Internet Engineering Task Force
Internet-Draft
Updates: 5812 (if approved)
Intended status: Standards Track
Expires: March 15, 2015

E. Haleplidis
University of Patras

September 11, 2014

ForCES Model Extension
draft-ietf-forces-model-extension-05

Abstract

This memo extends the Forwarding and Control Element Separation (ForCES) model defined in RFC 5812 and updates RFC5812 to allow complex datatypes for metadata, optional default values for datatypes, optional access types for structures. It fixes an issue with Logical Functional Block (LFB) inheritance. It also introduces two new features a new event condition BecomesEqualTo and LFB properties. The changes introduced in this memo do not alter the protocol and retain backward compatibility with older LFB models.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 15, 2015.

Copyright Notice

Table of Contents

1. Introduction

The ForCES Model [RFC5812] presents a formal way to define Forwarding Elements (FE) Logical Function Blocks (LFBs) using the eXtensible Markup Language (XML). [RFC5812] has been published a more than two years and current experience in its use has demonstrated need for adding new and changing existing modeling concepts.

Specifically this document updates the ForCES Model [RFC5812] to allow complex datatypes for metadata (Section 2.1), optional default values for datatypes (Section 2.2), optional access types for structures (Section 2.3) and fixes an issue with LFB class inheritance (Section 2.6). Additionally the document introduces two new features, a new event condition BecomesEqualTo (Section 2.4) and LFB properties (Section 2.5).

These extensions are an update to the ForCES model [RFC5812] and do not require any changes on the ForCES protocol [RFC5810] as they are simply changes of the schema definition. Additionally backward compatibility is ensured as XML libraries produced with the earlier schema are still valid with the new one. In order for XML libraries

produced by the new schema to be compatible with existing ForCES implementations, the XML libraries MUST NOT include any of the features described in this document.

Extensions to the schema and excerpts of the schema include the tags <!-- Extension RFC XXXX --> and <!-- /Extension RFC XXXX --> which designates the beginning and ending of extension text specified by this document in respect to the original ForCES Model [RFC5812] schema.

1.1. Requirements Language

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].

1.2. Definitions

This document uses the terminology defined in the ForCES Model in [RFC5812]. In particular, the reader is expected to be familiar with the following terms:

FE Model

LFB (Logical Functional Block) Class (or type)

LFB Instance

LFB Model

Element

Attribute

LFB Metadata

ForCES Component

LFB Class Library

2. ForCES Model Extensions

2.1. Complex datatypes for Metadata

Section 4.6. (Element for Metadata Definitions) in the ForCES Model [RFC5812] limits the datatype use in metadata to only atomic types. Figure 1 shows the xml schema excerpt where only typeRef and atomic are allowed for a metadata definition.


```
<xsd:complexType name="metadataDefsType">
  <xsd:sequence>
    <xsd:element name="metadataDef" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN"/>
          <xsd:element ref="synopsis"/>
          <xsd:element name="metadataID" type="xsd:integer"/>
          <xsd:element ref="description" minOccurs="0"/>
          <xsd:choice>
            <xsd:element name="typeRef" type="typeRefNMTOKEN"/>
            <xsd:element name="atomic" type="atomicType"/>
          </xsd:choice>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Figure 1: Initial MetadataDefType Definition in the schema

However there are cases where complex metadata are used in the datapath, for example two simple use cases can be seen in the OpenFlow 1.1 specification [OpenFlowSpec1.1] and beyond:

1. The Action Set metadata is an array of actions descriptors, which traverses the processing pipeline along with the packet data.

2. When a packet is received from a controller it may be accompanied

by a list of actions, as metadata, to be performed on it prior to being sent on the processing pipeline. This list of actions is also an array.

With this extension (Figure 2), complex data types are also allowed, specifically structs and arrays as metadata. The key declarations are required to check for validity of content keys in arrays and componentIDs in structs.


```
<xsd:complexType name="metadataDefsType">
<xsd:sequence>
  <xsd:element name="metadataDef" maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:NMTOKEN"/>
        <xsd:element ref="synopsis"/>
        <xsd:element name="metadataID" type="xsd:integer"/>
        <xsd:element ref="description" minOccurs="0"/>
        <xsd:choice>
          <xsd:element name="typeRef" type="typeRefNMTOKEN"/>
          <xsd:element name="atomic" type="atomicType"/>
          <!-- Extension RFC XXXX -->
          <xsd:element name="array" type="arrayType">
            <xsd:key name="contentKeyID1">
              <xsd:selector xpath="lfb:contentKey"/>
              <xsd:field xpath="@contentKeyID"/>
            </xsd:key>
          </xsd:element>
          <xsd:element name="struct" type="structType">
            <xsd:key name="structComponentID1">
              <xsd:selector xpath="lfb:component"/>
```

