Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: March 29, 2014

E. Haleplidis
University of Patras

September 25, 2013

ForCES Model Extension
draft-ietf-forces-model-extension-00

Abstract

Forwarding and Control Element Separation (ForCES) defines an architectural framework and associated protocols to standardize information exchange between the control plane and the forwarding plane in a ForCES Network Element (ForCES NE). RFC5812 has defined the ForCES Model provides a formal way to represent the capabilities, state, and configuration of forwarding elements within the context of the ForCES protocol, so that control elements (CEs) can control the FEs accordingly. More specifically, the model describes the logical functions that are present in an FE, what capabilities these functions support, and how these functions are or can be interconnected.

RFC5812 has been around for two years and experience in its use has shown room for small extensions without a need to alter the protocol while retaining backward compatibility with older xml libraries. This document extends the model to allow complex datatypes for metadata, optional default values for datatypes and optional access types for structures. The document also introduces three new features, bitmap as a new datatype, a new event condition BecomesEqualTo and LFB properties.

Status of This Memo

Table of Contents

1. Terminology and Conventions

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 follows the terminology defined by the ForCES Model in [RFC5812]. The required definitions are repeated below for clarity.

FE Model - The FE model is designed to model the logical processing functions of an FE. The FE model proposed in this document includes three components; the LFB modeling of individual Logical Functional Block (LFB model), the logical interconnection between LFBs (LFB topology), and the FE-level attributes, including FE capabilities. The FE model provides the basis to define the information elements exchanged between the CE and the FE in the ForCES protocol [RFC5810].

LFB (Logical Functional Block) Class (or type) - A template that represents a fine-grained, logically separable aspect of FE processing. Most LFBs relate to packet processing in the data path. LFB classes are the basic building blocks of the FE model.

LFB Instance - As a packet flows through an FE along a data path, it flows through one or multiple LFB instances, where each LFB is an instance of a specific LFB class. Multiple instances of the same LFB class can be present in an FE's data path. Note that we often refer to LFBs without distinguishing between an LFB class and LFB instance when we believe the implied reference is obvious for the given context.

LFB Model - The LFB model describes the content and structures in an LFB, plus the associated data definition. XML is used to provide a formal definition of the necessary structures for the modeling. Four types of information are defined in the LFB model. The core part of the LFB model is the LFB class definitions; the other three types of information define constructs associated with and used by the class definition. These are reusable data types, supported frame (packet) formats, and metadata.

Element - Element is generally used in this document in accordance with the XML usage of the term. It refers to an XML tagged part

of an XML document. For a precise definition, please see the full set of XML specifications from the W3C. This term is included in this list for completeness because the ForCES formal model uses XML.

Attribute - Attribute is used in the ForCES formal modeling in accordance with standard XML usage of the term, i.e., to provide attribute information included in an XML tag.

LFB Metadata - Metadata is used to communicate per-packet state from one LFB to another, but is not sent across the network. The FE model defines how such metadata is identified, produced, and

consumed by the LFBs, but not how the per-packet state is implemented within actual hardware. Metadata is sent between the FE and the CE on redirect packets.

ForCES Component - A ForCES Component is a well-defined, uniquely identifiable and addressable ForCES model building block. A component has a 32-bit ID, name, type, and an optional synopsis description. These are often referred to simply as components. LFB Component - An LFB component is a ForCES component that defines the Operational parameters of the LFBs that must be visible to the CEs.

LFB Class Library - The LFB class library is a set of LFB classes that has been identified as the most common functions found in most FEs and hence should be defined first by the ForCES Working Group.

2. Introduction

The ForCES Model [RFC5812] presents a formal way to define FEs Logical Function Blocks (LFBs) using 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 extends the ForCES Model to allow complex datatypes for metadata, optional default values for datatypes and optional access types for structures. Additionally the document introduces three new features, bitmap as a new datatype, a new event

condition BecomesEqualTo and LFB properties.

These extensions are an addendum 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.

XXX: Discussion is needed to specify whether bitmap required protocol definition of how bitmap is sent through the wire.

3. ForCES Model Extension proposal

3.1. 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 ony typeRef and atomic are allowed for a metadata definition.

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

1. The Action Set metadata follows a packet inside the Flow Tables. The Action Set metadata is an array of actions to be performed at the end of the pipeline.

