EmulationScript Schema Description

Abstract

This document describes an XML schema being developed to support generation and manipulation of "planning" and "scripting" documents and files used in support mobile network modeling. The schema provides for documents that can be used to orchestrate mobile network modeling using emulation environments such as the Naval Research Laboratory (NRL) "extensible Mobile Ad-hoc Network Emulator" (EMANE) framework or using network simulation tools such as *ns-3* The initial focus of this development is on planning, scripting, and generating mobile node location and motion properties for wireless network emulations and simulations. Some initial tools are available from <http://cs.itd.nrl.navy.mil/products> to create and manipulate files in the XML formats described here. Additional schema and related tools are planned for future development.

1. Introduction 2
2. Common XML Structures 4
2.1. Data Types 5
2.1.1. <time> Types 5
2.1.2. <location> Types 5
2.1.3. <motion> Types 6
3. *EmulationScript* Documents 8
3.1. Top-Level Elements 10
3.1.1. *EmulationScript* Document 10
3.1.2. <Event> Element 10
3.2. Event Modules 10
3.2.1. <Node> Module 10
3.2.2. TBD - Additional Scriptable Module Definitions 15
4. Planning Document Types 15
4.1. Common Planning Elements 16
4.1.1. <Node> Principal Element 16
4.1.2. <mark> and <wait> Primitive Element 17
4.2. *MotionPlan* Documents 18
4.2.1. Motion Primitives 19
4.2.2. Motion Pattern Definition 24
4.2.3. Random Waypoint Generator Definition 25
4.2.4. Node *MotionPlan* Specification 25
4.3. *NetworkPlan* Documents 26
4.3.1. <NetworkDefinition> Element 28
4.3.2. *NetworkPlan* <Node> Elements 29
4.3.3. NetworkPlan Example 31
5. *EmulationDirectory* Document 32
5.1. Common EmulationDirectory Elements 33
5.2. EMANE-Specific *EmulationDirectory* Elements 34
5.2.1. <emaneHost> Element 34
5.2.2. <emanePlatform> Element 34
5.2.3. <emaneNEM> Element 34

6. Modeling System Template Documents 34
6.1. *EmaneTemplate* Document Type 34
6.1.1. <HostPoolDefinition>, <PortPoolDefinition>, and <DevicePoolDefinition> Elements 35
6.1.2. PlatformTemplate Element 36
6.1.3. TransportTemplate Sub-Element 37
6.1.4. NemTemplate 39
6.1.5. Example EmaneTemplate Documents 40
7. Ancillary File Formats 42
7.1. Emulation Event Log (EEL) Format 42
7.1.1. <time> Field 43
7.1.2. <moduleID> Field 43
7.1.3. "location" Events 44
7.1.4. Propagation "pathLoss" Events 45
7.1.5. Module "address" Events 45
7.1.6. Module "param" Events 46
7.2. NRL Scripted Display Tool (SDT) Format 47
7.3. MITRE Mobility Format (MMF) 47
8. Example Utilities 48
9. Usage Notes 49
10. "ToDo" List 49
10.1. Comments and Questions 49

1. Introduction

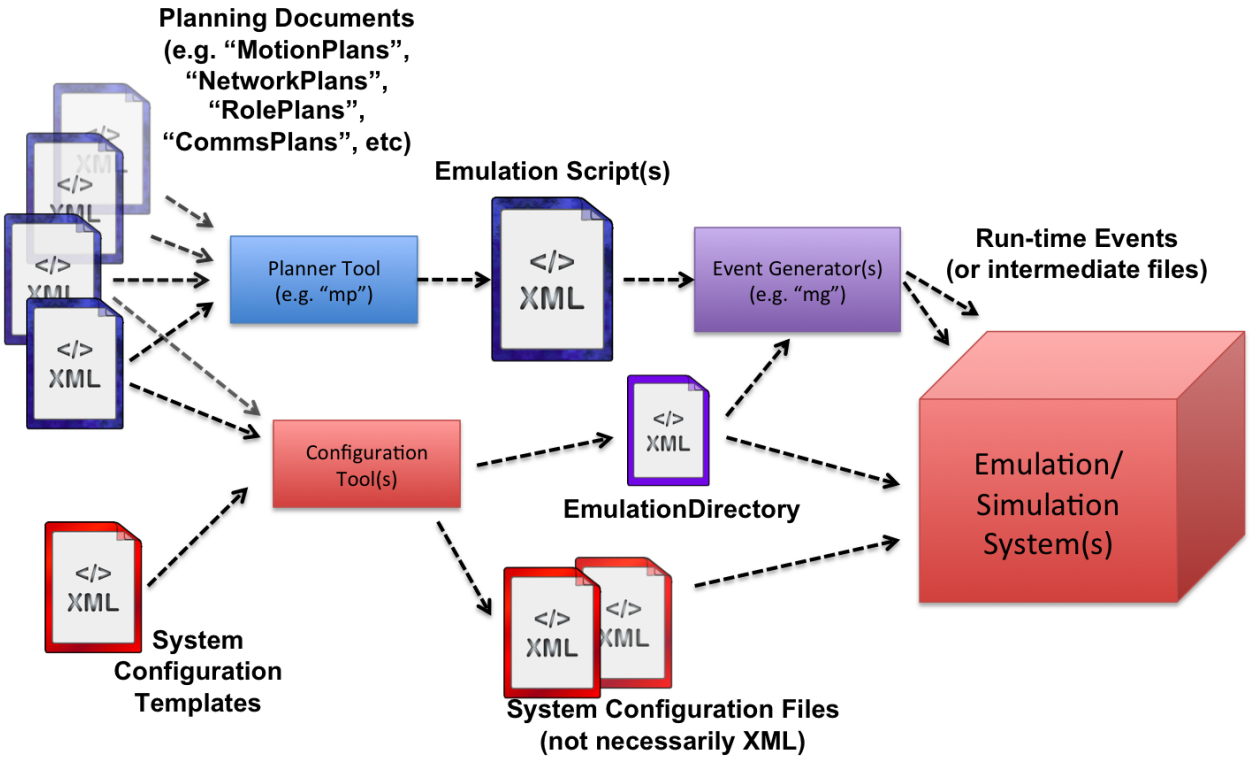
This document describes an accompanying eXtensible Markup Language (XML) Schema (“EmulationScriptSchema.xsd” document) specifying XML document types that can be used for scripting Mobile Network Emulation run-time events. More specifically, this design is provided to support operation of the open-source, extensible Mobile Network Emulation (EMANE) system developed by the Naval Research Laboratory (NRL) and CENGEN, Inc. However, the schema is intended to be sufficiently general that tools can be developed to generate configuration files and/or scripts for other mobile network modeling emulation or simulation systems (e.g. *ns-3*, OPNET, CORE, etc).

The schema supports elements to define certain mobile “node” characteristics (e.g. location, velocity, orientation, etc) and parameters of other functional modules and changes to those characteristics or parameters over the course of mobile network modeling experiment execution. The schema adopts a modular approach to the definition of mobile network modeling scenarios. A couple of categories of document types comprise this modular approach: "Planning Documents" and "Emulation Script Documents". While the “*EmulationScript*” schema specifies a document structure to contain time-ordered “Events” that tightly script (dynamically update) emulation module properties (e.g. Node location/motion and other), the "planning" documents allow aspects of the scenario to be preplanned in a more tenable, natural fashion. For example, a "*MotionPlan*" document type is supported that lets a user describe the intended motion plans for mobile nodes in the scenario in a somewhat natural fashion (e.g, start at a certain location, move to first waypoint, then another, etc). A "*MotionPlan*" document may describe the plans for multiple mobile "nodes" or multiple "*MotionPlan*" documents can be created for a set of nodes. Then, corresponding planning tools may concatenate these "plans" into a consolidated, scripted scenario. This approach allows variations of scenarios to be constructed with planning documents as "building blocks" in scenario construction. Note that the planning schema documents even allow inter-dependencies to be expressed (e.g. The motion of one node may be dependent upon the motion of another node or other planned scenario events). The *mp* tool mentioned in Figure 1, “Emulation Planning and Scripting Work Flow” is included in the source code distribution and provides a means for generating a time-ordered "*EmulationScript*" describing node mobility from one or more "MotionPlan" document. The toolset that includes *mp* will be expanded over time to include additional features and incorporate additional "planning" document types such as "*NetworkPlan*" and "*CommunicationPlan*" types that convey network system and node configuration and communication events, respectively.

Figure 1, “Emulation Planning and Scripting Work Flow” illustrates this modular approach of "planning" documents that can be created to separately describe different aspects of the scenario and "script" documents that are the resulting

composite of the separate "planning" components. Additionally, system-specific template and configuration files may be incorporated into the workflow to realize, in an automated fashion, the planned experimentation scenario in a specific modeling (emulation or simulation) system.

Figure 1. Emulation Planning and Scripting Work Flow



The "blue" portions of Figure 1, “Emulation Planning and Scripting Work Flow” correspond to the components of the planning and experiment orchestration process that can be somewhat independent of the target mobile network modeling system. The "red" portions are those components that will need to have specific knowledge of the target system. The template file provides an input to the system configuration process that can be used to control parameters of how the specific modeling system is to be set up or used. For example, in a distributed emulation system, this may correspond to information that identifies or identifies the computing resources to be used. Note the generation of the system configuration file(s) is a synthesis of the "generic" planning documents provided and this template. The "*EmulationDirectory*" document provides a mapping of the generic scenario elements (e.g., "Nodes" and their "interfaces", etc) to the instantiations of emulation system components (e.g., EMANE Network Emulation Modules (NEMs), ns-2 simulation "agents", etc) that represent those elements. The "*EmulationDirectory*" document then provides a reference to map scripted scenario configuration and "events" to specific run-time events or corresponding commands for the specific modeling system(s) in use. For example, one such mapping would the resolution of "*NetworkPlan*" "Node:interface" names to the specific NEM identifier when EMANE is used.

It is expected that different configuration tool sets can be created for different target modeling systems. Similarly, run-time control events are generated as a synthesis of the "scripts" that results from processing of planning documents *and* the specific system configuration that was created. The "purple" coloring of the "*EmulationDirectory*" document and "Event Generator" functional module illustrates their incorporation of both the "blue" generic scenario description and "red" system-specific configuration information. For example, in the case the EMANE system the system configuration files can provide the information needed to "map" from the general-purpose naming of "Nodes" and/or their network interfaces, etc in the emulation "script" to the numeric identifiers that correspond to the specific Network Emulation Module (NEM) instances representing those nodes and/or interfaces.

An XML schema is defined here to enable the use of existing and emerging XML content manipulation tools to perform operations on scripts. This includes filtering of events within scripts, potential merging of scripts, and manipulations that might parametrically alter a script. However, note the XML format described here is not intended to be a complete replacement for other simpler file formats that might be used to describe mobile node location and/or motion. In fact, this schema is purposefully designed to allow for conversion to and from such simpler file formats as needed and such alternative formats are described in this document.. For example, in some cases such as node location and motion, much more compact (as compared to XML) representations can be realized and may be more practical for some purposes due to file sizing and other factors. Similarly, this schema is expected to be capable of representing the content of several existing mobility script formats.

So, additionally, this document will specify alternate compact textual representations of some of the XML elements the described scheme introduces. While the “Script Schema” described here will be able to contain (and provide context for via XML tags) multiple script event types, and in fact will be able to evolve over time as new event types are determined, the simpler text formats will generally contain events for some particular subtext (e.g. node mobility). This ability to provide translation compatibility with other formats can make this schema useful for specifying and/or scripting scenarios outside of strictly EMANE or emulation purposes, including use with tools for discrete event network simulation (e.g. ns-2, OPNET, etc) or for analytical programs.

The first iteration of this document (and accompanying schema) primarily focuses on describing elements that can represent mobile node location and optionally velocity (or even richer motion patterns). However, the structure of the schema attempts to be sufficiently general to contain XML elements allow future inclusion of events that can initialize or update characteristics of any modules within the emulation system. This generalized capability will allow users to construct scripts that can describe alterations to arbitrary aspects of the emulation over time (provided that the EMANE or other run-time framework supports interpretation of these events and control of those aspects). Currently "*NetworkPlan*" and "*CommunicationPlan*" document types are envisioned in addition to the "*MotionPlan*" document type that is currently defined. Additionally, some additional document types are define to support EMANE-based operation. These include an "*EmaneTemplate*" document type that can contain a description of a specific EMANE system configuration that a tool chain may reference to automate the generation of EMANE configuration files. Additionally, a general-purpose "Emulation Event Log" (EEL) file format is defined as an intermediate file format for containing emulation events (e.g. <location> and/or "path loss" events) generated prior to emulation run-time. This log format may also be used to capture run-time generated events to enable analyses or "re-play" of scenarios that contain non-deterministic behaviors.

The development of this Mobile Network Modeling scenario planning concept and initial toolset was part of the development of the EMANE framework. However, the EMANE system is sufficiently modular that other scripting or control techniques could be applied to its operation, depending upon the design of the specific Network Emulation Modules (NEMs) and other components loaded into an instantiation of the EMANE system. The document formats and tools described here are intended to provide a default standard approach to be used for EMANE experimentation. It is expected that many of these documents can be the product of to-be-defined scenario generation programs or other tools, but of course, may be manually created as well. And, as mentioned above, it is possible that the schema described here can be used to generate configuration files and scripts for other emulation or simulation environments besides EMANE such as CORE, WISER, OPNET, ns-2/3, Qualnet, etc.

2. Common XML Structures

Some common XML structures are used in the *EmulationScript* and the various planning documents. These currently include types defined to represent time, location, and motion. These data type structures are embedded within different document content types (e.g. *EmulationScript* <Event> module properties, planning document primitives, etc). The "EmulationScriptSchema.xsd" document that accompanies this distribution provides formal W3C schema definitions of these data types as well as formal specification of many of the document types described here.

2.1. Data Types

This section describes some data types that are defined globally in the “EmulationScript” namespace. These data types include formats to describe some fundamental values that are used throughout the emulation system (e.g. “time”, <location>, etc).

2.1.1. <time> Types

The <Event:time> element is a complex type that may represent either a relative or absolute instant of time (Note: future versions of this specification may possibly limit the Event:time to be a relative time only). The “format” attribute of the Event:time element indicates the how the element’s text content should be interpreted. Possible <Event:time> text formats include:

1. “secs” indicating an floating point value of relative time in seconds.