```
                    <xsd:field xpath="@componentID"/>
                  </xsd:key>
                </xsd:element>
                <!-- /Extension RFC XXXX -->
              </xsd:choice>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
```

Figure 2: New MetadataDefType Definition for the schema

2.2. Optional Default Value for Datatypes

In the original schema, default values can only be defined for datatypes defined inside LFB components and not inside structures or arrays. Therefore default values of datatypes that are constantly being reused, e.g. counters with default value of 0, have to be constantly respecified. Additionally, datatypes inside complex datatypes cannot be defined with a default value, e.g. a counter inside a struct that has a default value of 0.

This extension allows the option to add default values to datatypes. These datatypes can then be referenced as simple components or within complex datatypes such as structs. A simple use case would be to

have a struct component where one of the components is a counter which the default value would be zero. To achieve that the counter must first be defined as a datatype with the required default value and then referenced in the struct. Default values MUST adhere the following formal restrictions:

1. Default Values MUST be ignored if the data type is not an atomic or a base data type.

2. When a datatype X with default value A is referenced from a datatype Y with a default value B, the default value of the datatype that references overrides the referenced default value, e.g. in this case Y's default value will be B.

3. Default Values of LFB components overrides any default value

specified from the dataTypeDef definition.

4. Default Values of datatypes reference in capabilities or metadata MUST be ignored.

This extension simply appends to the definition of the dataTypeDefsType in the XML schema from Figure 3 to Figure 4 to allow default values to dataTypeDefsType.

```
<xsd:complexType name="dataTypeDefsType">
  <xsd:sequence>
    <xsd:element name="dataTypeDef" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN"/>
          <xsd:element name="derivedFrom" type="xsd:NMTOKEN"
             minOccurs="0"/>
          <xsd:element ref="synopsis"/>
          <xsd:element ref="description" minOccurs="0"/>
          <xsd:group ref="typeDeclarationGroup"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Figure 3: Initial Excerpt of dataTypeDefsType Definition in the schema


```
<xsd:complexType name="dataTypeDefsType">
  <xsd:sequence>
    <xsd:element name="dataTypeDef" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:NMTOKEN"/>
          <xsd:element name="derivedFrom" type="xsd:NMTOKEN"
             minOccurs="0"/>
```

```
                <xsd:element ref="synopsis"/>
                <xsd:element ref="description" minOccurs="0"/>
                <xsd:group ref="typeDeclarationGroup"/>
                <!-- Extension RFC XXXX -->
                <xsd:element name="defaultValue" type="xsd:token"
                   minOccurs="0"/>
                <!-- /Extension RFC XXXX -->
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
```

Figure 4: New Excerpt of dataTypeDefsType Definition in the schema

Examples of using default values is depicted in Figure 5.


```
            <dataTypeDef>
```

```
        <name>ZeroCounter</name>
        <synopsis>A counter with default 0</synopsis>
        <typeRef>uint32</typeRef>
        <defaultValue>0</defaultValue>
      </dataTypeDef>
      <dataTypeDef>
      <name>CounterValues</name>
      <synopsis>Example default values in struct</synopsis>
      <struct>
        <component componentID="1">
          <name>GoodPacketCoutner</name>
          <synopsis>A counter for good packets</synopsis>
          <typeRef>ZeroCounter</typeRef>
        </component>
        <component componentID="2">
          <name>BadPacketCoutner</name>
          <synopsis>A counter for bad packets</synopsis>
          <typeRef>ZeroCounter</typeRef>
        </component>
      </struct>
    </dataTypeDef>
```

Figure 5: Example of optional default values

2.3. Optional Access Type for Structs

In the original schema, the access type can only be defined on components of an LFB and not on components within structs or arrays. That means that when it is a struct datatype it is not possible to fine-tune access type per component within the struct. A simple use case would be to have a read-write struct component where one of the components is a counter where the access-type could be read-reset or read-only, e.g. a read-reset or a read-only counter inside a struct.

This extension allows the definition of the access type for a struct component either in the datatype definitions or in the LFB component definitions.

When optional access type for components within a struct are defined, these components's access type MUST override the access type of the struct. For example if a struct has an access type of read-write but has a component that is a read-only counter, the counter's access type MUST be read-only.

The access type for a component in a capability is always read-only per [RFC5812]. If an access type is provided for a component in a capability, the access type MUST be ignored. Similarly if an access

type is provided for a struct in a metadata the access type MUST be ignored.

This extension alters the definition of the struct in the xml schema from Figure 6 to Figure 7.

```
    <xsd:complexType name="structType">
     <xsd:sequence>
       <xsd:element name="derivedFrom" type="typeRefNMTOKEN"
         minOccurs="0"/>
       <xsd:element name="component" maxOccurs="unbounded">
         <xsd:complexType>
           <xsd:sequence>
             <xsd:element name="name" type="xsd:NMTOKEN"/>
             <xsd:element ref="synopsis"/>
             <xsd:element ref="description" minOccurs="0"/>
             <xsd:element name="optional" minOccurs="0"/>
             <xsd:group ref="typeDeclarationGroup"/>
           </xsd:sequence>
           <xsd:attribute name="componentID" type="xsd:unsignedInt"
            use="required"/>
         </xsd:complexType>
       </xsd:element>
     </xsd:sequence>
   </xsd:complexType>
```

Figure 6: Initial xml for the struct definition in the schema

```
       <xsd:complexType name="structType">
        <xsd:sequence>
          <xsd:element name="derivedFrom" type="typeRefNMTOKEN"
            minOccurs="0"/>
          <xsd:element name="component" maxOccurs="unbounded">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="name" type="xsd:NMTOKEN"/>
                <xsd:element ref="synopsis"/>
                <xsd:element ref="description" minOccurs="0"/>
                <xsd:element name="optional" minOccurs="0"/>
                <xsd:group ref="typeDeclarationGroup"/>
              </xsd:sequence>
              <!-- Extension RFC XXXX -->
              <xsd:attribute name="access" use="optional"
                default="read-write">
                <xsd:simpleType>
                  <xsd:list itemType="accessModeType"/>
                </xsd:simpleType>
              </xsd:attribute>
              <!-- /Extension RFC XXXX -->
              <xsd:attribute name="componentID" type="xsd:unsignedInt"
                use="required"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
```