2. When a packet is received from a controller it may be accompanied by a list of actions to be performed on it prior to be sent on the flow table pipeline which 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"/>
            </xsd:choice>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
```

Figure 1: Initial MetadataDefType Defintion in the schema

```
  <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: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>
          </xsd:choice>
```

```
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Figure 2: New MetadataDefType Defintion for the schema

3.2. 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 optionally to add default values to atomic and typeref types, whether they are as simple or complex datatypes. 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.

This extension alters the definition of the typeDeclarationGroup in the xml schema from Figure 3 to Figure 4 to allow default values to TypeRef.

```
<xsd:element name="typeRef" type="typeRefNMTOKEN"/>
```

Figure 3: Initial Excerpt of typeDeclarationGroup Defintion in the schema


```
    <xsd:sequence>
<xsd:element name="typeRef" type="typeRefNMTOKEN"/>
  <xsd:element name="DefaultValue" type="xsd:token"
       minOccurs="0"/>
    </xsd:sequence>
```

Figure 4: New Excerpt of typeDeclarationGroup Defintion in the schema

Additionally it appends to the declaration of the AtomicType this xml (Figure 5) to allow default values to Atomic datatypes.

```
<xsd:element name="defaultValue" type="xsd:token" minOccurs="0"/>
```

Figure 5: Appending xml in of AtomicType Defintion in the schema

3.3. Optional Access Type for Structs

In the original schema, the access type can be only be defined on components of LFB and not on components in structs or arrays. However when it's a struct datatype it is not possible to fine-tune access type per component in 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.

With this extension is it allowed to define the access type for a struct component either in the datatype definitions or in the LFB component definitions.

When the optional access type for a struct component is defined it MUST override the access type of the struct. If by accident an access type for a component in a capability is defined, the access type MUST NOT be taken into account and MUST always be considered as read-only.

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>
        <xsd:attribute name="access" use="optional"
          default="read-write">
          <xsd:simpleType>
            <xsd:list itemType="accessModeType"/>
          </xsd:simpleType>
        </xsd:attribute>
        <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

3.4. New datatype: Bitmap

With the current schema it is valid to create a struct of booleans in order to simulate a bitmap value. However each boolean is sent as 4bytes. This extension adds the bitmap, a set of sequential named bits.

Bitmaps may be useful in describing capabilities, e.g. Link speed capabilities as multiple boolean values.

EDITOR: Discussion may be required as to whether there is a need for protocol description of how the bitmap is sent through the wire.

In the new schema, bits are named followed an optional bit value. An example:

```
 <dataTypeDef>
   <name>BitmapExample</name>
   <synopsis>A bitmap field example</synopsis>
   <bitmap size="5">
     <bit name="Bit1" position="1"/>
     <bit name="Bit3" position="3"/>
   </bitmap>
 </dataTypeDef>
```

Figure 8: Example of bitmap Defintion

The ordering of the bits MUST be implemented in the order that are defined in the xml library. Positions MUST range from 0 and reach bitmap size-1.

If the size of the bitmap is not defined, then the size of the bitmap MUST be the maximum position value of all the bit values, e.g. 4

The bitmap is defined in the model extension schema as follows:

```
<xsd:complexType name="bitmapType">
  <xsd:sequence>
    <xsd:element name="bit" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="position" type="xsd:NMTOKEN"
         use="required"/>
        <xsd:attribute name="name" type="xsd:NMTOKEN"
         use="required"/>
        <xsd:attribute name="defaultValue" type="booleanValues"
         use="optional"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
    <xsd:attribute name="size" type="xsd:integer" use="optional"/>
</xsd:complexType>
<xsd:simpleType name="booleanValues">
  <xsd:restriction base="xsd:integer">
```

```
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="1"/>
```



```
    </xsd:restriction>
  </xsd:simpleType>
```

Figure 9: New Excerpt of bitmap Defintion in the schema

Along with the needed addition to the typeDeclarationGroup Definition:

```
   <xsd:element name="bitmap" type="bitmapType"/>
```

Figure 10: New Excerpt of typeDeclarationGroup Defintion in the schema

3.5. 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 is exactly that of the BecomesEqualTo, the event is triggered. This event condition is particular useful when there is a need to monitor one or more states of an LFB or the FE. For example in the CEHA [I-D.ietf-forces-ceha] document 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 and MUST NOT generate 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 11: 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 must also take into account to use count or time filtering to avoid being overun by messages, e.g. in the case of rapid state changes.