2. “chron” indicating a relative chronograph in the “xs:duration” built-in format of "P[nY][nM][nD][T[nH][nM][n[.nnn]S]]" where: P denotes a period (required), nY indicates 'n' years, nM indicates 'n' months, nD indicates 'n' days, T indicates the start of a time section (required if you are going to specify hours, minutes, or seconds) nH indicates 'n' hours, nM indicates 'n' minutes, and n[.nnn]S indicates 'n.nnn' seconds.

3. “clock” indicating an absolute clock time in the “xs:time” built-in format of “HH:MM:SS[.sss]”. The respective 'hours' (HH), 'minutes' (MM), and 'seconds' (SS.[.sss] fields MUST be specified as 2-digit integers (leading zeroes as required) although the 'seconds' field can be extended with a decimal point and additional digits to describe a floating point value.

4. “dateTime” indicating an absolute date and time in the “xs:dateTime” built-in format of "YYYY-MM-DDThh:mm:ss[.sss]" where: 'YYYY' indicates the year, 'MM' indicates the month, and 'DD' indicates the day. T indicates the start of the required time section where 'hh' indicates the hour, 'mm' indicates the minute, and 'ss[.sss]' indicates the second (a decimal point and digits may extend the 'ss' field to a float value). Note: All components are required! To specify a time zone, you can either enter a dateTime in UTC time by adding a "Z" at the end, or you can specify an offset from the UTC time by adding a positive or negative ({+|-}) "hh::mm" quantity at the end (e.g., "2008-09-21T21:43:00+05:00" for "21 Sept, 2008 9:43 PM EST").

Note the <EmulationScript:startTime> element is similar to the <Event:time> but its text content is restricted to the absolute time formats of either “clock” or “dateTime”. It also has a “type” attribute to indicate the text content format.

(NOTE: At this time, the example tools provided with this distribution implement only the "secs" format).

2.1.2. <location> Types

The emulation script schema validates coordinates within the script to be a comma-delimited set of at least two and optionally three floating point values (i.e. “a,b[,c]”) with a fourth optional "altitude-interpretation" keyword specifier. The schema defines a “locationType” complex type that has a “type” attribute that indicates the geometry of the given coordinates. The current two possible geometry types are:

1. “gps” indicating the coordinates represent a geographic location of “latitude, longitude[,altitude[,{agl | msl}]] where ‘altitude’ is in units of meters, and

2. “cartesian” indicating the coordinates represent an “x,y[,z[,{agl | msl}]]” location.

Note that the ‘altitude’ and ‘z’ are the optional third value. When this third value is omitted, the altitude or z-axis value is implicitly zero. When the 'altitude' or 'z' value is given, a optional fourth comma-delimited keyword of "agl" (above-ground-level) or "msl" (mean-sea-level) MAY be given to provide the intended interpretation of the specified altitude

for systems that have terrain data available. It is RECOMMENDED that the default interpretation of altitude (or z-axis) be above-ground-level when no specific "agl" or "msl" keyword is provided.

It is also RECOMMENDED that a consistent geometry type be used within a given *EmulationScript* instance. However, future versions of this specification may provide a element that allows one type of geometry to be mapped to another.

This “locationType” is used in both *MotionPlan* and *EmulationScript* documents including use as the <Node:location> property element (described later) and to indicate locations as required for embedded elements describing motion (e.g. *EmulationScript* <Event:Node:motion> and *MotionPlan* motion primitive elements).

The format of “locationType” elements is

```
<location type=”gps”><lat>,<lon>[,<alt>[,{agl | msl}]]</location>
```

or

```
<location type=”cartesian”><x>,<y>[,<z>[,{agl | msl}]]</location>
```

Note the <alt> and <z> ordinates are optional in the “gps” and “cartesian” coordinate types, respectively. And the "agl" or "msl" specification is additionally optional when the <alt> or <z> is given.

2.1.3. <motion> Types

The schema currently specifies an element form that is used to describe the property of "motion" (i.e. movement in space) within an *EmulationScript* or *MotionPlan* document. For example, the “Node” module (described later) can have a <motion> property associated with it. The <motion> element form consists of two portions:

1. A <location> element indicating the current location of the applicable object, and

2. One of a set of different motion type sub-elements.

Note the mandatory inclusion of the current location as part of the motion specification is intentional. The reasons for this include the fact that the <location> given is implicitly the starting point for the specified motion and that this simplifies the process for run-time emulation controllers to “jog-shuttle” (i.e. rewind/fast-forward) to specific points in time as needed. The inclusion of this <location> element in the motion specification requires that *EmulationScript* writers or generation tools need to provide a current <location> that is consistent with prior <location> and <motion> events to evoke smooth patterns of motion. However, it is possible that “instant teleportation” can be scripted if desired as well.

A few different basic motion types are currently described in the schema. In the future, additional types will be added and this document will be updated to describe these added types. The sections below provide and overview of the motion types currently supported. The *MotionPlan* document type has corresponding "motion primitive" types of similar form specified for these <motion> sub-element types. Additional details on these can be found in Section 4.2.1, “Motion Primitives”. Note that it is NOT expected that *EmulationScript* <motion> events will be hand-scripted, but instead would be automatically generated by a tool that interprets a *MotionPlan* document. The *mp* tool included in the accompanying source code distribution is an example of such a tool.

The following XML excerpt provides some example instances of the motion type structure within <Node:motion> Events in an *EmulationScript* document:

```
<EmulationScript>
  <Event>
    <time>0.000000</time>
    <Node name="node01">
      <motion>
        <location>38.800000,-76.950000</location>
        <waypoint>
```

```
          <destination>38.820000,-77.000000</destination>
          <velocity>10.000000</velocity>
        </waypoint>
      </motion>
    </Node>
    <Node name="node02">
      <location>38.820000,-76.950000</location>
    </Node>
    <Node name="node03">
      <location>38.840000,-76.950000</location>
    </Node>
  </Event>
  <Event>
    <time>10.000000</time>
    <Node name="node02">
      <motion>
        <location>38.820000,-76.950000</location>
        <waypoint>
          <destination>38.820000,-77.000000</destination>
          <velocity>10.000000</velocity>
        </waypoint>
      </motion>
    </Node>
  </Event>
</EmulationScript>
```

2.1.3.1. <waypoint> Motion Type

The <waypoint> motion type is specified with a <destination> location and the <velocity> at which the "Node" moves there. The intended interpretation of this motion type is that the node will move directly towards the "destination" at the given "velocity" until it either reaches the destination or its motion is superseded with a different motion (or fixed location) property in the script (i.e. motion is halted when the waypoint destination is reached). Note that the requirement that <motion> elements contain a current <location> location means that it is possible that Nodes may instantly "teleport" to a new location if the location value given in a new <location> or <motion> update is inconsistent with any prior specified motion or location.

2.1.3.2. <vector> Motion Type

The <vector> motion type is specified with "azimuth" and optionally "elevation" angle(s) indicating the direction (or "bearing") of movement and the "velocity" at which the "Node" moves there. Like the <waypoint> type, motion in a straight line (using "great circle" when GPS coordinates are used) is thus defined, but with no specified final destination or halting point. The motion continues indefinitely until it is superseded by another motion (or <location>) update. The <circle> motion type parameter sub-elements include:

1. "velocity" of motion (default units of meters per second) ,

2. "azimuth" (i.e. bearing) angle in units of degrees (±360°). This is relative to due North in GPS coordinates or to the y-axis in Cartesian coordinates, and

3. "elevation" angle (optional) in units of degrees (±90°). This indicates the rate of change in altitude (or along the z-axis) as motion occurs. Note that at plus (or minus) 90°, the motion velocity is 100% applied towards altitude (height) change and not motion in the given bearing direction occurs! If the optional "elevation" element is not include, a value of 0° (no altitude changeis assumed.

2.1.3.3. <circle> Motion Type

The <circle> motion type describes a circular path about a center location with a fixed radius and altitude (if applicable) at a fixed velocity. The <circle> motion type parameter elements include:

1. "center" location coordinates with optional altitude/height which, if omitted, implies the motion follows any applicable terrain. (Note that the "center" element is optional for MotionPlan documents and that the circle center will be set to the node's current location in the motion plan and the motion will spiral outwards to the circle perimeter at a constant angular velocity based on the objective circle "radius" and "velocity"),

2. "radius" of the circle in units of meters, and

3. "velocity of motion (default units of meters per second) where a positive "velocity" implies a clockwise direction (as viewed from a higher altitude looking downard) and a negative "velocity" is used to specify a counter-clockwise direction.

The intended interpretation of this motion type is that the node will follow the circular path continuously at the given "velocity" until its motion is superseded with a different motion (or fixed location) property in the script. The required "motion:location" element accompanying the <circle> motion specification MUST be a point that is consistent (valid) with respect to the specified circle and serves as the starting point of the circular motion.

2.1.3.4. <loiter> Motion Type

The <loiter> motion type describes a circular path of fixed radius and relative height (if applicable) about a location that is moving according to another "reference" motion type. The <loiter> motion type parameter elements include:

1. An embedded reference motion specification that consists of one of the motion primitives described here. For "MotionPlan" documents, the "reference motion" may actually be a <pattern> which was previously defined using the "PatternDefinition" element.

2. The "radius" of the loiter circle in units of meters.

3. The "velocity of the loiter motion (default units of meters per second) where a positive "velocity" implies a clockwise direction (as viewed from a higher altitude looking downard) and a negative "velocity" is used to specify a counter-clockwise direction.

4. An optional "height" (in meters by default) that is an altitude (z-axis) offset from the reference motion current location.

The intended interpretation of this motion type is that the node will follow the circular ploiter ath continuously at the given loiter "velocity" until its motion is superseded with a different motion (or fixed location) property in the script. The required reference motion element accompanying the <loiter> motion specification provides the moving "center" location for the loiter circle that is conducted. Note that the "reference motion" may also be a fixed <location> element, and is thus equivalent to a circle motion definition in this case.

3. *EmulationScript* Documents

To support this goal of generalized event scripting, the top-level hierarchy of the Emulation Script Schema described here consists of the "*EmulationScript*" document that principally contains a list of abstract "Event" elements. Each "Event" contains a "time" value element corresponding to the time at which the event should be executed and one or more sub-elements that identify the emulation "Module" instance(s) (each instance is identified by an "name" attribute string) and any properties that are being set or altered (i.e. the Event action(s) to be taken). These sub-elements are specified in a hierarchical fashion so that possibly multiple or complex characteristics and/or sub-attributes can be described as needed. The following pseudo-script outlines this hierarchy:

```
<EmulationScript>
   <Event>
      <time> time1 </time>
      <Module1 name=”moduleA”>
         <property1> value </>
         <property2> value </>
```

```
         …
      </Module1>
      <Module1 name=”moduleB”>
         <property1> value </>
         <property2> value </>
         …
      </Module1>
      <Module2 name=”moduleZ”>
         <propertyX> value </>
         <propertyY> value </>
         …
      </Module2>
   </Event>
   <Event>
      <time> time2</time>
      <Module1 name=”moduleA”>
         <property1> value </>
         <property2> value </>
         …
      </Module1>
   </Event>
   …
</EmulationScript>
```

A specific example of this hierarchy is scripting of mobile “`Node`” mobility properties within an EmulationScript document. In this case, the “`Node`” corresponds to the “Module” being addressed with specific node instances being identified by their identifier (name as indicated by the required “`id`” attribute). The current version of the .xsd schema file accompanying this document specifies the “`Event:Node`” type and some properties related to scripting node location and/or motion. The simple example given here scripts some interface parameters and location of two nodes (“node01” and “node02”) with initial interface parameters and locations given for time “0.0” (seconds) and updated locations at time “1.0”:

```
<EmulationScript>
   <Event>
      <time> 0.0 </time>
      <Node name=”node01”>
         <location>38.123,-78.524,100</>
         <interface name="wifi0">
             <frequency units="GHz">2.4</frequency>
             <power units="Watts">6.0</power>
         </interface>
      </Node>
      <Node name=”node02”>
         <location>38.232,-78.254,800 </>
         <interface name="wifi0">
             <frequency units="GHz">2.4</frequency>
             <power units="Watts">6.0</power>
         </interface>
      </Node>
   </Event>
   <Event>
      <time> 1.0 </time>
      <Node name=”node01”>
         <location>38.124,-78.525,100</>
      </Node>
      <Node name=”node02”>
         <location>38.233,-78.255,800 </>
      </Node>
   </Event>
   …
</EmulationScript>
```

This section describes the top-level *EmulationScript*" document and <Event> elements, describes data types that are used throughout the document (i.e. by multiple sub-element types), and then provides sections describing the various sub-element types (Modules and their properties). This document and its associated schema will continue to be updated as this script format evolves.

3.1. Top-Level Elements

The top-level elements that comprise an emulation script are the *EmulationScript* document container itself and the <Event> element type. The *EmulationScript* essentially consists of a time-ordered list of "Events".

3.1.1. *EmulationScript* Document

An emulation script document is encapsulated with the <EmulationScript> tag. The content of the document is a list of <Event> elements as described below. The list of script Events MUST be sorted by order of their <time> value.

The <EmulationScript> also can contain an optional <startTime> element that specifies an absolute time (possibly including date) that corresponds to the script's "time zero" starting time. This element, if included, MUST be the first element within the <EmulationScript> document.

3.1.2. <Event> Element

The <Event> element is used to contain a list of emulation modules (with their associated properties) that are to be set or updated at a given time. Each <Event> corresponds to a specific instance of time, and the <time> element of the <Event> indicates this. Note that the <time> element is a complex type that may express either an absolute time (using XML date/time conventions) or a relative time in units of seconds. The "relative" time is reference to the _start_ of the emulation where the start is time 00:00:00 (0.0 seconds) The format of the <time> element is further described in the "Global Data Types" section below.

3.2. Event Modules

The targets of the contents of the <Event> element are the various "modules" that comprise the emulation system. These include "modules" that represent the modeled (emulated or simulated) components of the systems such as "Nodes" and their associated configuration items (e.g., software processes such as network routing or application daemons) or environmental characteristics (e.g., geographic location and motion properties). These also may include "modules" that are control components of the emulation system itself.

Module element instances and, in some cases, their sub-elements are uniquely identified by a "name" attribute value. The colon character ':' is RESERVED as a delimiter for hierarchical concatenation of Module and associated sub-element names and MUST NOT be used in "name" values. The use of spaces in names is also discouraged but is allowed, if necessary. To support this, the double-quote character "" is RESERVED and MUST NOT be used in names. Implementations MAY use colon and double-quote characters in strings representing this hierarchical module and sub-element naming system.