Figure 7: New xml for the struct definition in the schema

An example of using optional access types for structs can be depicted in Figure 8

```
<component componentID="1" access="read-write">
   <name>PacketFlows</name>
   <synopsis>Packet Flows, match and counter</synopsis>
   <struct>
    <component componentID="1">
      <name>FlowMatch</name>
      <synopsis>Flow Match</synopsis>
      <typeRef>MatchType</typeRef>
    </component>
    <component componentID="2" access="read-only">
      <name>MatchCounter</name>
      <synopsis>Packets matching the flow match</synopsis>
      <typeRef>ZeroCounter</typeRef>
    </component>
  </struct>
</component>
```

Figure 8: Example of optional access types for struct

2.4. New Event Condition: BecomesEqualTo

This extensions adds one more event condition in the model schema, that of BecomesEqualTo. The difference between Greater Than and Less Than, is that when the value becomes exactly that of the BecomesEqualTo, the event is triggered. This event condition is particularly useful when there is a need to monitor one or more states of an LFB or the FE. For example in the CE High Availability (CEHA) [RFC7121] RFC it may be useful for the master CE to know which backup CEs have just become associated in order to connect to them and begin synchronizing the state of the FE. The master CE could always poll for such information but getting such an event will speed up the process and the event may be useful in other cases as well for

monitoring state.

The event MUST be triggered only when the value of the targeted component becomes equal to the event condition value. Implementations MUST NOT generate subsequent events while the targeted component's value remains equal to the event condition's value.

The BecomesEqualTo is appended to the schema as follows:

```
                    <xsd:element name="eventBecomesEqualTo"
                substitutionGroup="eventCondition"/>
```

Figure 9: New Excerpt of BecomesEqualTo event condition definition in the schema

It can become useful for the CE to be notified when the state has changed once the BecomesEqualTo event has been triggered, e.g. the CE may need to know when a backup CE has lost association. Such an event can be generated either by defining a second event on the same component, namely an Event Changed, or by simply reusing BecomesEqualTo and use event properties, in particular event hysteresis. We append the following definition for the event hysteresis defined in section 4.8.5.2 in [RFC5812], with V being the hysteresis value:

- For an <eventBecomesEqualTo/> condition, after the last notification a new <eventBecomesEqualTo/> notification MUST be generated only one time once the value has changed by +/- V.

For example using the value of 1 for V, will in effect create a singular event that will notify the CE that the value has changed by at least 1.

A developer of a CE should consider using count or time filtering to avoid being overrun by messages, e.g. in the case of rapid state changes.

2.5. LFB Properties

The previous model definition specifies properties for components of

LFBs. Experience has shown that, at least for debug reasons, it would be useful to have statistics per LFB instance to monitor sent and received messages and errors in communication between CE and FE. These properties are read-only.

In order to avoid ambiguity on protocol path semantics, this document specifies that the LFB component with ID 0 specifically MUST refer to LFB properties and ID 0 MUST NOT be used for any component definition. This disallowment is backwards compatible as no known LFB definition uses LFB component with ID 0. Any command with a path starting from LFB component 0 refers to LFB properties. The following change in the xml schema disallows usage of LFB component 0:

```
<xsd:attribute name="componentID" type="xsd:unsignedInt"
 use="required">
```

Figure 10: Initial xml for LFB Component IDs


```
<!-- Extension Added restriction to component ID -->
<xsd:attribute name="componentID" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:unsignedInt">
      <xsd:minExclusive value="0"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
<!-- End of extension -->
```

Figure 11: New xml to disallow usage of 0 as LFB Component

The following datatype definitions are to be used as properties for LFB instances.

```
<dataTypeDef>
   <name>LFBProperties</name>
   <synopsis>LFB Properties definition</synopsis>
```

```
            <struct>
               <component componentID="1">
                  <name>PacketsSentToCE</name>
                  <synopsis>Packets sent to CE</synopsis>
                  <typeRef>uint32</typeRef>
               </component>
               <component componentID="2">
                  <name>SentErrorPacketsToCE</name>
                  <synopsis>Error Packets sent to CE</synopsis>
                  <typeRef>uint32</typeRef>
               </component>
               <component componentID="3">
                  <name>BytesSentToCE</name>
                  <synopsis>Bytes sent to CE</synopsis>
                  <typeRef>uint32</typeRef>
               </component>
               <component componentID="4">
                  <name>SentErrorBytesToCE</name>
                  <synopsis>Error Bytes sent to CE</synopsis>
                  <typeRef>uint32</typeRef>
               </component>
               <component componentID="5">
                  <name>PacketsReceivedFromCE</name>
                  <synopsis>Packets received from CE</synopsis>
                  <typeRef>uint32</typeRef>
               </component>
               <component componentID="6">
                  <name>ReceivedErrorPacketsFromCE</name>
                  <synopsis>Error Packets received from CE</synopsis>
                  <typeRef>uint32</typeRef>
```



```
               </component>
               <component componentID="7">
                  <name>BytesReceivedFromCE</name>
                  <synopsis>Bytesreceived from CE</synopsis>
                  <typeRef>uint32</typeRef>
               </component>
               <component componentID="8">
                  <name>ReceivedErrorBytesFromCE</name>
                  <synopsis>Error Bytes received from CE</synopsis>
                  <typeRef>uint32</typeRef>
               </component>
```

```
      </struct>
    </dataTypeDef>
```

