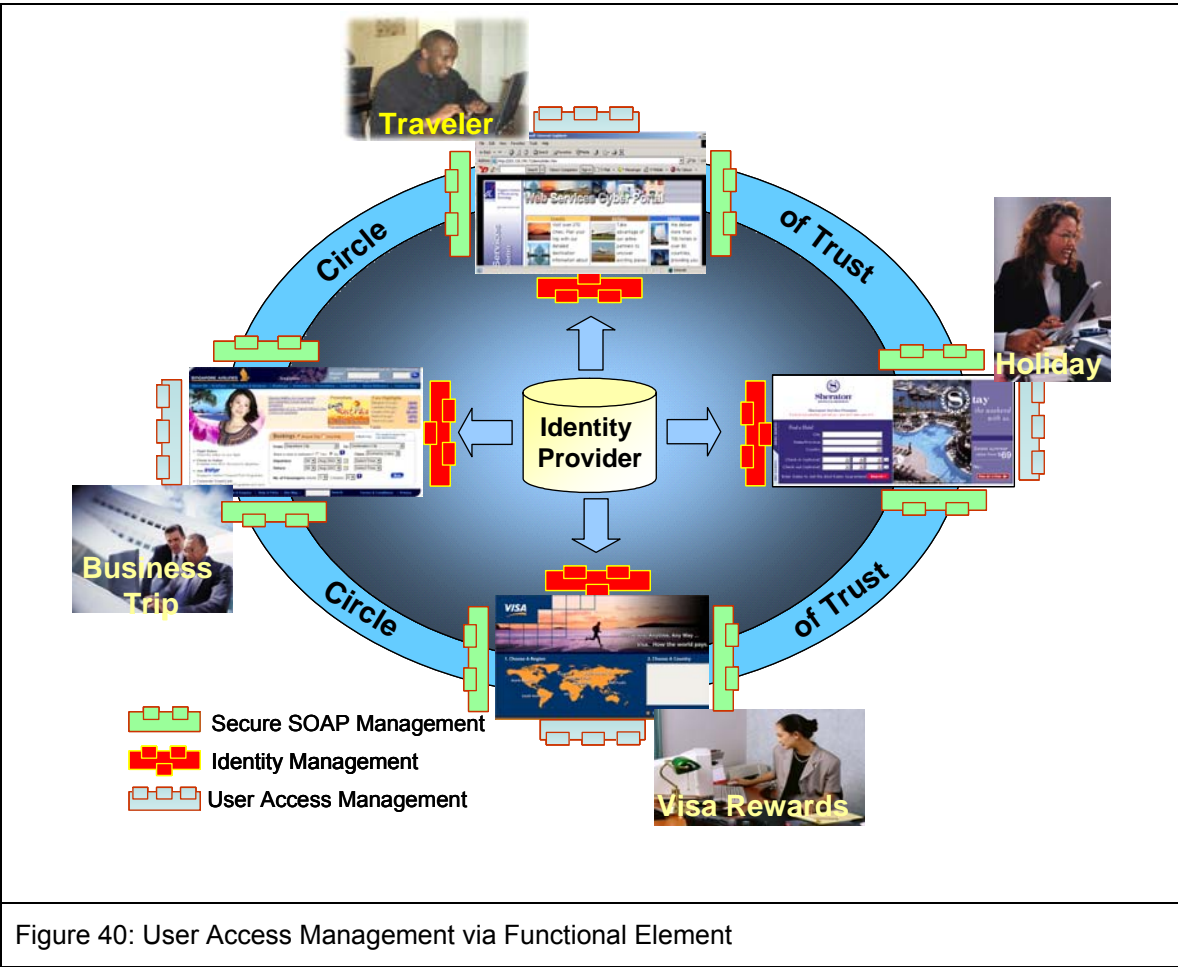


Figure 40: User Access Management via Functional Element

## 4 References

1. FWSI TC, OASIS, **Web Service Implementation Methodology Working Draft 0.1**, <http://www.oasis-open.org/apps/org/workgroup/fwsi/documents.php>, September 2004.
2. FWSI TC, OASIS, **Functional Elements Requirements Approved Document 02**, <http://www.oasis-open.org/apps/org/workgroup/fwsi/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](http://www.projectliberty.org/specs/index.html#ID-FF_Specs).
9. Liberty Alliance, **ID-WSF 1.0 Specifications**, version 1.0, [http://www.projectliberty.org/specs/index.html#ID-WSF\\_Specs](http://www.projectliberty.org/specs/index.html#ID-WSF_Specs).
10. **Web Services Federation Language (WS-Federation)**, <http://www-106.ibm.com/developerworks/webservices/library/ws-fed/>, July 2003.
11. Chan, L.P., **WSRA Use Case Specifications – Identity Management**, version 0.3, JSSL of Singapore Institute of Manufacturing Technology, December 2003.
12. Yin, Z.L., **WSRA Use Case Specifications – Log Utility**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
13. Limbu, D.K., **WSRA Use Case Specifications - Notification Engine**, version 1.2, JSSL of Singapore Institute of Manufacturing Technology, December 2002.
14. Wu, Y.Z., **WSRA Use Case Specifications - Phase & LC Management**, version 1.3, JSSL of Singapore Institute of Manufacturing Technology, October 2003.
15. 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.

---

## Appendix A. Acknowledgments

Special thanks to the following individuals who contributed significantly towards to the of this specification:

- Chan Lai Peng
- Cheng Yushi
- Dilip Kumar Limbu
- V. Ramasamy
- Wu Yingzi
- Xu Xingjian, and
- Yin Zunliang.

The committee would also like to express its appreciation for the encouragement and guidance provided by Jamie Clark throughout the course of the TC work.

The committee would also like to record its heartfelt appreciation to IBM Rational (Singapore) Pte. Ltd. for kindly agreeing to allow the use of the Rational Tools towards the creation of the Use Case Model used in this document.

## Appendix B. Revision History

The following revision of this document represents the major milestones achieved.

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"><li>• Deprecated 2 FEs, namely Presentation Transformer and Service Tester</li><li>• Replaced the deprecated FEs with Transformer and Quality of Service (QoS) FEs respectively</li><li>• Added 10 new FEs identified for version 2.0</li><li>• Minor changes to the following FEs:<ul style="list-style-type: none"><li>○ Phase &amp; Lifecycle Management</li><li>○ Secure SOAP Management</li><li>○ Sensory</li><li>○ Service Management</li><li>○ Service Registry</li><li>○ Web Service Aggregator</li></ul></li><li>• Usage Scenarios (added 1 more</li></ul>