These "modules" (e.g. Nodes, etc) MAY also be referenced in the Planning Documents described in Section 4, "Planning Document Types". Note there are some attributes define for these "module" elements and sub-elements that are actually relevant to Planning Document use and not germaine for *EmulationScript* processing.

3.2.1. <Node> Module

The <Node> module element is defined to identify and set properties for distinct, typically physically, entities that are being virtually represented within the mobile network modeling system. Examples of "Nodes" may include mobile or fixed platforms such as vehicles, buildings, people, devices (e.g. robots), etc. A "Node" may correspond to a single intermediate (e.g. router) or end (e.g. host) system within the possibly mobile network, but complex emulation scenarios may also include Nodes that contain multiple network *subsystems* as well as network interfaces. These emulated systems

and interfaces are anchored to the Node location as they are a physical component of the Node itself (Provisions for relative location are planned for this schema). A notional taxonomy of <Node:host:interface> is defined here to model that aspect of Node configuration where the <host> element identifies Node subsystems that possess their own network <interface> instances. The planning documents and other document types described here often invoke the <Node> element and this taxonomy of <Node:host:interface> structure.

The <Node> module element has a single REQUIRED attribute specified, its "name" attribute. This attribute declares the existence of a <Node> with the given "name" value upon its first occurrence in processing or references an existing <Node> declaration upon re-occurrence. Thus <Node> "name" values MUST be globally-unique within the context of a network modeling scenario.

The <Node> module MAY OPTIONALLY contain <host> sub-element instances to reference one or more computing host systems associated with the Node. These <host> elements can have <interface> sub-elements that correspond to network interface devices. Alternatively, the "shorthand" notation of attaching an <interface> sub-element *directly* to a <Node> is allowed with an implicit <host name="default"> element pre-defined for each <Node> instance. Thus, the following two *EmulationScript* <Event:Node> instantiations are equivalent (Note that these <Event> examples do not actually adjust any Node or interface parameters or attributes, but simply illustrate the instantiation of the "Node:host:interface" taxonomy within an *EmulationScript* <Event>):

```
<EmulationScript>
   <Event>
      <time> 0.0 </time>
      <Node name=”car1”>
         <host name="default">
            <interface name="wlan0"/>
            <interface name="eth0"/>
         </host>
         <host name="laptop">
            <interface name="eth0"/>
         </host>
      </Node>
   </Event>
</EmulationScript>
```

```
<EmulationScript>
   <Event>
      <time> 0.0 </time>
      <Node name=”car1”>
         <interface name="wlan0"/>
         <interface name="eth0"/>
         <host name="laptop">
            <interface name="eth0"/>
         </host>
      </Node>
   </Event>
</EmulationScript>
```

Both of these "scripts" reference a "Node" with two computing "hosts" and a total of 3 network "interfaces". The interfaces are identified (explicitly in the first case, and implicitly in the second case) through the "Node:host:interface" taxonomy as:

1. car1:default:wlan0

2. car1:default:eth0

3. car1:laptop:eth0

Details on the <host> and <interface> sub-elements that can be associated with <Node> module instances are desribed in the following sections. Note that use of the <Node> element definition and its sub-elements also apply to

the Planning Documents described in Section 4, “Planning Document Types” as well as the *EmulationScript* document type presented here.

3.2.1.1. Node <host> Sub-element

The <host> child element can be included within a <Node> structure when there are multiple network subsystems associated with a Node. As previously mentioned an implicit <host> with a "name" value of default is assumed such that <interface> sub-elements MAY be direct children of a <Node> element without the need for an intermediate <host> declaration. The <host> abstraction provides a means to express groups of network systems (end or intermediate systems) that share common Node properties, most notably physical location for mobile networks. This allows simplification of scenario planning (e.g. MotionPlan creation) aspects that are independent of network-related items.

The <host> element simply has a single "name" attribute that allows the logical "host" to identified within the context of its parent <Node>. Note the "name" value default is RESERVED for a <host> namespace that is implicitly associated with <Node> instances, although it is permitted to explicity use the default keyword to explicitly specify this implicit <host>, if desired.

3.2.1.2. Node/host <interface> Sub-element

The <interface> sub-element MAY be a direct descendant of a <Node> parent or a child of a <host> sub-element associated with a <Node>. In either case, the <interface> element has the following attributes defined:

1. A REQUIRED "name" attribute that identifies the interface uniquely in the context of its parent <Node> or <host>

2. An OPTIONAL "type" attribute that identifies, by name, the type of interface device represented.

3. An OPTIONAL "net" attribute that identifies, by name, a defined logical "network" (or subnetwork) to which the interface is attached

4. An OPTIONAL "assign" attribute that can dictate address assignment modality for the interface. Possible "assign" values include:

default	An address should be assigned at configuration-time from the <NetworkDefinition> address space, if defined.
autoconf	An address is to be assigned to the interface at run-time via auto-configuration.
none	An address should not be assigned to this interface.

The "type" and "net" attributes are principally used in the *NetworkPlan* document type described in Section 4.3, “*NetworkPlan* Documents”. In those documents, the <interface> "type" may be directly specified or inferred when a "net" is reference that has a pre-defined default interface type specified. Thus, at least one of these attributes is generally needed when declaring the set of interfaces associated with a <Node> or <host> in a *NetworkPlan* document.

For purposes of network modeling, parameters of "Node:host:interface" are often set and possibly updated during the course of a scenario. Some common interface parameters such as address configuration can be specified. And to support the goals of mobile, wireless network modeling, some parameters somewhat common to wireless interfaces are defined here. For example, wireless radio frequency (RF) transmission power and frequency are parameters that affect communication range and hence connectivity for wireless networks. Not all parameters will apply to all interface types. Different interface types will often have different parameters and features. An additional general-purpose <param> sub-element convention is defined to allow parameter specifications in addition to the "common" parameters defined here to be opaquely conveyed through processes that manipulate the XML documents this schema describes.

The following sections describe the currently-defined parameter sub-element types. Additional property or parameter sub-element types will be defined and documented in the future.

3.2.1.2.1. Interface <address> Parameter

A <Node:host:interface> instance MAY have one or more OPTIONAL <address>sub-elements that associate addresses with the given networ interface. In some scenarios, addresses may be pre-assigned during experiment configuration while, in other cases, auto-configuration protocols or other actions that execute during the experiment may assign addresses during run-time. The <address> element is defined to convey interface address associations for *EmulationScript* documents, *NetworkPlan* documents, or other planning documents related to modeled network operations.

The text content of the <address> element contains a string representation of an address. A special <address> text value of "none" is RESERVED to indicate that all addresses of a given "type" should be removed from the <interface> address association list. The following attributes can be used to specify how the address is to be associated with the given <interface>:

type Indicates the address type which may also imply the format of the element's text content. If the "type" is not given, it is assumed that the type can be automatically determined from the format (e.g. IPv4 dotted decimal or IPv6 colon-delimited). This may not be possible for all address formats and use of the "type" attribute is RECOMMENDED and may even be REQUIRED in future versions of this specification. For the special case of the address text value "none", all addresses (of all types) are removed from the interface address list if the "type" attribute is omitted. Valid types include:

- ipv4 The address text corresponds to an IPv4 address in dotted-decimal. The address portion MAY also be appended with a "slash" character '/' and a numeric value to indicate a prefix mask length in bits.
- ipv6 The address text corresponds to an IPv6 address in the usual colon-delimited formats. The address portion MAY also be appended with a "slash" character '/' and a numeric value to indicate a prefix mask length in bits.
- mac The address text corrsponds to a layer-2 Media Access Control (MAC) address. This will usually be a colon-delimited set of 6 hexadecimal values (48-bit) to convey a IEEE 802 address, but may also include other formats such as the similar EUI-64 format, if applicable, or simply a decimal value.

cmd An OPTIONAL "command" that can be used to specify whether the given address is to be added or removed to/from the interface address list, or to replace the current address(es) for the given "type". If the "cmd" attribute is omitted, a default command value of replace is to be assumed. Valid "cmd" values and their interpretations include:

- add The given address is to be added to interface's list of associated addresses for the corresponding "type" value. I.e., individual lists are maintained for each address "type". Note that if the address text value is "none", no action is taken for this "cmd" value.
- remove The given address is to removed from the interface's list of associated addresses for the corresponding "type" value. Note that if the address text value is "none", no action is taken for this "cmd" value.
- replace The interface's list of addresses for the corresponding "type" value is to be replaced with the given address. Thus, a single address may replace an entire list of multiple addresses. If the address text value is "none", *all* addresses for the given "type" are dissassociated from the interface. And if the "type" is NOT specified and the text value is "none" all address of all types are disassociated from the interface (i.e., *all* address association lists are emptied). The replace action is the default, assumed action when no "cmd" attribute is given.

The following *NetworkPlan* XML excerpt illustrates the assignment of a single IPv4 address to an interface of the "default" host of a Node named "car1":

```
<Node name="car1">
    <interface name="wifi0" type="WIFI">
        <address type="ipv4" cmd="replace">192.168.1.1</address>
    </interface>
</Node>
```

3.2.1.2.2. Interface Transmission <power> Parameter

Transmission power is generally a defining aspect of wireless interface network connectivity and thus a <Node:host:interface> sub-element tagged <power> is defined to set the transmission power level for the given interface. For example, the <power> sub-element may be embedded in an *EmulationScript* <Event> for a <Node> to set or update the power during run-time or it may be included as a <Node:interface> parameter in a *NetworkPlan* document.

The <power> element, like most other parameter elements, has a "units" attribute that may be set to a value of dBm, Watts, milliwatts, or mw to indicate the unit type of the floating point numerical value contained in the <power> elements text content. Tools that process documents from this schema SHOULD accept the full set of unit types listed here.

The following XML excerpt illustrates the assignment of a transmit power to the "WiFi" interface of the "default" host of a Node named "car1":

```
<Node name="car1">
    <interface name="wifi0">
        <power units="Watts">1.0</power>
    </interface>
</Node>
```

3.2.1.2.3. Interface Radio <frequency> Parameter

Similar to transmission power, the transmission radio frequency is a determinant of wireless communication connectivity. Typically, lower frequencies propagate a longer distance than higher frequencies and this information is needed in modeling systems to compute connectivity. The <frequency> sub-element is provided to denote the transmission frequency for an interface.

Similar to the other parameter elements, the <frequency> element has a REQUIRED "units" attribute that can be used to convey the units of the floating point value contained in the elements text content. Valid <frequency> "units" values include MHz, GHz, KHz, and Hz. Tools that process documents from this schema SHOULD accept the full set of unit types listed here.

By default, the interface's receive frequency is assumed to be the same as the transmit frequency. However, a "mode" attribute is provided so that the transmit and receive frequencies may be set independently. The "mode" attribute may have a value of tx or rx to convey the aspect (transmit or receive, respectively) to which the frequency value applies. When the OPTIONAL "mode" attribute is not included, the <frequency> value is assumed to apply to both the transmit and receive aspects of the interface.

The following provides some examples of frequency sub-element usage:

```
<Node name="car1">
    <interface name="wifi0">
        <frequency units="GHz">2.4</frequency>
    </interface>
    <interface name="radio0">
        <frequency mode="tx" units="GHz">3.5</frequency>
        <frequency mode="rx" units="GHz">3.6</frequency>
    </interface>
</Node>
```

3.2.1.2.4. Generic <param> Parameters

The <param> sub-element provides a "pass-through" mechanism for embedding parameters into a *NetworkPlan* document (or into *EmulationScript* <Event> content) that are not fully-defined by this schema. This allows valid extension of the capabilities described here without explicit schema modification. However, it is hoped that the "library" of fully-defined parameters will grow as this framework evolves and that this <param> mechanism is not overly used.

The <param> element has three attributes:

1. REQUIRED "name" to provide a text identifier to the parameter

2. OPTIONAL "units" to provide annotation of the unit type of the named parameter value

3. REQUIRED "value" to convey a numeric or string value for the parameter

A <param> "name" MUST be different than any of the fully-defined parameter elements (e.g. power, frequency, rate, etc).

The following illustrates an example use of the <param> element:

```
<Node name="car1">
    <interface name="wifi0">
        <param name="polarity" value="vertical"\>
    </interface>
</Node>
```

3.2.2. TBD - Additional Scriptable Module Definitions

(TBD – enumerate/describe additional "Modules" of the emulation system that may be dynamically scripted/controlled during operation)

4. Planning Document Types

A family of "planning" documents is defined to provide a means to modularly, and naturally describe different aspects of a mobile network scenarios. These aspects include node mobility, network device assignments, etc. For the most part, planning documents are somewhat independent of each other. However, since the goal of the planning documents is to provide a basis for a script of orchestrated "Events" that define an experimentation scenario, there are some common element types that are shared among the family of planning documents described here. And furthermore, many of the element types are reflective of those defined for the *EmulationScript* document type. For example, one of the higher level entities of network scenarios is the "Node" and the various planning documents and the EmulationScript itself define a "Node" element that refers (usually by "name") to indivual nodes within the scenario. And, generally, the different planning document types are used to describe different aspects of "Node" behaviors, roles, and properties within the planned experiment. For example, the "*MotionPlan*" document type can be used orchestrate the location and motion of Nodes within a scenario while the "*NetworkPlan*" document type describes the networks planned and which Nodes participate in those networks.

An initial set of planning document types is specified to support automated generation of modeling system configuration files. The set will be expanded during the course of time. The initial set are the planning document types most needed to automatically generate mobile network scenarios. The initial set includes the following document types:

1. *MotionPlan* - orchestration of node mobility with sets of motion primitives and motion patterns.