Properties for LFB instances

2.6. LFB class inheritance

The ForCES model [RFC5812] allows inheritance for LFB classes. However the xml schema defines only the LFB class from which an LFB class inherits. Recent implementations have identified an issue where ambiguity rises when different versions of the parent LFB class exists. This document augments the derivedFrom part of the LFB class definition with an optional version attribute when the derivedFrom field is used.

Having the version attribute as optional was a decision based on the need to maintain backwards compatibility with the XML schema defined in [RFC5812]. However if the version is omitted then the derivedFrom will always specify the first version of the parent LFB class, which usually is version 1.0.

This extension alters the definition of the derivedFrom in the xml schema from Figure 12 to Figure 13.

```
<xsd:element name="derivedFrom" minOccurs="0"/>
```

Figure 12: Initial xml for the LFB class inheritance


```
<xsd:element name="derivedFrom" minOccurs="0">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:NMTOKEN">
```

```
                        <xsd:attribute name="version"
                          type="versionType" use="optional"/>
                      </xsd:extension>
                    </xsd:simpleContent>
                  </xsd:complexType>
                </xsd:element>
```

Figure 13: New xml for the LFB class inheritance

An example of the use of the version attribute is given in Figure 14

```
<derivedFrom version="1.0">EtherPHYCop</derivedFrom>
```

Figure 14: Example of use of new xml for LFB class Inheritance

2.7. Enhancing XML Validation

As specified earlier this is not an extension but an enhancement of the schema to provide additional validation rules. This includes adding new key declarations to provide uniqueness as defined by the ForCES Model [RFC5812]. Such validations work only on within the same xml file.

This document introduces the following validation rules that did not exist in the original schema in [RFC5812]:

1. Each metadata ID must be unique.

2. LFB Class IDs must be unique.

3. Component ID, Capability ID and Event Base ID must be unique per LFB.

4. Event IDs must be unique per LFB.

5. Special Values in Atomic datatypes must be unique per atomic datatype.

3. XML Extension Schema for LFB Class Library Documents

This section includes the new XML Schema. Note that the namespace number has been updated from 1.0 to 1.1

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.1"
   xmlns:lfb="urn:ietf:params:xml:ns:forces:lfbmodel:1.1"
   targetNamespace="urn:ietf:params:xml:ns:forces:lfbmodel:1.1"
   elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         Schema for Defining LFB Classes and associated types
         (frames, data types for LFB attributes, and metadata).
      </xsd:documentation>
   </xsd:annotation>
   <xsd:element name="description" type="xsd:string"/>
   <xsd:element name="synopsis" type="xsd:string"/>
   <!-- Document root element: LFBLibrary -->
   <xsd:element name="LFBLibrary">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="description" minOccurs="0"/>
            <xsd:element name="load" type="loadType"
               minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="frameDefs" type="frameDefsType"
               minOccurs="0"/>
            <xsd:element name="dataTypeDefs" type="dataTypeDefsType"
               minOccurs="0"/>
            <xsd:element name="metadataDefs" type="metadataDefsType"
               minOccurs="0"/>
            <xsd:element name="LFBClassDefs" type="LFBClassDefsType"
               minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="provides" type="xsd:Name"
            use="required"/>
      </xsd:complexType>
      <!-- Uniqueness constraints -->
      <xsd:key name="frame">
         <xsd:selector xpath="lfb:frameDefs/lfb:frameDef"/>
         <xsd:field xpath="lfb:name"/>
      </xsd:key>
      <xsd:key name="dataType">
         <xsd:selector xpath="lfb:dataTypeDefs/lfb:dataTypeDef"/>
         <xsd:field xpath="lfb:name"/>
      </xsd:key>
      <xsd:key name="metadataDef">
         <xsd:selector xpath="lfb:metadataDefs/lfb:metadataDef"/>
         <xsd:field xpath="lfb:name"/>
      </xsd:key>
      <xsd:key name="metadataDefID">
         <xsd:selector xpath="lfb:metadataDefs/lfb:metadataDef"/>
```

```
         <xsd:field xpath="lfb:metadataID"/>
```



```
      </xsd:key>
      <xsd:key name="LFBClassDef">
         <xsd:selector xpath="lfb:LFBClassDefs/lfb:LFBClassDef"/>
         <xsd:field xpath="lfb:name"/>
      </xsd:key>
      <xsd:key name="LFBClassDefID">
         <xsd:selector xpath="lfb:LFBClassDefs/lfb:LFBClassDef"/>
         <xsd:field xpath="@LFBClassID"/>
      </xsd:key>
   </xsd:element>
   <xsd:complexType name="loadType">
      <xsd:attribute name="library" type="xsd:Name" use="required"/>
      <xsd:attribute name="location" type="xsd:anyURI"
         use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="frameDefsType">
      <xsd:sequence>
         <xsd:element name="frameDef" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="name" type="xsd:NMTOKEN"/>
                  <xsd:element ref="synopsis"/>
                  <xsd:element ref="description"
                     minOccurs="0"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="dataTypeDefsType">
      <xsd:sequence>
         <xsd:element name="dataTypeDef" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="name" type="xsd:NMTOKEN"/>
                  <xsd:element name="derivedFrom" type="xsd:NMTOKEN"
                     minOccurs="0"/>
                  <xsd:element ref="synopsis"/>
                  <xsd:element ref="description"
                     minOccurs="0"/>
```