3.6. LFB Properties

The current model definition specifies properties for components of LFBs. Experience however has proven valuable at least for debug reasons, to have statistics per LFB instance to monitor sent/received messages and errors for communication between CE and FE. These properties are read-only.

Editorial: Need additional discussion whether we need these LFB packets. Additional definition may be required to handle protocol

messages for LFB properties

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

3.7. 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 deinfed by the ForCES Model [RFC5812]. Such validations work only on within the same xml file.

The following validation rules have been appended 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.

4. XML Extension Schema for LFB Class Library Documents

```
<?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:1.0"
   xmlns:lfb="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
   targetNamespace="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
   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" />
               </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>
         <!-- Extension -->
         <xsd:sequence>
            <!-- /Extension -->
            <xsd:element name="typeRef" type="typeRefNMTOKEN" />
            <!-- Extension -->
            <xsd:element name="DefaultValue" type="xsd:token"
               minOccurs="0" />
         </xsd:sequence>
         <xsd:element name="bitmap" type="bitmapType"/>
         <!-- /Extension -->
         <xsd:element name="atomic" type="atomicType" />
         <xsd:element name="array" type="arrayType">
            <!-- Extension -->
            <!--declare keys to have unique IDs -->
            <xsd:key name="contentKeyID">
               <xsd:selector xpath="lfb:contentKey" />
               <xsd:field xpath="@contentKeyID" />
            </xsd:key>
            <!-- /Extension -->
         </xsd:element>
         <xsd:element name="struct" type="structType">
            <!-- Extension -->
            <!-- key for componentIDs uniqueness in a struct -->
            <xsd:key name="structComponentID">
               <xsd:selector xpath="lfb:component" />
               <xsd:field xpath="@componentID" />
            </xsd:key>
```

```
            <!-- /Extension -->
          </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 -->
             <xsd:key name="SpecialValue">
                <xsd:selector xpath="specialValue" />
                <xsd:field xpath="@value" />
             </xsd:key>
             <!-- /Extension -->
          </xsd:element>
          <!-- Extension -->
          <xsd:element name="defaultValue" type="xsd:token"
             minOccurs="0" />
          <!-- /Extension -->
       </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>
    <!-- Extension -->
    <xsd:complexType name="bitmapType">
      <xsd:sequence>
        <xsd:element name="bit" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:attribute name="position" type="xsd:NMTOKEN"
             use="required"/>
            <xsd:attribute name="name" type="xsd:NMTOKEN"
             use="required"/>
            <xsd:attribute name="defaultValue" type="booleanValues"
             use="optional"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
        <xsd:attribute name="size" type="xsd:integer" use="optional"/>
    </xsd:complexType>
    <xsd:simpleType name="booleanValues">
      <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="0"/>
        <xsd:maxInclusive value="1"/>
      </xsd:restriction>
    </xsd:simpleType>
    <!-- /Extension -->
     <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 -->
                   <xsd:attribute name="access" use="optional"
                    default="read-write">
                     <xsd:simpleType>
```

```
                    <xsd:list itemType="accessModeType"/>
                  </xsd:simpleType>
                </xsd:attribute>
                <!-- /Extension -->
               <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 -->
                     <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 -->
                     </xsd:element>
                     <xsd:element name="struct" type="structType">
                        <!-- Extension -->
                        <!-- 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 -->
                     </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" type="xsd:NMTOKEN"
                     minOccurs="0" />
                  <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>
                  <xsd:attribute name="componentID"
                    type="xsd:unsignedInt" use="required" />
               </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 -->
     <xsd:element name="eventBecomesEqualTo"
      substitutionGroup="eventCondition"/>
   <!-- /Extension -->
    <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>
```

OpenFlow XML Library

5. Acknowledgements

The author would like to acknowledge Joel Halpern, Jamal Hadi and Dave Hood for their comments and discussion that helped shape this document in a better way.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

The security considerations that have been described in the ForCES Model RFC [RFC5812] apply to this document as well.

8. References

8.1. Normative References

[I-D.ietf-forces-ceha]
Ogawa, K., Wang, W., Haleplidis, E., and J. Salim, "ForCES Intra-NE High Availability", draft-ietf-forces-ceha-07 (work in progress), May 2013.

[OpenFlowSpec1.1]
http://www.OpenFlow.org/, "The OpenFlow 1.1 Specification.", , <http://www.OpenFlow.org/documents/

OpenFlow-spec-v1.1.0.pdf>.

[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.

8.2. Informative References

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

Author's Address

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

Email: ehalep@ece.upatras.gr