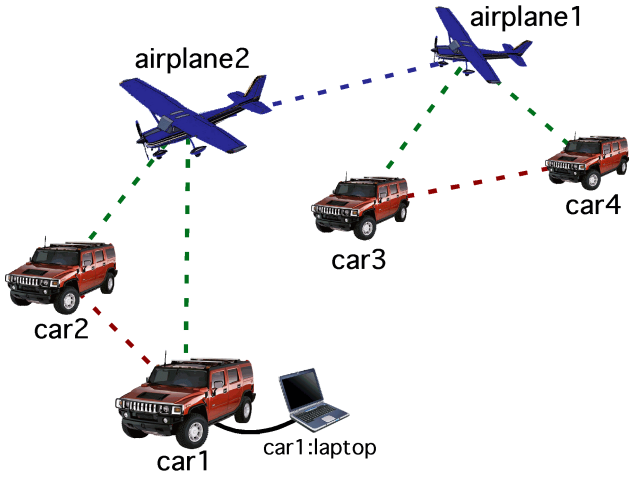
2. *NetworkPlan* - definition of networks and assignment of nodes (via interfaces) to those networks

Note that the common top-level <Node> element type is defined in each of these planning document types. The <Node> is one of the fundamental "Modules" for which the EmulationScript can convey properties and changes to those prop-

erties over time. For planning documents that are used to orchestrate dynamic properties (e.g. Node motion or location), "primitive" element types are defined and a "duration" attribute is commonly provided for these ...

For illustrative purposes in the description of EmulationScript planning and related documents, a "reference scenario" is used. A visualization of this example scenario is provided in Figure 2, “"Vehicular Network with Aerial Assist" Example Scenario”. This scenario represents a "vehicular network with aerial assist" where four ground nodes (cars) are assisted by two aerial nodes (airplanes) that can provide communication relay support. Note there are three different link types illustrated with the different color "dashed" lines. These include the "red" ground-to-ground "WIFI" links, "green" air-to-ground "WIMAX" links and the "blue" air-to-air "RADIO" link. Additionally, the "car1" Node has a "laptop" host aboard in addition to the assumed Node "default" network host. This scenario will be referenced in example MotionPlan and NetworkPlan content that is presented here.

Figure 2. "Vehicular Network with Aerial Assist" Example Scenario



4.1. Common Planning Elements

Planning documents share some common principal element definitions and these generally correspond directly to *EmulationScript* "modules" as described in Section 3.2, “Event Modules”. One of the most commonly reference modules is the <Node> type that corresponds to an "entity" within the modeled network environment. For example a <Node> might represent a person carrying a wireless network device or a vehicle that contains one or more computer "hosts" with associated network devices. In addition to principal elements, there are some additional common primitive element types that are used to coordinate the orchestration of different planning documents. For example the <mark> and <wait> primitives are used to designate (i.e., <mark>) the occurrence or completion of some arbitrary action by name and <wait> for some "marked" action to occur, respectively. This section describes the common planning elements.

4.1.1. <Node> Principal Element

The <Node> element as described in Section 3.2.1, “<Node> Module” is used to instantiate or reference "named" entitites within a planned scenario. For purposes of network modeling, <Node> instances contain <host> sub-elements that have associated network <interface> devices. This grouping allows the sub-element <host> and/or <interface> to inherit properties that are assigned on a "nodal" basis. For example, when a <Node> has a <location> or <motion> property defined in a *MotionPlan*, the associated <host> and <interface> sub-elements of that Node inherit that <location> (or <motion>) property. Descriptions of the use of Node sub-elements related to location or motion are provided in Section 4.2, “*MotionPlan* Documents” and descriptions of the use of sub-elements related to network components (e.g. "hosts" and/or "interfaces") are provided in Section 4.3, “*NetworkPlan* Documents”.

4.1.2. <mark> and <wait> Primitive Element

The <mark> and <wait> primitive elements provide a special planning mechanism that can be used in the planning documents to respectively annotate (i.e., <mark>) a specific emulation script event/time with a "name" and specify other planning action(s) with a dependency (i.e., <wait>) on that annotated cue (i.e. "marker name"). For example, if a <mark> primitive follows a <waypoint> primitive in a MotionPlan, the given <mark> (identified by "name") is to be cached in a resultant EmulationScript at the Event time cooresponding to _when_ that waypoint destination is reached (or its "duration" expires). Other plans may then specify a dependency to have an action take place upon the named <mark> event.

The <mark> and <wait> "name" values are considered *global* across the set of planning documents being processed. In the future, when additional planning documents (i.e. in addition to MotionPlan) documents are supported, the <mark> and <wait> primitives may be used to cue Node motion events off of other planning events or vice versa (e.g. Trigger a planned "comms" transmission off of a Node's arrival to a waypoint). Since the <mark> instances (identified by their "name" attribute) are global, the occurrences of <mark name="markerName\> MUST be unique with respect to execution of the planning documents. Thus, the <mark> primitive thus SHOULD NOT typically be embedded within a "pattern" primitive. In the future, a form of context-specific <mark> primitive may be defined that allows a <mark> to be instantiated within the namespace of some specific planning element (e.g. annotations implicitly marked with a concatenation of <nodeName:markerName>) that would allow embedding of <mark> annotations within "patterns. That feature is not yet fully defined.

Multiple <wait> instances can be dependent upon a single <mark> instance. Also note that it may be possible to specify "deadlock" cases with poor <mark> and <wait> usage. Planning tools should issue error messages when such deadlocks occur.

The formats of the <mark> and <wait> primitive elements are, respectively:

```
<mark name="markerName"/>
```

and

```
<wait>markerName</wait>
```

The intended use of the <mark> and <wait> primitives is to enable coordination and dependencies of multiple "plans" and their actions with one another. For example, if one MotionPlan document (or element), identifies a <mark> after a given waypoint, another MotionPlan may use the <wait> primitive to key other MotionPlan actions to occur when the given waypoint is reached (by the given "Node"). More specifically, this allows one to specify motion of one or more Nodes that is dependent on the actions of another Node. The <mark> and <wait> primitives allow script generation tools to parse planning documents (e.g. MotionPlan documents) and iteratively process them to create a final EmulationScript that implements the series of dependent actions. The following MotionPlan excerpt provides an example of <mark> and <wait> usage:

```
<MotionPlan>

   <Node name="node1">
      <location type="cartesian">100,100,0</location>
      <waypoint>
         <destination type="cartesian">100,200</destination>
         <velocity>10.0</velocity>
      </waypoint>
      <mark name="node1Waypoint1"/>
      <wait>node2Waypoint1</wait>
      <waypoint>
         <destination type="cartesian">100,300</destination>
         <velocity>10.0</velocity>
      </waypoint>
      <mark name="node1Waypoint2"/>
      <waypoint>
```

```
   </Node>

   <Node name="node2">
      <location type="cartesian">200,100,0</location>
      <wait>node1Waypoint1</wait>
      <waypoint>
         <destination type="cartesian">200,200</destination>
         <velocity>10.0</velocity>
      </waypoint>
      <mark name="node2Waypoint1"\>
      <wait>node1Waypoint2</wait>
      <waypoint>
         <destination type="cartesian">200,300</destination>
         <velocity>10.0</velocity>
      </waypoint>
      <mark name="node2Waypoint2"\>
      <waypoint>
   </Node>

</MotionPlan>
```

In this example, "node1" begins at an initial location of (100,100) and immediately begins moving towards the waypoint destination of (100,200). However, "node2" waits until "node1" reaches its first waypoint (marked as "node1Waypoint1") before beginning its first motion. Similarly, "node1" waits at its first waypoint until "node2" reaches its first waypoint (marked as "node2Waypoint1") before moving to its second waypoint. In this example, the use of <mark> and <wait> primitives allow a staggered motion among the two nodes to be planned.

4.2. *MotionPlan* Documents

The "EmulationScriptSchema.xsd" defines a "*MotionPlan*" XML element that can be used at the top level in a document that describes motion sequencing on a nodal basis. The goal of this document is to provide a relatively free-form method for users to sequence node motions. A software tool is provided then that generates "*EmulationScript*" documents from these "*MotionPlan*" documents. To summarize, the "*EmulationScript*" specifies motion events on a time-ordered basis with the tight requirement that motion updates specifically list the current location of the node when the update occurred. This makes the emulation script useful for run-time motion generation that might be dynamically controlled (e.g. shuttled over in time), but complicated to generate. The *MotionPlan* is, in contrast, based on defining the desired sequences of motion on a nodal basis in such a way that it provides a relatively simple user (or algorithmic) format for specifying node motion. Then, the *MotionPlan* format can be used by software tools to generate *EmulationScript* <Node:motion> update events.

The "*MotionPlan*" schema lets complex motion be described as a concatenation of motion primitives (directly related to the motion types the EmulationScript schema supports). Furthermore, sequences of motion primitives can be encapsulated as named motion "patterns" and those patterns can be re-used by reference to specify, possibly repetitive, node motion or as part of more complex pattern definitions. The following pseudo-XML provides a simplified overview of the *MotionPlan* schema:

```
<MotionPlan>
   <PatternDefinition name=”loop”>
      <waypoint>
         <destination> lat1, lon1, alt1</>
         <velocity>12.0</>
      </waypoint>
      <waypoint>
         <destination> lat2, lon2, alt2</>
         <velocity>12.0</>
      </waypoint>
      <waypoint>
         <destination> lat3, lon3, alt3</>
         <velocity>12.0</>
      </waypoint>
```

```
      <waypoint>
         <destination> lat1, lon1, alt1</>
         <velocity>12.0</>
      </waypoint>
   </PatternDefinition>
   <Node name=”node01”>
      <location> lat0,lon0,alt0</location>
      <pause>120.0</pause>
      <pattern repeat=”-1” duration=”600.0”>loop</pattern>
      <waypoint>
         <destination>lat4,lon4,alt4</>
         <velocity>20.0</>
      </waypoint>
      <pause>120.0</>
      <circle>
         <center>lat5,lon5,alt5</>
         <radius>30.0</>
         <velocity>15.0</>
      </circle>
   </Node>
</MotionPlan>
```

In this approximate *MotionPlan* example, a “loop” pattern is defined that consists of a triangle of three waypoints. Then, the “node01” motion plan is specified with an initial location and pause of 120.0 seconds followed by the triangle “loop” pattern with an undefined number of “repeats”, but limited to a “duration” of 600.0 seconds. After the 600.0 seconds of time spent in the “loop”, the node traverses to a specified <waypoint> and begins another 120.0 second <pause>. After this pause, our “node01” finally enters into a 30 meter radius <circle> motion pattern for an indefinite amount of time.

The file “`mpExample1.xml`” included with the example tools and schema documents distribution contains a version of the pseudo-example of Figure 3, but with real GPS coordinates. The *mp* utility can be used to parse this example *MotionPlan* file and generate a resultant EmulationScript document with the corresponding location/motion events for the “node01” entity.

4.2.1. Motion Primitives

The “*MotionPlan*” motion primitives include basic elements that correspond to the “*EmulationScript*” motion types and some added elements to pace and sequence these types. In the “*MotionPlan*”, an emulation node’s motion is expressed as a concatenated sequence of primitive motion types. The current motion types include <`location`>, <`waypoint`>, <`vector`>, <`circle`>, and <`loiter`> plus a <`pause`> primitive to temporarily halt motion at the current point in the motion plan. Additionally, a <`pattern`> primitive is provided that reference a sequence of motion primitives contained in a named <`PatternDefinition`>. The motion primitives also have a “duration” attribute that can be set to a specify time limits for each primitive to pace the execution of the concatenated set of motion primitives. An intended use of the "*MotionPlan*" is to generate a set of Node mobility events for an "*EmulationScript*".

The “*MotionPlan*” is a slightly loose specification of motion elements and so specific behaviors must be described for transitioning from one motion primitive to the next. For example, the transition from a prior location to a waypoint is relatively obvious. The<waypoint> motion primitive implicitly indicates movement from a prior location to a given "waypoint:destination" at the "waypoint:velocity”. However, transitioning to a "circle" motion from a previous location may require that an intermediate <waypoint> motion event be generated to smoothly transition to a location on the circle's perimeter. As an example for this case, the RECOMMENDED approach for this is to establish a direct route (great circle route when GPS geometry is used) from the last motion location towards the "circle:center" to a point that intersects the circle perimeter path. This intersecting location will serve as the initial location for the circle motion. In the case that the previous location is _within_ the circle's radius, then an intermediate waypoint specifying direct (shortest) route to the circle perimeter SHOULD be generated. The "velocity" for these intermediate waypoint transitions SHOULD be that of the target circle motion (i.e. "circle:velocity"). And, as noted below, the time required to complete the transition SHOULD be considered an inclusive part of any maximum motion pattern "duration" time set for the circle motion specification.

4.2.1.1. <location> Primitive

The <location> primitive specifies immediate re-location (i.e., teleportation) to specified coordinates. The <location> element has a “type” attribute with an implicit default value of “gps” (geodetic coordinates). The attribute may alternatively be set to a value of “cartesian” to indicate the coordinates represent Cartesian coordinates.

The “duration” attribute can be used to specify how long the node should remain at the specified location. Also a <pause> primitive following a <location> element equivalently indicates the dwell time the node should remain at the specified location before beginning transition to any succeeding motion primitive (or pattern). Note that the “duration” attribute value and a succeeding <pause> time will be additive.

The default format of the <location> element is:

```
<location>lat,lon,alt</location>
```

The transition to the location is an immediate change in location from the last location of a preceding motion primtive.

4.2.1.2. <waypoint> Primitive

The <waypoint> primitive specifies a “destination” towards which the node should move at specified “velocity”. The motion proceeds until the destination is reached or the “duration” attribute time expires, if specified. When the waypoint motion has completed (i.e. when the “duration” time has expired if applicable, or the waypoint destination is reached if no “duration” time limit is specified), the subsequent motion primitive is considered. If a specified “duration” time is longer than it takes to reach the “destination”, then the node will pause at the “destination” location until the “duration” time expires.

Note the destination element type is the same as that of the <location> primitive previously described where its coordinate format is “gps” by default with the “cart” value option available via the “geometry” attribute. The default units of “velocity” elements in this document is “meters per second”, but alternative “units” are available if the corresponding attribute is set. However, the example tools (described later) currently implemented against this specification assume units of “meters” and “meters per seconds” for distance and velocity values, respectively and do not yet support other unit types.

The format of the <waypoint> element is:

```
<waypoint>
   <destination>lat,lon[,alt]</destination>
   <velocity>metersPerSec</velocity>
</waypoint>
```

The transition to the waypoint motion from a prior location is simply the waypoint specification itself.

4.2.1.3. "vector" Primitive

The <vector> primitive specifies a direction, given in “azimuth” and optional “elevation” angles at which the node should move at specified “velocity”. The motion is proceeds indefinitely or until its “duration” attribute time expires, if applicable. Thus a “duration” attribute MUST be specified unless the desired behavior is continuous, unbounded motion along the given vector direction.