```
                <xsd:group ref="typeDeclarationGroup"/>
                <!-- Extension RFC XXXX -->
                <xsd:element name="defaultValue" type="xsd:token"
                   minOccurs="0"/>
                <!-- /Extension RFC XXXX -->
             </xsd:sequence>
          </xsd:complexType>
       </xsd:element>
```



```
      </xsd:sequence>
   </xsd:complexType>
   <!-- Predefined (built-in) atomic data-types are: char, uchar,
   int16, uint16, int32, uint32, int64, uint64, string[N], string,
   byte[N], boolean, octetstring[N], float32, float64 -->
   <xsd:group name="typeDeclarationGroup">
      <xsd:choice>
         <xsd:element name="typeRef" type="typeRefNMTOKEN"/>
         <xsd:element name="atomic" type="atomicType"/>
         <xsd:element name="array" type="arrayType">
            <!-- Extension RFC XXXX -->
            <!--declare keys to have unique IDs -->
            <xsd:key name="contentKeyID">
               <xsd:selector xpath="lfb:contentKey"/>
               <xsd:field xpath="@contentKeyID"/>
            </xsd:key>
            <!-- /Extension RFC XXXX -->
         </xsd:element>
         <xsd:element name="struct" type="structType">
            <!-- Extension RFC XXXX -->
            <!-- key declaration to make componentIDs
              unique in a struct -->
            <xsd:key name="structComponentID">
               <xsd:selector xpath="lfb:component"/>
               <xsd:field xpath="@componentID"/>
            </xsd:key>
            <!-- /Extension RFC XXXX -->
         </xsd:element>
         <xsd:element name="union" type="structType"/>
         <xsd:element name="alias" type="typeRefNMTOKEN"/>
      </xsd:choice>
   </xsd:group>
   <xsd:simpleType name="typeRefNMTOKEN">
```

```
      <xsd:restriction base="xsd:token">
         <xsd:pattern value="\c+"/>
         <xsd:pattern value="string\[\d+\]"/>
         <xsd:pattern value="byte\[\d+\]"/>
         <xsd:pattern value="octetstring\[\d+\]"/>
      </xsd:restriction>
   </xsd:simpleType>
   <xsd:complexType name="atomicType">
      <xsd:sequence>
         <xsd:element name="baseType" type="typeRefNMTOKEN"/>
         <xsd:element name="rangeRestriction"
           type="rangeRestrictionType" minOccurs="0"/>
         <xsd:element name="specialValues" type="specialValuesType"
            minOccurs="0">
            <!-- Extension RFC XXXX -->
```



```
            <xsd:key name="SpecialValue">
               <xsd:selector xpath="specialValue"/>
               <xsd:field xpath="@value"/>
            </xsd:key>
            <!-- /Extension RFC XXXX -->
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="rangeRestrictionType">
      <xsd:sequence>
         <xsd:element name="allowedRange" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:attribute name="min" type="xsd:integer"
                  use="required"/>
               <xsd:attribute name="max" type="xsd:integer"
                  use="required"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="specialValuesType">
      <xsd:sequence>
         <xsd:element name="specialValue" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="name" type="xsd:NMTOKEN"/>
```

```
                  <xsd:element ref="synopsis"/>
               </xsd:sequence>
               <xsd:attribute name="value" type="xsd:token"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="arrayType">
      <xsd:sequence>
         <xsd:group ref="typeDeclarationGroup"/>
         <xsd:element name="contentKey" minOccurs="0"
            maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="contentKeyField"
                     type="xsd:string" maxOccurs="unbounded"/>
               </xsd:sequence>
               <xsd:attribute name="contentKeyID" type="xsd:integer"
                  use="required"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
```



```
      <xsd:attribute name="type" use="optional" default="variable-size">
         <xsd:simpleType>
            <xsd:restriction base="xsd:string">
               <xsd:enumeration value="fixed-size"/>
               <xsd:enumeration value="variable-size"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="length" type="xsd:integer"
         use="optional"/>
      <xsd:attribute name="maxLength" type="xsd:integer"
         use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="structType">
      <xsd:sequence>
         <xsd:element name="derivedFrom" type="typeRefNMTOKEN"
            minOccurs="0"/>
         <xsd:element name="component" maxOccurs="unbounded">
            <xsd:complexType>
```