Rev	Date	By Whom	What
			usage scenario for SSO)
fws-fe-2.0-guidelines-spec-wd-02.doc	20-Dec-2005	Puay Siew Tan	<p>Revision of working draft for Version 2.0 of the FE Specs. This is based on feedback/comments received todate:</p> <ul style="list-style-type: none"> <li>• Added the “Deprecated” phrase in the title of Presentation Transformer and Service Tester. Easier for readers to see.</li> <li>• Added the checking of permission sets for Data Integrator</li> <li>• Added Invoke Service Use Case in Service Router</li> <li>• Corrected some minor syntax and grammar errors</li> </ul>

7075

---

## Appendix C. Notices

7076

7077 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
7078 that might be claimed to pertain to the implementation or use of the technology described in this  
7079 document or the extent to which any license under such rights might or might not be available;  
7080 neither does it represent that it has made any effort to identify any such rights. Information on  
7081 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
7082 website. Copies of claims of rights made available for publication and any assurances of licenses  
7083 to be made available, or the result of an attempt made to obtain a general license or permission  
7084 for the use of such proprietary rights by implementors or users of this specification, can be  
7085 obtained from the OASIS Executive Director.

7086 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
7087 applications, or other proprietary rights which may cover technology that may be required to  
7088 implement this specification. Please address the information to the OASIS Executive Director.

7089 Copyright © OASIS Open 2004. All Rights Reserved.

7090 This document and translations of it may be copied and furnished to others, and derivative works  
7091 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
7092 published and distributed, in whole or in part, without restriction of any kind, provided that the  
7093 above copyright notice and this paragraph are included on all such copies and derivative works.  
7094 However, this document itself does not be modified in any way, such as by removing the  
7095 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS  
7096 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual  
7097 Property Rights document must be followed, or as required to translate it into languages other  
7098 than English.

7099 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
7100 successors or assigns.

7101 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
7102 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
7103 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE  
7104 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
7105 PARTICULAR PURPOSE.