The default units type for “velocity” elements in this document is “meters per second”, but alternative “units” are available if the corresponding attribute is set. However, the tools currently implemented against this specification assume units of “meters” and “meters per seconds” for distance and velocity values, respectively. The “azimuth” angle is with respect to due North (GPS coordinates) or the y-axis (Cartesian coordinates) and MUST be in the range of ±360°. The optional “elevation” angle MUST be in the range of ±90° and sets the portion of the motion velocity applied towards altitude (or z-axis) position change. Note that at “elevation” values of +90° or -90°, there is no horizontal motion in the bearing direction given by the “azimuth” angle (i.e. the motion is applied 100% as a change in altitude or z-axis position). The default “elevation” value is 0.0 degrees (i.e. no vertical motion).

The format of the <vector> element is:

```
<vector>
   <velocity>metersPerSec</velocity>
   <azimuth>degrees</azimuth >
   <elevation>degrees</elevation >
</vector>
```

The transition to the vector motion is simply to begin the indicated motion direction/velocity from the current location. Note that since the <vector> motion primitive contains no absolution location parameters, it is very applicable for <PatternDefinition> uses where relative motion behaviors can be specified from an arbitrary initial location.

4.2.1.4. <circle> Primitive

The <circle> primitive is used specify motion along the perimeter of a circle of a specified “radius” (in meters by default) about a “center” location at a given “velocity” (again, “meters per second” by default). The “altitude” (or “z-axis”) coordinate is fixed and indicated as part of the “center” location specification. Note that the “center” may be omitted and will be assumed to be the current node location according to the motion plan (i.e. based upon the prior node location and/or motion specified). The <circle> also has an optional “revs” element that indicates the maximum number (or fractional number) of circle revolutions to complete before motion is completed (and halted if no subsequent motion is specified). Note the “revs” value is a floating-point number and thus can specify partial completion of the specified circle as desired. If the “revs” are completed before any specified “duration” time expires, then the motion remains halted at the last location until the “duration” expires before proceeding to the next motion primitive.

The default format of the <circle> element is:

```
<circle>
   <center>lat,lon[,alt]</center>
   <radius>meters</radius>
   <velocity>metersPerSec</velocity>
   <revs>value</revs>
</circle>
```

The transition to the <circle> motion from a prior location is to set an intermediate waypoint motion pattern (using the circle motion velocity) directly towards the “center” location that terminates on the perimeter of the circle. In the case the prior location is within the radius of the circle, an outward-spiraling motion is used to reach the circle perimeter with the spiral progressing at a fixed angular velocity based on the circle’s objective “radius” and “velocity”. Note that the time consumed by this transition motion is considered part of motion “duration” time limit, if applicable.

4.2.1.5. <loiter> Primitive

The <loiter> primitive is provided to specify a circular (hovering or loitering) motion that is relative to another embedded motion primitive or pattern. The form of the <loiter> element is:

```
<loiter>
    <reference motion primitive or pattern reference/>
    <velocity>metersPerSec</velocity>
    <radius>meters</radius>
    <height>meters</height>
</loiter>
```

The transition to the <loiter> motion is that same as that for a <circle> motion primitive. Note that the <velocity> aspect of the <loiter> motion is converted to a constant *angular* velocity (i.e. "degrees" per second) of the revolution of the motion about the center reference motion. Any other motion primitive (except <loiter>) or pattern may be applied as the reference motion for the <loiter>. The loiter circle <radius> (in meters) child is required, but the offset <height> (in meters) is optional and is assumed to be zero if omitted.

Note the <loiter> motion will continue indefinitely, even if the reference motion halts, unless the motion primitive "duration" attribute is applied to the <loiter> primitive to limit the time spent observing this motion pattern. The

<loiter> "duration" value overrides the duration(s) of the reference motion or pattern components (i.e., the total motion time here will not exceed the <loiter> duration>).

The following XML excerpt provides a couple examples of <loiter> usage:

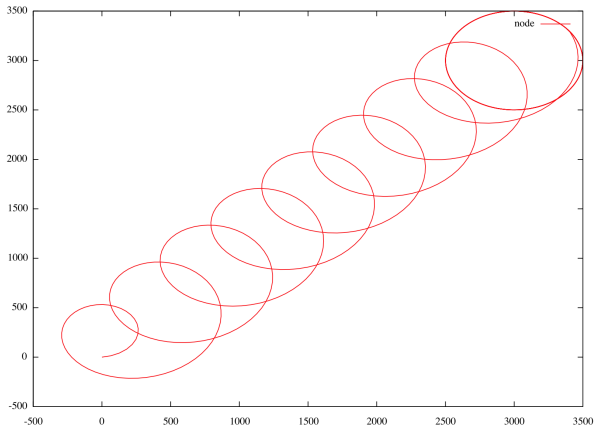
```
<location type="cartesian">0, 0</location>
<loiter duration="500.0">
    <waypoint>
        <destination type="cartesian">3000, 3000</destination>
        <velocity>10.0</velocity>
    </waypoint>
    <velocity>60.0</velocity>
    <radius>500.0</radius>
</loiter>

<loiter>
    <pattern>path</pattern>
    <velocity>50.0</velocity>
    <radius>100.0</radius>
    <height>1000.0</height>
</loiter>
```

In the first <loiter> primitive above, the resulting motion will be a circular motion centered about a traversal to given <waypoint>. The center traversal proceeds using the <waypoint> velocity of 10.0 meters/sec. The circular (loiter/hover) motion will revolve at a constant angular velocity about the moving center with an optional <height> offset. The angular velocity is a function of the <loiter> velocity and radius and is computed as:

```
angular velocity = <loiter:velocity> / <loiter:radius>
```

in units of radians per second. Although this might not 100% correspond to real-world motion behaviors, it is expected to be sufficient for network connectivity modeling purposes. More sophisticated loiter motion models may be introduced to this schema in the future. An illustration of the resulting motion for the first example from above where the <loiter> traverses from a center location (0, 0) to (3000,3000) is given in Figure 3, “Loiter motion example”.

Figure 3. Loiter motion example



4.2.1.6. <randpoint> Primitive

The <randpoint> primitive can be used to generate randomly selected node waypoints or locations. Each time a <randpoint> primitive is process it may generate a random waypoint or location within the bounding location box, time, and speed bounds of the given RandpointGenerator. Note that the named RandpointGenerator MUST be defined within a previous or within the same file for the <randpoint> to be successfully processed. Note that the <randpoint>

motion type is *not* part of the <Event> motion type set as the random points are transformed to a set of waypoints when processing *MotionPlan* documents to create resultant *EmulationScript* documents.

The format of the <randpoint> element is:

```
<randpoint seed="1" mode="variable">generatorName</randpoint>
```

The optional "seed" attribute value can be used to initialize the random number generator of the referenced RandpointGenerator instance. The default seed value is "1". An example use of this seed attribute would be to create a repeatable "pattern" compromised of random waypoints. The "seed" attribute could be used on the first <randpoint> in the pattern to ensure the random number generation state for the associated RandpointGenerator is set to a specific value at the start of pattern generation.

The optional "mode" attribute is one of three possible behaviors that are observed when the <randpoint> primitive is processed:

1. "variable" - the default behavior (when no "mode" is specified) results in a fresh random waypoint destination each time the given <randpoint> is processed (i.e. if it is embedded within a "pattern").

2. "fixed" - In this "mode", a fresh waypoint destination is generated ONLY on the _first_ time the <randpoint> is processed (i.e. if it is embedded within a pattern). On subsequent encounters of an embedded fixed <randpoint>, the same destination location is assigned to the waypoint, but with potentially different random speed within any specified speed or time bounds for the associated RandpointGenerator. A "pattern" with a mix of "variable" and "fixed" randpoints can be used so that multiple nodes can periodically converge (in time and/or space given the bounds) in a repeated fashion at the "fixed" (but randomly generated) location(s).

3. "instant" - In this "mode" a <waypoint> is NOT generated, but instead a random <location> is generated to which the node instantly "teleports". Nodes MAY be assigned an instant <randpoint> _instead_ of a explicit <location> as an initial location within MotionPlan documents. The location bounding box of the associated RandpointGenerator is used, but any time or speed bounds are ignored.

The following example illustrates the use of the <randpoint> primitive to set random initial locations for nodes _and_ a subsequent random waypoint. A "<pattern> is defined so the 3 Nodes shown follow the same pattern (random initial location and random waypoint).

```
<MotionPlan>

  <RandpointGenerator name="rgen">
    <minLocation type="gps">38.778750, -77.082917</minLocation>
    <maxLocation type="gps">38.895139, -76.969306</maxLocation>
    <minVelocity>5.0</minVelocity>
    <maxVelocity>25.0</maxVelocity>
    <minTime>10.0</minTime>
    <maxTime>180.0</maxTime>
    <seed>20212</seed>
  </RandpointGenerator>

  <PatternDefinition name="rand">
    <randpoint mode="instant">rgen</randpoint>
    <randpoint>rgen</randpoint>
  </PatternDefinition>

  <Node name="node01">
    <pattern>rand</pattern>
  </Node>

  <Node name="node02">
    <pattern>rand</pattern>
  </Node>
```

```
  <Node name="node03">
    <pattern>rand</pattern>
  </Node>

</MotionPlan>
```

Note that if the first <randpoint> in the <PatternDefinition> had its optional "seed" attribute, then all three nodes would end up with the same initial random location and subsequent random waypoint (including velocity) being generated.

4.2.1.7. <pause> Primitive

The <pause> primitive element is provided to specify intervals of immobility from the completion of a prior motion primitive or pattern before beginning transition to a subsequent motion primitive or pattern.

The format of the <pause> element is:

```
<pause duration="seconds"/>
```

where “seconds” is a floating point value greater than or equal to zero.

4.2.1.8. <pattern> Primitive

The <pattern> primitive identifies (by name) a motion sequence that was previously-defined using the “PatternDefinition” element described below. The <pattern> element also has an optional “repeat” attribute that can be used to indicate how many times the motion sequence should repeat before completing and proceeding to any subsequent motion. The default “repeat” value is “0” indicating the motion pattern is executed once while a negative “repeat” value indicates the pattern should be repeated indefinitely.

The format of the <pattern> element is:

```
<pattern repeat=”0”>patternName</pattern>
```

Note that, as with other motion primitives, the “duration” attribute can be applied to limit the time period during which the motion pattern is executed before transitioning to the next motion, if any. If not “duration” is specified, the <pattern> motion is considered complete when the number of “repeats” has occurred.

4.2.1.9. <mark> and <wait> Primitives

The <mark> and <wait> primitive elements as described in Section 4.1.2, “<mark> and <wait> Primitive Element” can be applied to *MotionPlan* document Node motion or <PatternDefinition> sequences. An example of their use in a *MotionPlan* is provided in that section.

4.2.2. Motion Pattern Definition

A <PatternDefinition> element is provided to encapsulate a concatenated set of motion primitives and assign a “name” to the defined pattern. Then, the <pattern> motion primitive element can be used to refer to that definition by name and used as a motion primitive to specify node motion or even as part of other, more complex <PatternDefinition> instances. The content of the <PatternDefinition> is a set of the motion primitive elements previously described and the “name” attribute is used to assign a name string to the pattern defined. The format of the <PatternDefinition> element is:

```
<PatternDefinition name=”patternName”>
   <motion primitive 1/>
   <motion primitive 2/>
   …
</PatternDefinition>
```

The purpose of the <PatternDefinition> element is to allow a motion “circuit” or “path” to be defined that might be re-used by multiple mobile nodes at different <Node:timeOffset> values (see below). <PatternDefinition> elements are top-level elements of *MotionPlan* documents.

4.2.3. Random Waypoint Generator Definition

A <RandpointGenerator> element is provided to define a random waypoint generator instance that can be referenced by name by <randpoint> motion primitives to generate a random waypoints as part of a motion plan. Each <RandpointGenerator> instance has a set of parameters that control the nature of the waypoints generated (e.g. x,y,z location bounding box, min/max velocity bounds, etc).

```
<RandpointGenerator name=”generatorName”>
   <seed> value </seed>
   <minLocation> x,y[,z] min bounds </minLocation>
   <maxLocation> x,y[,z] max bounds </maxLocation>
   <minVelocity> value< /minVelocity>
   <maxVelocity> value </maxVelocity>
   <minTime> value< /minTime>
   <maxTime> value </maxTime>
</RandpointGenerator>
```

The <seed> is an integer value that seeds the random number generator used for selecting the random waypoints and velocity of motion. <RandpointGenerator> instances that use the same <seed> value and other parameters will generate the same set of random waypoints. Thus, if one defined a <RandpointGenerator> per node and those nodes followed similar patterns of randpoint motion, they could be set to follow the same random pattern (Note that creating a pattern of "fixed" mode randpoints could do the same although the approach described here could embed the randpoint in a <PatternDefinition> (per node) and with a "-1" repeats value, created indefinite but equal patterns of random motion). Use of this approach with <timeOffset> and/or<locationOffset> values set for the nodes could allow creating of scripted "group" random motion patterns.

The <minLocation> and <maxLocation> elements are used to define a bounding box from which the resultant random waypoint destinations are selected.

The <minVelocity> and <maxVelocity> elements provide a range over which the velocity used to reach a selected waypoint is picked using a uniformly distributed random variable.

The <minTime> and <maxTime> elements allow the optional specification of a time bound on how much time is spent traversing to waypoints. If <minTime> and <maxTime> are set to a negative value, then there is now lower or upper bound on traversal time. If they are set to equal values, then the velocity is adjusted (within limits of <minVelocity> and <maxVelocity>) to meet that schedule. This lets the generator be configured to generate waypoints for specific time intervals and allows a set of nodes following a common motion pattern defined that include "fixed" mode randpoints to periodically converge at the fixed point(s) in space *and* time.