```
                <xsd:sequence>
                   <xsd:element name="name" type="xsd:NMTOKEN"/>
                   <xsd:element ref="synopsis"/>
                   <xsd:element ref="description"
                      minOccurs="0"/>
                   <xsd:element name="optional" minOccurs="0"/>
                   <xsd:group ref="typeDeclarationGroup"/>
                </xsd:sequence>
                <!-- Extension RFC XXXX -->
                <xsd:attribute name="access" use="optional"
                   default="read-write">
                   <xsd:simpleType>
                      <xsd:list itemType="accessModeType"/>
                   </xsd:simpleType>
                </xsd:attribute>
                <!-- Extension RFC XXXX -->
                <xsd:attribute name="componentID" type="xsd:unsignedInt"
                   use="required"/>
             </xsd:complexType>
          </xsd:element>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="metadataDefsType">
       <xsd:sequence>
          <xsd:element name="metadataDef" maxOccurs="unbounded">
             <xsd:complexType>
                <xsd:sequence>
                   <xsd:element name="name" type="xsd:NMTOKEN"/>
                   <xsd:element ref="synopsis"/>
```



```
                   <xsd:element name="metadataID" type="xsd:integer"/>
                   <xsd:element ref="description"
                      minOccurs="0"/>
                   <xsd:choice>
                      <xsd:element name="typeRef" type="typeRefNMTOKEN"/>
                      <xsd:element name="atomic" type="atomicType"/>
                      <!-- Extension RFC XXXX -->
                      <xsd:element name="array" type="arrayType">
                         <!--declare keys to have unique IDs -->
                         <xsd:key name="contentKeyID1">
                            <xsd:selector xpath="lfb:contentKey"/>
                            <xsd:field xpath="@contentKeyID"/>
```

```
                        </xsd:key>
                        <!-- /Extension RFC XXXX -->
                     </xsd:element>
                     <xsd:element name="struct" type="structType">
                        <!-- Extension RFC XXXX -->
                        <!-- key declaration to make componentIDs
                           unique in a struct -->
                        <xsd:key name="structComponentID1">
                           <xsd:selector xpath="lfb:component"/>
                           <xsd:field xpath="@componentID"/>
                        </xsd:key>
                        <!-- /Extension RFC XXXX -->
                     </xsd:element>
                  </xsd:choice>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="LFBClassDefsType">
      <xsd:sequence>
         <xsd:element name="LFBClassDef" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="name" type="xsd:NMTOKEN"/>
                  <xsd:element ref="synopsis"/>
                  <xsd:element name="version" type="versionType"/>
                  <xsd:element name="derivedFrom" minOccurs="0">
                    <xsd:complexType>
                      <xsd:simpleContent>
                        <xsd:extension base="xsd:NMTOKEN">
                          <xsd:attribute name="version"
                            type="versionType" use="optional"/>
                        </xsd:extension>
                      </xsd:simpleContent>
                    </xsd:complexType>
```



```
                  </xsd:element>
                  <xsd:element name="inputPorts"
                   type="inputPortsType" minOccurs="0"/>
                  <xsd:element name="outputPorts"
                   type="outputPortsType" minOccurs="0"/>
```

```
            <xsd:element name="components"
             type="LFBComponentsType" minOccurs="0"/>
            <xsd:element name="capabilities"
             type="LFBCapabilitiesType" minOccurs="0"/>
            <xsd:element name="events" type="eventsType"
               minOccurs="0"/>
            <xsd:element ref="description"
               minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="LFBClassID" type="xsd:unsignedInt"
            use="required"/>
      </xsd:complexType>
      <!-- Key constraint to ensure unique attribute names
      within a class: -->
      <xsd:key name="components">
         <xsd:selector xpath="lfb:components/lfb:component"/>
         <xsd:field xpath="lfb:name"/>
      </xsd:key>
      <xsd:key name="capabilities">
         <xsd:selector xpath="lfb:capabilities/lfb:capability"/>
         <xsd:field xpath="lfb:name"/>
      </xsd:key>
      <xsd:key name="events">
         <xsd:selector xpath="lfb:events/lfb:event"/>
         <xsd:field xpath="lfb:name"/>
      </xsd:key>
      <xsd:key name="eventsIDs">
         <xsd:selector xpath="lfb:events/lfb:event"/>
         <xsd:field xpath="@eventID"/>
      </xsd:key>
      <xsd:key name="componentIDs">
         <xsd:selector xpath="lfb:components/lfb:component"/>
         <xsd:field xpath="@componentID"/>
      </xsd:key>
      <xsd:key name="capabilityIDs">
         <xsd:selector xpath="lfb:capabilities/lfb:capability"/>
         <xsd:field xpath="@componentID"/>
      </xsd:key>
      <xsd:key name="ComponentCapabilityComponentIDUniqueness">
         <xsd:selector
            xpath="lfb:components/lfb:component|
            lfb:capabilities/lfb:capability|lfb:events"/>
         <xsd:field xpath="@componentID|@baseID"/>
```

```
         </xsd:key>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="versionType">
   <xsd:restriction base="xsd:NMTOKEN">
      <xsd:pattern value="[1-9][0-9]*\.([1-9][0-9]*|0)"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="inputPortsType">
   <xsd:sequence>
      <xsd:element name="inputPort" type="inputPortType"
         maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="inputPortType">
   <xsd:sequence>
      <xsd:element name="name" type="xsd:NMTOKEN"/>
      <xsd:element ref="synopsis"/>
      <xsd:element name="expectation" type="portExpectationType"/>
      <xsd:element ref="description" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="group" type="xsd:boolean"
      use="optional" default="0"/>
</xsd:complexType>
<xsd:complexType name="portExpectationType">
   <xsd:sequence>
      <xsd:element name="frameExpected" minOccurs="0">
         <xsd:complexType>
            <xsd:sequence>
               <!-- ref must refer to a name of a defined
                 frame type -->
               <xsd:element name="ref" type="xsd:string"
                  maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="metadataExpected" minOccurs="0">
         <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
               <!--ref must refer to a name of a defined metadata-->
               <xsd:element name="ref" type="metadataInputRefType"/>
               <xsd:element name="one-of"
                  type="metadataInputChoiceType"/>
            </xsd:choice>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
```

```
   </xsd:complexType>
   <xsd:complexType name="metadataInputChoiceType">
      <xsd:choice minOccurs="2" maxOccurs="unbounded">
         <!-- ref must refer to a name of a defined metadata -->
         <xsd:element name="ref" type="xsd:NMTOKEN"/>
         <xsd:element name="one-of" type="metadataInputChoiceType"/>
         <xsd:element name="metadataSet" type="metadataInputSetType"/>
      </xsd:choice>
   </xsd:complexType>
   <xsd:complexType name="metadataInputSetType">
      <xsd:choice minOccurs="2" maxOccurs="unbounded">
         <!-- ref must refer to a name of a defined metadata -->
         <xsd:element name="ref" type="metadataInputRefType"/>
         <xsd:element name="one-of" type="metadataInputChoiceType"/>
      </xsd:choice>
   </xsd:complexType>
   <xsd:complexType name="metadataInputRefType">
      <xsd:simpleContent>
         <xsd:extension base="xsd:NMTOKEN">
            <xsd:attribute name="dependency" use="optional"
               default="required">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                     <xsd:enumeration value="required"/>
                     <xsd:enumeration value="optional"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="defaultValue" type="xsd:token"
               use="optional"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="outputPortsType">
      <xsd:sequence>
         <xsd:element name="outputPort" type="outputPortType"
            maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="outputPortType">
      <xsd:sequence>
         <xsd:element name="name" type="xsd:NMTOKEN"/>
```

```
         <xsd:element ref="synopsis"/>
         <xsd:element name="product" type="portProductType"/>
         <xsd:element ref="description" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="group" type="xsd:boolean"
         use="optional" default="0"/>
```



```
   </xsd:complexType>
   <xsd:complexType name="portProductType">
      <xsd:sequence>
         <xsd:element name="frameProduced" minOccurs="0">
            <xsd:complexType>
               <xsd:sequence>
                  <!-- ref must refer to a name of a defined
                     frame type -->
                  <xsd:element name="ref" type="xsd:NMTOKEN"
                     maxOccurs="unbounded"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="metadataProduced" minOccurs="0">
            <xsd:complexType>
               <xsd:choice maxOccurs="unbounded">
                  <!-- ref must refer to a name of a
                     defined metadata -->
                  <xsd:element name="ref"
                     type="metadataOutputRefType"/>
                  <xsd:element name="one-of"
                     type="metadataOutputChoiceType"/>
               </xsd:choice>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="metadataOutputChoiceType">
      <xsd:choice minOccurs="2" maxOccurs="unbounded">
         <!-- ref must refer to a name of a defined metadata -->
         <xsd:element name="ref" type="xsd:NMTOKEN"/>
         <xsd:element name="one-of" type="metadataOutputChoiceType"/>
         <xsd:element name="metadataSet" type="metadataOutputSetType"/>
      </xsd:choice>
   </xsd:complexType>
```

```
   <xsd:complexType name="metadataOutputSetType">
      <xsd:choice minOccurs="2" maxOccurs="unbounded">
         <!-- ref must refer to a name of a defined metadata -->
         <xsd:element name="ref" type="metadataOutputRefType"/>
         <xsd:element name="one-of" type="metadataOutputChoiceType"/>
      </xsd:choice>
   </xsd:complexType>
   <xsd:complexType name="metadataOutputRefType">
      <xsd:simpleContent>
         <xsd:extension base="xsd:NMTOKEN">
            <xsd:attribute name="availability" use="optional"
               default="unconditional">
               <xsd:simpleType>
```



```
                  <xsd:restriction base="xsd:string">
                     <xsd:enumeration value="unconditional"/>
                     <xsd:enumeration value="conditional"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:attribute>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="LFBComponentsType">
      <xsd:sequence>
         <xsd:element name="component" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="name" type="xsd:NMTOKEN"/>
                  <xsd:element ref="synopsis"/>
                  <xsd:element ref="description"
                     minOccurs="0"/>
                  <xsd:element name="optional" minOccurs="0"/>
                  <xsd:group ref="typeDeclarationGroup"/>
                  <xsd:element name="defaultValue" type="xsd:token"
                     minOccurs="0"/>
               </xsd:sequence>
               <xsd:attribute name="access" use="optional"
                  default="read-write">
                  <xsd:simpleType>
                     <xsd:list itemType="accessModeType"/>
                  </xsd:simpleType>
```

```
            </xsd:attribute>
            <!-- Extension Added restriction to component ID -->
            <xsd:attribute name="componentID" use="required">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:unsignedInt">
                     <xsd:minExclusive value="0"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:attribute>
            <!-- End of extension -->
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="accessModeType">
   <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="read-only"/>
      <xsd:enumeration value="read-write"/>
      <xsd:enumeration value="write-only"/>
      <xsd:enumeration value="read-reset"/>
```