4.2.4. Node *MotionPlan* Specification

In addition to <PatternDefinition> elements, “*MotionPlan*” documents can contain one or more <Node> elements that encapsulate motion primitives and/or patterns that are assigned to the node instance indicated by the <Node:id> attribute. The <Node> element may start with an *optional* <timeOffset> and/or <locationOffset> element. It then has a mandatory <location> element that specifies the initial location of the node. After that, any number of motion primitives and/or patterns may be applied to specify the node’s motion. The format of the <Node> element is:

```
<Node name=”nodeId”>
   <timeOffset>seconds</timeOffset>
   <locationOffset>
      <distance>meters</distance>
      <azimuth>degrees</azimuth>
      <elevation>degrees</elevation>
   </locationOffset>
```

```
   <location>lat,lon,alt</location>
   <motion primitive or pattern 1>
   <motion primitive or pattern 2>
   <motion primitive or pattern 3>
      …
</Node>
```

The <locationOffset> has the following form:

```
<locationOffset>
    <distance>meters></distance>
    <azimuth>degrees</azimuth>
    <elevation>degrees</elevation>
</locationOffset>
```

The <distance> specifies the magnitude (in meters) of the offset from any given or computed location for the <Node> while the <azimuth> and <elevation> are used to specify the direction of the offset. The <azimuth> value is in units of degrees in the range of -180.0 degrees to +180.0 degrees. The azimuth angle is relative to "North" for geodetic (gps) coordinates. The <elevation> value is also in units of degrees in the range of -90.0 to +90.0 degrees ("straight down" to "straight up", respectively). The location offset, when given, is applied to all locations set or computed for the given node within a *MotionPlan* document. Only a single <locationOffset> may be applied per <Node> instance. However, separate *MotionPlan* <Node> instances may be applied to a given named node and the primitives of those separate instances are logicially concatenated. The subsequent *MotionPlan* <Node> references/instances MAY provide overriding <locationOffset> specifications, if desired, but will result in discontiguous motion. It MAY be similarly possible specify different <offsetTime> values for a given node, but again, disjoint motions will tend to result. It is generally safer to use the <pause> primitive to play games with time, if desired.

TBD - Also mention that the a <Node:heading> and <Node:orientation> element with <pitch>, <yaw>, and <roll> values are planned _and_ that child <Node:host> elements MAY potentially each have their own individual <locationOffset> which is relative to the Node:location _and_ the Node:heading/orientation (This will matter for large platforms with distributed antenna and if we eventually start to consider superstructure, etc in propagation prediction, etc)

4.3. *NetworkPlan* Documents

The purpose of the *NetworkPlan* document type is to allow a somewhat general definition of the communication devices and attributes associated with Nodes (and/or their associated "hosts") in planned scenarios. Additionally, relationships among the devices can be created by defining logical "Network" groups and associated the specified communication devices (a.k.a. "interfaces") with those networks. The *NetworkPlan* also supports configuration-time assignment of layer 3 network addresses to systems within the scenario if desired. Also, some standard parameters of network interfaces, including wireless interfaces, MAY be expressed within *NetworkPlan* documents and these will be mapped to the corresponding model of the interface device in a target modeling system. The key to this mapping is the "type" attribute that MUST be associated with <interface> instances within the *NetworkPlan*. It is expected that system-specific templates or tools will be able to map a *NetworkPlan* <interface> "type" name to a corresponding model implementation. The parameters associated with *NetworkPlan* interfaces (e.g. wireless transmission power, frequency, etc) are expected to be assignable to that corresponding implementation in some meaningful manner.

Note the <interface> sub-element is used in slightly different ways in *NetworkPlan* <NetworkDefinition> and <Node> structures. However, in both cases, the <interface> sub-element can use the "common" interface parameters described in Section 3.2.1.2, “Node/host <interface> Sub-element”.

Here is an excerpt of the NetworkPlan for our reference "vehicular network with aerial assist" scenario:

```
<NetworkPlan>

    <NetworkDefinition name="GroundNet">
        <address>192.168.1.0/24</address>
        <interface type="WIFI" assign="default">
```

```
            <freq units="GHz">2.42</freq>
            <power units="Watts">6</power>
        </interface>
    </NetworkDefinition>

    <NetworkDefinition name="Air2GroundNet">
        <address>192.168.2.0/24</address>
        <interface type="WIMAX">
            <freq units="GHz">3.5</freq>
            <power units="Watts">6</power>
        </interface>
    </NetworkDefinition>

    <NetworkDefinition name="AirborneNet">
        <address>192.168.3.0/24</address>
        <interface type="RADIO">
            <freq units="GHz">4.56 </freq>
            <power units="Watts">60</power>
        </interface>
    </NetworkDefinition>

    <Node name="airplane1">
        <interface name="radio0" net="AirborneNet" assign="default">
            <power units="Watts">60 </power>
        </interface>
        <interface name="wimax0" net="Air2GroundNet"/>
    </Node>

    <Node name="airplane2">
        <interface name="radio0" net="AirborneNet" assign="default">
            <power units="Watts">60 </power>
        </interface>
        <interface name="wimax0" net="Air2GroundNet"/>
    </Node>

    <Node name="car1">
        <interface name="wifi0" net="GroundNet"/>
        <interface name="wimax0" net="Air2GroundNet"/>

        <NetworkDefinition name="lan">
            <interface type="ETH" assign="autoconf">
            <address>192.168.100.0/24</address>
        </NetworkDefinition>

        <interface name="eth0" net="lan">
            <address>192.168.100.1</address>
        </interface>

        <host name="laptop" >
            <interface name="eth0" net="lan">
            </interface>

        </host>
    </Node>

    <Node name="car2">
         <interface name="wifi0" net="GroundNet"/>
         <interface name="wimax0" net="Air2GroundNet"/>
    </Node>

    ...

</NetworkPlan>
```

In this example, three logical <NetworkDefinition> specifications are made. Then <Node> instances are created with <interface> assignments that are mapped to the defined networks via the <interface> "net" attribute. Note that the <interface> child element of a <NetworkDefinition> defines the "type and characteristics of a "default" interface that may be used to attach to the given network. However, <Node> definitions MAY also specify "new" <interface> types to be associated with a <NetworkDefinition>. It is expected that the scenario planner is assigning "compatible" <interface> type derivatives or simply wishes to have configured addresses assigned from the <NetworkDefinition:address> space, if applicable. Also note that, for the most part, the <Node:interface> declarations get their "type" *indirectly* defined via their reference to a <NetworkDefinition> with a default <interface> specification included.

Also note the use of a "Node-scoped" <NetworkDefinition> in this example for the Ethernet lan aboard the car1 <Node> instance. The "car1:default:eth0" interface is given an overriding explicit address assignment (it is the auto-configuration server in our example scenario) at pre-configuration time while the "car1:laptop:eth0" interface is to receive an address during experiment run-time via auto-configuration. Having these "ETH" (Ethernet) interface instantiations reference a common <NetworkDefinition> allows configuration tools to recognize which interfaces should be interconnected (i.e. attached to the same hub or wire).

4.3.1. <NetworkDefinition> Element

The <NetworkDefinition> element allows for a logical "network" to be declared (by name) and have a single, optional <address> space and/or optional default <interface> specification assigned. When <NetworkDefinition> instances are included in a *NetworkPlan* <Node:interface> (or <Node:host:interface>) declarations MAY reference a <NetworkDefinition> for configuration information. <NetworkDefinition> declarations may be included as top level elements in a *NetworkPlan* document as "globally-scoped" instances and MUST be assigned unique names. <NetworkDefinition> declarations may also be included as child elements in <Node> instances for "Node-scoped" network definitions that may be used to interconnect <host> children of the Node when present. In this case the Node-scoped <NetworkDefinition> "name" is logically prefixed with the <Node> "name" value with a colon delimiter. When an *NetworkPlan* <interface> instance references a <NetworkDefinition> by name (via the <interface> "net" attribute), the scope of parent <Node> is first searched for a matching <NetworkDefinition> name *before* the global <NetworkDefinition> scope is searched.

The <NetworkDefinition> has a single REQUIRED "name" attribute that is used to assign a unique name to the logical network being defined.

The optional <NetworkDefinition> <address> and <interface> sub-elements used to establish some default characteristics for Node/host <interface> instances are described in detail in sections below. However, a <NetworkDefinition> can still provide the useful role pf co-associating different Node/host <interface> instances together (when they refer to a common <NetworkDefinition> instance be name) even if it does not include the <address> or <interface> children.

The following *NetworkPlan* excerpt provides the "GroundNet" <NetworkDefinition> for the reference example scenario:

```
<NetworkDefinition name="GroundNet">
    <address>192.168.1.0/24</address>
    <interface type="WIFI" assign="default">
        <freq units="GHz">2.42</freq>
        <power units="Watts">6</power>
    </interface>
</NetworkDefinition>
```

4.3.1.1. Network Definition <address> Sub-element

The <NetworkDefinition> MAY contain an <address> child element that can be used specify a network address space for the network. The format of this sub-element is the same as described in Section 3.2.1.2.1, “Interface <address> Parameter”. However, only a single address space MAY be associated with a <NetworkDefinition> instance (This

may be re-visited in the future). Currently IPv4 or IPv6 address spaces can be specified in the <address> element text content in the form of "prefix/maskLength" where the "prefix" is a valid IPv4 or IPv6 address and the "maskLength" is the length of the "prefix" in bits. The specification of an address space is OPTIONAL. If used, addresses from this space SHOULD be assigned to interfaces associated with the <NetworkDefinition> (via the <interface> "net" attribute) if the <interface> "assign" attribute is set to a value of default or omitted. Possible <interface> "assign" values are described in Section 4.3.2, "*NetworkPlan* <Node> Elements".

In the future, the <address> element construct may be enrichened to support additional address families and functions.

4.3.1.2. Network Definition <interface> Sub-element

The <NetworkDefinition> MAY also contain an <interface> child element to describe the "type" and default parameters of interfaces to be associated with the defined network. The <NetworkDefinition> <interface> element has the following attributes:

1. A REQUIRED "type" attribute that declares or identifies the interface device type by name. Modeling system-specific configuration tools will use this type identifier to map a particular <interface> "type" to its corresponding implementation model.

2. An OPTIONAL "assign" attribute that specifies how or if network addresses are to be assigned to resultant <interface> instances at configuration time, run-time, or never. Possible values for the "assign" attribute include:

   default — An address should be assigned at configuration-time from the <NetworkDefinition> address space, if defined.

   autoconf — An address is to be assigned to the interface at run-time via auto-configuration.

   none — An address should not be assigned to this interface.

Note the <NetworkDefinition> <interface> element does *not* need to have a "name" attribute specified.

<Node:interface> instances that reference the <NetworkDefinition> will inherit the given <NetworkDefinition> default <interface> "type" and parameters unless those instances provide and overriding "type" or parameter values. A <NetworkDefinition:interface> specification MUST include a "type" element. If a <NetworkDefinition> does not contain a default <interface> specification, then referencing <Node:interface> instances MUST provide a "type" declaratiion. Note that <address> children can not be associated with <NetworkDefinition:interface> instances since these provide a "model" for interfaces associated with the defined network rather than actual interface device instances.

4.3.2. *NetworkPlan* <Node> Elements

The *NetworkPlan* document type references the <Node> module and its <host> and <interface> sub-elements per the "Node:host:interface" taxonomy described in Section 3.2.1, "<Node> Module". The main purpose of the *NetworkPlan* is to associate network interface devices with the Nodes and/or their associated hosts. Global or Node-scoped <NetworkDefinition> instances can be created and referenced so that system configuration tools can automatically assign network addresses to these interfaces, if desired.

The use of the <Node> and <host> elements in the *NetworkPlan* is to simply identify, by their respect "name" attribute values, which entity (Node) and/or subsystem (host) with which a specified <interface> is associated.

4.3.2.1. Node <interface> Sub-element

The <interface> sub-element in the *NetworkPlan* makes use of the "name", "type" and "net" attributes to establish a named interface and device type for the given parent <Node> or <host>, and to optionally associate it with a specific <NetworkDefinition> instance. Note that when an <interface> references a <NetworkDefinition> that incudes its own default <interface> specification, the use of the "type" attribute is OPTIONAL since the referenced <Net-

workDefinition:interface> provides a default "type" specification. The value of the <interface> "type" is intended to be used by configuration tools to map *NetworkPlan* <interface> instances to corresponding model implementations identified in a system-specific "template" document (e.g. *EmaneTemplate* document, etc).

The *NetworkPlan* Node/host <interface> sub-element has the follow attributes:

1. REQUIRED "name" attribute to declare a name identifier for the interface. This value must be unique within the context of the parent <Node> or <host> instance.

2. "type" attribute that MAY be used to specify the interface device type by name. This attribute is OPTIONAL if the "net" attribute is present and references a <NetworkDefinition> with its own default <interface> specification, but is REQUIRED if not.

3. "net" attribute that MAY be used to reference, by name, a <NetworkDefinition> instance. If a "net" is not specified, then the "type" attribute must be included.

4. An OPTIONAL "assign" attribute that can be used to specify how/if an address is to be assigned to the interface. The value of the Node/host <interface> "assign" attribute overrides that of an associated <NetworkDefinition>.

As described in Section 3.2.1, "<Node> Module", an <interface> element MAY be a direct child of a <Node> parent and a <host> "name" value of default is implicit for this case.

By default, the Node/host <interface> is assigned (or not) addresses per the "assign" configuration of an associated <NetworkDefinition>, if applicable. However, a Node/host <interface> MAY instead have specific address(es) explicitly assigned (in an overriding fashion) by use of one or more child <address> elements. The format of this sub-element is the same as described in Section 3.2.1.2.1, "Interface <address> Parameter". When an <address> child is given, any "assign" value is ignored.

The following *NetworkPlan* excerpt provides an example <Node> specification from the reference example scenario:

```
<Node name="car1">

    <interface name="wifi0" net="GroundNet"/>

    <interface name="wimax0" net="Air2GroundNet"/>

    <NetworkDefinition name="lan">
       <interface type="ETH" assign="autoconf"/>
    </NetworkDefinition>

    <interface name="eth0" net="lan">
        <address type="ipv4">192.168.100.1</address>
    </interface>

    <host name="laptop">
        <interface name="eth0" net="lan">
        </interface>
    </host>
</Node>
```

In this example, the implicit <Node> default host has interfaces named wifi0, wimax0, and eth0. The wireless interfaces (wifi0 and wimax0), reference globally-scoped <NetworkDefinition> instances (GroundNet and Air2GroundNet, respectively) and can receive parameters and address assignments from them. A "Node-scoped" <NetworkDefinition> child is included to establish an on-board lan network for the car1 Node. The eth0 interface is independently declared and has an address explicitly set (since it will act as an auto-configuration server in the planned scenario). The laptop host also has an "ETH"<interface> declared associated with the same lan as the default host "ETH" interface but is to receive an address assignment at experiment run-time via auto-configuration.

4.3.3. NetworkPlan Example

The following is an complete *NetworkPlan* that represents a configuration of interface devices for our reference example scenario:

```
<NetworkPlan>

    <NetworkDefinition name="GroundNet">
        <address type="ipv4">192.168.1.0/24</address>
        <interface type="WIFI" assign="default">
            <freq units="GHz">2.42</freq>
            <power units="Watts">6</power>
        </interface>
    </NetworkDefinition>

    <NetworkDefinition name="Air2GroundNet">
        <address type="ipv4">192.168.2.0/24</address>
        <interface type="WIMAX">
            <freq units="GHz">3.5</freq>
            <power units="Watts">6</power>
        </interface>
    </NetworkDefinition>

    <NetworkDefinition name="AirborneNet">
        <address type="ipv4">192.168.3.0/24</address>
        <interface type="RADIO">
            <freq units="GHz">4.56 </freq>
            <power units="Watts">60</power>
        </interface>
    </NetworkDefinition>

    <Node name="airplane1">
        <interface name="radio0" net="AirborneNet" assign="default">
            <power units="Watts">60 </power>
        </interface>
        <interface name="wimax0" net="Air2GroundNet"/>
    </Node>

    <Node name="airplane2">
        <interface name="radio0" net="AirborneNet" assign="default">
            <power units="Watts">60 </power>
        </interface>
        <interface name="wimax0" net="Air2GroundNet"/>
    </Node>

    <Node name="car1">
        <interface name="wifi0" net="GroundNet"/>

        <interface name="wimax0" net="Air2GroundNet"/>

        <NetworkDefinition name="lan">
            <address type="ipv4">192.168.100.0/24</address>
            <interface type="ETH" assign="autoconf"/>
        </NetworkDefinition>

        <interface name="eth0" net="lan">
            <address type="ipv4">192.168.100.1</address>
        </interface>

        <host name="laptop" >
            <interface name="eth0" net="lan"/>
        </host>
    </Node>
```

```
    <Node name="car2">
         <interface name="wifi0" net="GroundNet"/>
         <interface name="wimax0" net="Air2GroundNet"/>
    </Node>

    <Node name="car3">
         <interface name="wifi0" net="GroundNet"/>
         <interface name="wimax0" net="Air2GroundNet"/>
    </Node>

    <Node name="car4">
         <interface name="wifi0" net="GroundNet"/>
         <interface name="wimax0" net="Air2GroundNet"/>
    </Node>

</NetworkPlan>
```

5. *EmulationDirectory* Document

The *EmulationDirectory* document type provides an XML-based container for information that can provide mappings from the "named" Plannning Document or *EmulationScript* elements (e.g. named "`Node`", "`interface`", etc elements) to the instantations of representations of those elements within specific modeling systems. This document thus provides a sort of "name to identifier" resolution database that can be referenced by system configuration tools, the run-time system itself, or even during post-experiment data analysis to help map captured data (e.g. network traffic flows) against the planned scenario elements. The general structure of the *EmulationDirectory* document is reflective of the taxonomy of "Node:host:interface" and other principal network modeling elements. Some of the information (e.g. interface->address mappings) may be general while, in many cases, system-specific child elements are defined against target modeling architectures (e.g. EMANE, CORE, ns-3 etc). For example, "`emanePlatform`" and "`emaneNEM`" elements with "id" attributes are provided to map "`Node:host:interface`" scenario elements to specific EMANE platform or NEM identifiers that were realized during system configuration generation.

The *EmulationDirectory* document contains only information that is static during the execution of an experiment scenario with an emphasis on the mapping from the scenario planning "namespace" to the modeling system-specific representation. Emulation properties that are changed during the course of an experiment should be instead inserted into the EmulationScript document for the corrresponding scenario. A combination of the *EmulationDirectory* and the *EmulationScript* (or log) can then be reference (i.e., such as during post-processing) to correlate logged data or events with planned scenario components. As previously mentioned, it is possible that the *EmulationScript* document type can also be used as a logging format if desired. If this is the case, then the *EmulationDirectory* may also have utility in being used to map from any modeling system-specific identifiers to the planning/scripting namespace during logging operations.

Here is a short excerpt of an *EmulationDirectory* document for a scenario-driven configuration generated for an EMANE system using our reference "vehicular network with aerial assist" scenario.

```
<EmulationDirectory>
   <Node name="car1">
      <host name="default">
         <emaneHost host="emane3"/>
         <interface name="wifi0">
            <emanePlatform id="3" host="emane3"/>
            <emaneNEM id="5"/>
         </interface>
         <interface name="wimax0">
            <emanePlatform id="3" host="emane3"/>
            <emaneNEM id="6"/>
         </interface>
         <interface name="eth0">
            <emanePlatform id="3" host="emane3"/>
            <emaneNEM id="7"/>
```

```
         </interface>
      </host>
      <host name="laptop">
         <emaneHost host="emane4"/>
         <interface name="eth0">
            <emanePlatform id="4" host="emane4"/>
            <emaneNEM id="8"/>
         </interface>
      </host>
   </Node>
   <Node name="car2">
      <host name="default">
         <emaneHost host="emane1"/>
         <interface name="wifi0">
            <emanePlatform id="1" host="emane1"/>
            <emaneNEM id="1"/>
         </interface>
         <interface name="wimax0">
            <address>192.168.2.4</address>
            <emanePlatform id="1" host="emane1"/>
            <emaneNEM id="2"/>
         </interface>
      </host>
   </Node>
   <Node name="airplane2">
      <host name="default">
         <emaneHost host="emane5"/>
         <interface name="radio0">
            <emanePlatform id="6" host="emane21"/>
            <emaneNEM id="9"/>
         </interface>
         <interface name="wimax0">
            <emanePlatform id="5" host="emane5"/>
            <emaneNEM id="10"/>
         </interface>
      </host>
   </Node>
</EmulationDirectory>
```

5.1. Common EmulationDirectory Elements

The structure of the *EmulationDirectory* document reflects that of the Planning and *EmulationScript* documents where principal modules and their sub-elements are identified by "name" attributes. Then, some properties, including modeling system-specific ones, are associated with the modules and/or their sub-elements. In the example shown above, "Nodes" and their respective "hosts" and "interfaces" are identified from one or more *NetworkPlan* documents for the scenario. Then the `<Node:host>` and `<Node:host:interface>` instances are decorated with elements identifying the corresponding EMANE-specific components (i.e. `<emanePlatform>` and `<emaneNEM>` child elements).

It is possible that some non-system-specific scenario information may be included in an *EmulationScript* document. However,only information that is static during the entire course of experiment execution can be included. It is recommended that information that may change over the course of experiment execution be instead included in *EmulationScript* and/or logging documents that pertain to the experiment. Furthermore, it is also RECOMMENDED that any information that might ever possibly change during any experiment execution (even if it typically does not) NOT be included in *EmulationDirectory* documents but instead included or logged in other more appropriate containers in the interest of consistency (i.e. we don't want to have to build tools that have guess where they need to look for particular information!). For example, the *EmulationScript* document that includes events for any time-varying scenario information (as well as "time zero" configuration information) is a more appropriate repository for such information, although that document type is limited to general-purpose (i.e. not modeling system-specific) scenario details.

5.2. EMANE-Specific *EmulationDirectory* Elements

This section describes *EmulationDirectory* child elements that are specific to EMANE scenario configurations.

5.2.1. <emaneHost> Element

The <emaneHost> element is used to associate which computing platform in an EMANE emulation system is hosting user application of protocol processes for a named <Node:host> from the scenario *NetworkPlan*. The "host" attribute value of the <emaneHost> element provides a hostname or IP address of the corresponding EMANE system computing resource. In some cases (e.g. "distributed" NEM deployment), the <emaneHost> element and <emanePlatform> element described below may refer to the same computing resource.

5.2.2. <emanePlatform> Element

The <emanePlatform> element provides information about which EMANE NEM "platform" is hosting the NEM processing for a named <Node:host:interface> from the scenario *NetworkPlan*. Note that for "distributed" EMANE NEM instantiations, the <emanePlatform> and <emaneHost> will be the same computing platform while for "centralized" EMANE NEM instantions, the <emanePlatform> and <emaneHost> computing platforms will likely be different machines.

The <emanePlatform> "id" attribute value specifies the EMANE Platform identifier (1-65534) while the "host" attribute value specifies a hostname or IP address of the corresponding EMANE system computing resource (i.e. machine).

5.2.3. <emaneNEM> Element

The <emaneNEM> element identifies the EMANE NEM that is associated with a named <Node:host:interface> from the scenario *NetworkPlan*. The "id" attribute value of the <emaneNEM> element specifies the EMANE NEM identifier (1-65534) of NEM corresponding to the <Node:host:interface>.

6. Modeling System Template Documents

This section describes the format of configuration template documents for some specific modeling systems. The goal of these documents is to provide the context necessary to automate the generation of modeling system (e.g. EMANE, CORE, ns-3, etc) configuration files based upon the generic (non-system-specific) scenario description that is provided in the planning documents and/or emulation script(s). It is expected that a modestly small number of "templates" can be created for a given modeling system and/or installation to meet the needs of many different network modeling scenarios. It is also anticipated that these templates can be generated on-demand for systems that use dynamically-configured (or instantiated) clusters of machines (or virtual machines) for network emulation or simulation purposes.

6.1. *EmaneTemplate* Document Type

The *EmaneTemplate* document type provides a means to list the resources available (i.e., computing machinery and available allowed network ports and devices) for allocation to support network modeling scenarios using EMANE. Additionally, the template provides a mapping from generic named interface "type" instances in a scenario *NetworkPlan* document to the specific EMANE Network Emulation Module (NEM) definitions and default parameter sets. For example, in a *NetworkPlan*, a Node interface might reference (via its "type" attribute) a RADIO type and the *EmaneTemplate* can provide a mapping from this to a corresponding EMANE NEM definition (e.g. "rfpipe.xml") and parameter set. The *EmaneTemplate* document can also convey whether specific NEM types are to be run in a "distributed" or "centralized" deployment configuration and limits can be set on the number of NEMs instantiated per computing platform to assist in resource management.

It is expected that a modest number of *EmaneTemplates* or variants will be needed for a given installation of EMANE. Note that it is possible that these *EmaneTemplate* documents may be dynamically generated to reflect a suite of virtual

machines that have been instantiated to support an EMANE experiment. Alternatviely, a control system that dynamically instantiates virtual machines to support EMANE experiments MAY instead reference the *EmaneTemplate* and/or resulting *EmulationDirectory* (see above) to determine the set of virtual machines needed. Further refinement of the *EmaneTemplate* and *EmulationDirectory* document types may be accomplished to reflect the requirements of virtual EMANE machine configuration, deployment, and management.

The principal components of the *EmaneTemplate* document type are:

- Host, Port, and Device Pool definition and reference elements
- Platform templating element
- Transport configuration templating element
- NEM templating element

Note that the EMANE Platform, Transport, and NEM templating elements all can contain <ParamTemplate> sections that allow "pass-through" conveyance of the <param> element type used in EMANE configuration. However, in the case of NEM <param>, there are some "common" parameter type that MAY be overridden by corresponding *NetworkPlan* <interface> parameters, when given.

6.1.1. <HostPoolDefinition>, <PortPoolDefinition>, and <DevicePoolDefinition> Elements

The *EmaneTemplate* document type can include top level <HostPoolDefinition>, <PortPoolDefinition>. and <DevicePoolDefinition> elements that are used to define named sets ("pools") of hosts (identified by name or address), port numbers, and network interface devices (identified by name) that may drawn upon for assignment of various EMANE processes (e.g., NEM "platforms", transport daemon instances, etc). The sets can be expressed as comma-delimited lists of ranges or individual host name/addresses, port numbers, or device names. Ranges are expressed as pairs of names, numbers, or network addresses (IPv4 or IPv6) delimited by either a dash '-' character of ellipsis '...' (thre periods). When ranges are specified for names, the start of the range and end of the range MUST include an embedded numeric index and have consistent prefixes and suffixes (e.g., "host1-host23").

These "pools" are assigned an identity via their "name" attribute. IMPORTANT: All defined pools MUST contain mutually-exclusive sets of values. These pools may be subsequently referenced in *EmaneTemplate* <PlatformTemplate> and <TransportTemplate> instances to indicate the computing resources which may be assigned to resultant "platform" or "transport" instances needed to implement a given scenario against the template. Note that different <PlatformTemplate> and <TransportTemplate> MAY reference the same "pools" when common resources (e.g. hosts or port numbers) can be used for multiple purposes.

The <DevicePoolDefinition> is provided for the special EMANE "transraw.xml" transport type that binds a "raw" physical interface (i.e., device) to a given NEM transport instance. Thus, EMANE host platforms with multiple interfaces can be configured to use some of their interfaces for a "raw" connection to an external system or device.

The following XML text provides some examples of these pool definitions. Note that each <HostPoolDefinition>, <PortPoolDefinition>. and <DevicePoolDefinition> may contain one or multiple respective <hosts>, <ports>, and <devices> child elements with text content that provides the comma-delimited sets of ranges and/or items identified by name, address, or number depending upon the context. Thus a "pool" can be constructed by the logical concatenation of multiple lists of individual items or "ranges" of names, addresses, or numbers.

```
<HostPoolDefinition name="emaneMachines">
    <hosts>emane1,emane3,emane5-emane20</hosts>
    <hosts>192.168.1.1-192.168.1.31</hosts>
</HostPoolDefinition>

<PortPoolDefinition name="emanePorts">
```

```
        <ports>12000-13000</ports>
</PortPoolDefinition>

<DevicePoolDefinition name="ethDevices">
    <devices>eth1-eth8</devices>
</DevicePoolDefinition>
```

Note that an *EmaneTemplate* document can contain multiple pool definitions of each type and the referencing elements (e.g., <PlatformTemplate>, <TransportTemplate>, etc instances) may reference multiple pool definitions. The pools can be referenced by <HostPool>, <PortPool>, and <DevicePool> child elements where the element text contains the "name" of the reference pool.

6.1.2. PlatformTemplate Element

The <PlatformTemplate> is the principal top level element of *EmaneTemplate* documents. The <PlatformTemplate> and its components (e.g. the <TransportTemplate> and <NemTemplate> elements described below) define how an EMANE system (instantiated on computing hosts listed in the <HostPoolDefinition> elements described above using the port number and device resources given by <PortPoolDefinition> and <DevicePoolDefinition> listings) is to be configured in realizing scenarios derived from *NetworkPlan* and other planning documents.