```
      <xsd:enumeration value="trigger-only"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="LFBCapabilitiesType">
   <xsd:sequence>
      <xsd:element name="capability" maxOccurs="unbounded">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="name" type="xsd:NMTOKEN"/>
               <xsd:element ref="synopsis"/>
               <xsd:element ref="description"
                  minOccurs="0"/>
               <xsd:element name="optional" minOccurs="0"/>
               <xsd:group ref="typeDeclarationGroup"/>
            </xsd:sequence>
            <xsd:attribute name="componentID" type="xsd:integer"
               use="required"/>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
```

```
   <xsd:complexType name="eventsType">
      <xsd:sequence>
         <xsd:element name="event" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="name" type="xsd:NMTOKEN"/>
                  <xsd:element ref="synopsis"/>
                  <xsd:element name="eventTarget"
                    type="eventPathType"/>
                  <xsd:element ref="eventCondition"/>
                  <xsd:element name="eventReports"
                   type="eventReportsType" minOccurs="0"/>
                  <xsd:element ref="description"
                     minOccurs="0"/>
               </xsd:sequence>
               <xsd:attribute name="eventID" type="xsd:integer"
                  use="required"/>
            </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="baseID" type="xsd:integer"
         use="optional"/>
   </xsd:complexType>
   <!-- the substitution group for the event conditions -->
   <xsd:element name="eventCondition" abstract="true"/>
   <xsd:element name="eventCreated"
    substitutionGroup="eventCondition"/>
```



```
   <xsd:element name="eventDeleted"
    substitutionGroup="eventCondition"/>
   <xsd:element name="eventChanged"
    substitutionGroup="eventCondition"/>
   <xsd:element name="eventGreaterThan"
    substitutionGroup="eventCondition"/>
   <xsd:element name="eventLessThan"
    substitutionGroup="eventCondition"/>
   <!-- Extension RFC XXXX -->
   <xsd:element name="eventBecomesEqualTo"
      substitutionGroup="eventCondition"/>
   <!-- /Extension RFC XXXX -->
   <xsd:complexType name="eventPathType">
      <xsd:sequence>
```

```
         <xsd:element ref="eventPathPart" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <!-- the substitution group for the event path parts -->
   <xsd:element name="eventPathPart" type="xsd:string"
      abstract="true"/>
   <xsd:element name="eventField" type="xsd:string"
      substitutionGroup="eventPathPart"/>
   <xsd:element name="eventSubscript" type="xsd:string"
      substitutionGroup="eventPathPart"/>
   <xsd:complexType name="eventReportsType">
      <xsd:sequence>
         <xsd:element name="eventReport" type="eventPathType"
            maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:simpleType name="booleanType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="0"/>
         <xsd:enumeration value="1"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

ForCES LFB XML Schema

4. Acknowledgements

The author would like to acknowledge Joel Halpern, Jamal Hadi Salim and Dave Hood for their comments and discussion that helped shape this document in a better way. Special acknowledgements to Joel Halpern for resolving the issue with the default values. Adrian Farrel for his AD review, Ben Campbell for his Gen-ART review and Tom Yu for his security review, all of which improved the quality of this

document. Additionally the following members of the IESG Stephen Farrel, Barry Leiba and Ted Lemon for their reviews and comments that shaped the final version of this document.

5. IANA Considerations

IANA has registered a new XML namespace, as per the guidelines in RFC

3688 [RFC3688].

URI: The URI for this namespace is

urn:ietf:params:xml:ns:forces:lfbmodel:1.1

Registrant Contact: IESG

XML: none, this is an XML namespace

6. Security Considerations

This specification adds only a few constructs to the initial model defined in [RFC5812], namely namely a new event, some new properties and a way to define optional access types and complex metadata. In addition this document addresses and clarifies an issue with the inheritance model by introducing the version of the derivedFrom LFB class. These constructs and the inheritance model change do not change the nature of the initial model.

Thus the security considerations defined in [RFC5812] apply to this specification as well, as the changes proposed here are simply constructs to write XML library definitions, as in [RFC5812]. These changes, as well as the clarification of the inheritance issue of the initial model, have no effect on the security semantics of the protocol.

In regards to pervasive monitoring (PM), as discussed in [RFC7258], this specification defines ways to expose more complex information, namely metadata, exchanged between LFBs and between CEs and FEs and a new event. These changes have very little or no effect in terms of making PM simpler or more effective to an attacker who controls the LFBs. The new metadata might make for slightly more elegant PM, but does not enable any really new way to (ab)use LFBs for PM. The same applies for the new event.

Finally, this document does not provide any protocol specification and as such does not specifies how information will be transmitted between respective entities, thus PM mitigation for a passive attacker that simply wants to eavesdrop on the information exchanged is out of scope.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC5810] Doria, A., Hadi Salim, J., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, March 2010.

[RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010.

[RFC7121] Ogawa, K., Wang, W., Haleplidis, E., and J. Hadi Salim, "High Availability within a Forwarding and Control Element Separation (ForCES) Network Element", RFC 7121, February 2014.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, May 2014.

7.2. Informative References

[OpenFlowSpec1.1] http://www.OpenFlow.org/, "The OpenFlow 1.1 Specification.", <http://www.OpenFlow.org/documents/OpenFlow-spec-v1.1.0.pdf>.

Author's Address

Evangelos Haleplidis
University of Patras
Department of Electrical and Computer Engineering
Patras 26500
Greece

Email: ehalep@ece.upatras.gr