Skeleton EmaneDocument and <PlatformTemplate> structure:

```
<EmaneTemplate>

    <HostPoolDefinition> ... </>
    <PortPoolDefinition> ... </>

    <PlatformTemplate>

        <HostPool> ... </>
        <PortPool> ... </>

        <TransportTemplate> ... </>

        <NemTemplate> ... </>

        <NemTemplate> ... </>

        <NemTemplate> ... </>

    </PlatformTemplate>

    ...

<EmaneTemplate>
```

The <PlatformTemplate> has the following attributes defined:

1. OPTIONAL "nemlimit" to indicate the maximum number of NEMs instantiated per computing host.

The <PlatformTemplate> "nemlimit" attribute is used to set a limit on the number of NEMs that may be instantiated on a computing host (EMANE platform). It is expected this would be applied when "centralized" NEM processing is performed and that CPU performance limitations dictate that a limited number NEMs can realistically be executed per computing host. Appropriate "nemlimit" values depend upon the complexity of the EMANE NEM being implemented and the processing capabilities of the host machines being used.

The <PlatformTemplate> has the following child elements:

1. One or more <HostPool> elements referencing <HostPoolDefinition> instances

2. One or more <PortPool> elements that reference <PortPoolDefinition> instances

3. OPTIONAL "default" <TransportTemplate> child

4. One or more <NemTemplate> elements

The <HostPool> element text content MUST contain a reference to named <HostPoolDefinition>. Configuration tools will use this reference to assign computing hosts to NEM execution for NEMs associated with the given <PlatformTemplate>. In the case of "distributed" NEMs, these hosts will also be used to perform end system functions (e.g. application and protocol execution). At least one <HostPoolDefinition> MUST be referenced, but multiple <HostPool> children can be included within a <PlatformTemplate> to reference multiple <HostPoolDefinition> instances if needed.

The <PortPool>element text content MUST contain a reference to a named <PortPoolDefinition>. The reference list of port numbers and/or ranges are used for assignment (on a per-host basis) of network ports for use by EMANE processes (e.g. NEMs, transport daemon endpoints, etc). Note that, unless a <TransportTemplate> child provides overriding <PortPool> reference(s) of its own, the <PlatformTemplate> port pool references are used for both NEM and transport endpoint connections in EMANE configuration.

A <PlatformTemplate> MAY OPTIONALLY contain a <TransportTemplate> child element to provide a "default" model for EMANE transport management and daemon instantiation. The details of the <TransportTemplate> element are described in Section 6.1.3, “TransportTemplate Sub-Element”. However, note that it is the configuration of the <TransportTemplate> for a platform or its NEMs that determine whether "distributed" or "centralized" deployment of a particular NEM is observed.

Finally, a <PlatformTemplate> MUST contain one or more <NemTemplate> child elements that provide the "mapping" from *NetworkPlan* <interface> "type" to a specific EMANE NEM model. The <NemTemplate> can also contain a <ParamTemplate> section to establish a set of default NEM parameters for a particular modeled interface device "type".

An *EmaneTemplate* document MAY contain multiple <PlatformTemplate> instances as needed to assign the execution of different NEM types to specific computing hosts, specify a mix of "distributed" and "centralized" NEM execution, etc.

6.1.3. TransportTemplate Sub-Element

The <TransportTemplate> is available as a sub-element of either a <PlatformTemplate> or an <NemTemplate> instance. No more than single <TransportTemplate> MAY be contained in each <PlatformTemplate> or <NemTemplate> instance. When a <TransportTemplate> is a sub-element of a <PlatformTemplate>, it provides a "default" model for EMANE transport instances for the NEM instances associated with resultant "platforms". A <TransportTemplate> child of an <NemTemplate> provides an overriding model for the transport endpoint associated with the resultant NEM instances. Note that a <TransportTemplate> MUST be defined for a <PlatformTemplate> or all of its <NemTemplate> children such that a transport instantiation model is established for any NEMs created.

A <TransportTemplate> has the following attributes:

1. A single REQUIRED "definition" attribute

The <TransportTemplate> MUST have a "definition" attribute specified which value conveys the corresponding EMANE transport definition file (e.g. "transvirtual.xml", "transraw.xml", etc).

The <TransportTemplate> has the following child elements:

1. OPTIONAL, multiple <HostPool> references

2. OPTIONAL, multiple <PortPool> references

3. OPTIONAL <DevicePool> references (REQUIRED for "transraw.xml" definition)

4. OPTIONAL, multiple <ParamTemplate> sections

The <TransportTemplate> can contain OPTIONAL <HostPool> child elements that reference EmaneTemplate <HostPoolDefinition> instances. Inclusion of a <HostPool> implies that the NEMs for which the <TransportTemplate> provides a transport endpoint model are to implemented as "centralized" NEMs running on a separate computing platform than the transport endpoint (i.e., offloading NEM processing to a different machine than the host for which it serves as an emulated network interface). When a <TransportTemplate> omits specifying a <HostPool>, then its associated NEMs are instantiated in the "distributed" fashion where the associated NEM processes are executed on the same machine for which it serves as an emulated network interface.

The <TransportTemplate> MAY contain one or more <PortPool> subelements that reference <PortPoolDefinition> instances for port number assignments made for the remote transport endpoints. If the <PortPool> is omitted from the <TransportTemplate>, then the transport endpoints draw from the same <PortPool> references given in the parent <PlatformTemplate>. Note that "hybrid" <PlatformTemplate> specifications can be created by embedding overriding <TransportTemplate> definitions within <NemTemplate> instances where, for the <PlatformTemplate> and/or the child <NemTemplate> instances, some include <HostPool> references and some do not.

The <TransportTemplate> MAY contain a <DevicePool> specification that defines which raw network devices are available for assignment when needed. This is REQUIRED when the EMANE "transraw.xml" transport definition is specified.

The <TransportTemplate> MAY also contain a <ParamTemplate> section which provides one or more EMANE <param> element (see EMANE documentation) specifications that are passed directly through to the resultant EMANE "platform.xml" and, subsequently generated, "transport.xml" documents.

IMPORTANT: Any <PlatformTemplate> or <NemTemplate> MUST contain no more than one <TransportTemplate> child. The <TransportTemplate> child of an <NemTemplate> overrides any default transport model specified by a parent <PlatformTemplate:TransportTemplate> definition. If an <NemTemplate> does not contain a <TransportTemplate> child, then its <PlatformTemplate> parent MUST contain a default <TransportTemplate> specification.

The following abridged example illustrates a skeleton "hybrid" configuration <EmaneTemplate> with two different <PlatformTemplate> embedded definitions. The first <PlatformTemplate> allows for "distributed" "rfpipe.xml" NEMs to be instantiated from the pool of EMANE hosts entitled "emaneMachines" while the second <PlatformTemplate> dictates "centralized" deployment of "80211abg.xml" NEMs from the "wifiMachines" <HostPool>. Note that the <TransportTemplate> for this second <PlatformTemplate> references the same "emaneMachines" <HostPool> so that a scenario can include hosts that contain both "radio" ("rfpipe.xml") and "wifi" ("80211abg.xml") interfaces.

```
<EmaneTemplate>

    <HostPoolDefinition name="emaneMachines">
       <hosts>emane1-emane20</hosts>
    </HostPoolDefinition>

    <HostPoolDefinition name="wifiMachines"
       <hosts>emane21-emane25</hosts>
    </HostPoolDefinition>

    <PortPoolDefinition name="emanePorts">
        <ports>12000-13000</ports>
    </PortPoolDefinition>

    <PlatformTemplate>

        <HostPool>wifiMachines</HostPool>
        <PortPool>emanePorts</PortPool>
```

```
        <TransportTemplate definition="transvirtual.xml"/>

        <NemTemplate type="RADIO" definition="rfpipe.xml"/>

    </PlatformTemplate>

    <PlatformTemplate nemlimit="5">

        <HostPool>wifiMachines</HostPool>
        <PortPool>emanePorts</PortPool>

        <TransportTemplate definition="transvirtual.xml">
            <HostPool>emaneMachines</HostPool>
        </TransportTemplate>

        <NemTemplate type="WIFI" definition="80211abg.xml"/>

    </PlatformTemplate>

</EmaneTemplate>
```

6.1.4. NemTemplate

The <NemTemplate> is available as a sub-element of <PlatformTemplate> instances. The purpose of the <NemTemplate> is to assist in the automated mapping of a *NetworkPlan* "interface" type to a corresponding EMANE NEM definition and parameter set. The <NemTemplate> can also optionally provide an overriding <TransportTemplate> definition for its realization.

The <NemTemplate> has the following attributes:

1. REQUIRED "type" attribute that corresponds to *NetworkPlan* <NetworkDefinition> and/or <Node:host:interface> "type" designator attributes.

2. REQUIRED "definition" attribute to identify the EMANE NEM definition XML file.

3. OPTIONAL "group" attribute that can be set to true or false to indicate whether (or not) a groupid <param> should be generated for that NEM type where corresponding <interface> instances referencing the same <NetworkDefinition> will use a common groupid <param> value. When omitted, the "group" attribute has an assumed value of false (i.e., a groupid <param> is not generated).

The <NemTemplate> MUST have a "type" attribute that specifies a *NetworkPlan* "interface type" name. Additionally, it MUST have a "definition" attribute specified which value conveys the corresponding EMANE NEM definition file (e.g. "rfpipe.xml", "80211abg.xml", etc). These two attributes provide the "mapping" from a *NetworkPlan* <interface> "type" to a specific EMANE NEM model definition. The "group" attribute can be set to true to indicate that NEMs representing *NetworkPlan* <interface> instances referencing a common <NetworkDefinition> should each include a groupid <param> child with a common group identifier value. This allows NEMs representing "wired" interface types (e.g. Ethernet) to be logically "connected".

The <NemTemplate> has the following child elements:

1. OPTIONAL <TransportTemplate> definition to override parent <PlatformTemplate> 'default' transport model, if applicable

2. OPTIONAL <ParamTemplate> section to convey default NEM parameters passed into generated EMANE XML configuration documents. Note that it is possible for the associated *NetworkPlan* <interface> to specify common parameters such as transmit "power", radio transmission "frequency", etc to override the corresponding templated parameters.

The <NemTemplate> MAY have a <TransportTemplate> child which defines the transport endpoint model for the NEM. If it is not included, the parent <PlatformTemplate> MUST have a default <TransportTemplate> specified.

The <NemTemplate> can also contain a <ParamTemplate> section which provides one or more NEM <param> element specifications that define parameter values for the resultant NEMs realized. Note that it is possible that *NetworkPlan* <Node:host:interface> definitions may have their own <param> elements that can override the values given in the <NemTemplate>. This may done for more general parameters deemed as prototypical for network interfaces, including wireless systems. Examples include transmit "power", data "rate", transmit "frequency", etc. These parameters are described in XXX. The following is an example <NemTemplate> section that omits the optional <TransportTemplate> child element, but includes a <ParamTemplate> sections with some basic parameters:

```
<NemTemplate type="RADIO" definition="rfpipe.xml">
    <ParamTemplate>
        <param name="bitrate" units="kbps" value="1000"/>
        <param name="txpower" units="dBm" value="46"/>
        <param name="frequency" units="MHz" value="4200"/>
    </ParamTemplate>
</NemTemplate>
```

6.1.5. Example EmaneTemplate Documents

This section provides a couple of simple examples of the *EmaneTemplate* document type. These illustrate "distributed" and "centralized" EMANE configurations. However, "hybrid" configurations can also be achieved if some of the <PlatformTemplate> or <NemTemplate> definitions contain <TransportTemplate> instances that reference a <HostPoolDefinition> (i.e., a <TransportTemplate> that references a <HostPoolDefinition> implies the associated NEM should be instantiated in the "centralized" fashion where the NEM executes on a separate emulation machine than the remote endpoint of its associated transport daemon. In other words, the NEM execution is off-loaded from the machine for which it serves as a network interface).

The following XML text provides approximately the shortest possible example of an *EmaneTemplate* document for a fully "distributed" configuration where EMANE NEM "platform" and "transport" functionality are co-located on the physical hosts and thus will directly correspond to "Node" instances from a *NetworkPlan* document. In this example, four NEM types are defined that specifies use of a couple of EMANE models to implement *NetworkPlan* "interfaces" that are of type "WIFI", "WIMAX", "RADIO", and "ETH". Different EMANE models and/or parameter sets are set and references to realize appropriate communication behaviors. Note that "ETH" <NemTemplate> uses the EMANE "commeffects.xml" definition and has it "group" attribute set to true so that "ETH" interfaces attached to the same NetworkPlan <NetworkDefinition> have a common groupid <param> value (i.e. those Ethernet interfaces attached to the same "net" will be linked together).

```
<EmaneTemplate>

    <HostPoolDefinition name="emaneMachines">
        <hosts>emane1-emane25</hosts>
    </HostPoolDefinition>

    <PortPoolDefinition name="emanePorts">
        <ports>12000-13000</ports>
    </PortPoolDefinition>

    <PlatformTemplate>
        <ParamTemplate>
            <param name="otamanagerfroup" value="224.1.2.8:45702"/>
            <param name="eventservicegroup" value="224.1.2.8:45703"/>
        </ParamTemplate>

        <HostPool>emaneMachines</HostPool>
        <PortPool>emanePorts</PortPool>

        <TransportTemplate definition="transvirtual.xml"\>
```

```
        <NemTemplate type="WIFI" definition="80211abg.xml" >
            <ParamTemplate>
                <param name="bitrate" units="kbps" value="1000"/>
                <param name="txpower" units="dBm" value="23"/>
                <param name-"frequency" units="MHz" value="2400"/>
            </ParamTemplate>
        </NemTemplate>

        <NemTemplate type="WIMAX" definition="rfpipe.xml" >
            <ParamTemplate>
                <param name="bitrate" units="kbps" value="5000"/>
                <param name="txpower" units="dBm" value="37"/>
                <param name-"frequency" units="MHz" value="3500"/>
            </ParamTemplate>
        </NemTemplate>

        <NemTemplate type="RADIO" definition="rfpipe.xml" >
            <ParamTemplate>
                <param name="bitrate" units="kbps" value="1000"/>
                <param name="txpower" units="dBm" value="46"/>
                <param name-"frequency" units="MHz" value="4200"/>
            </ParamTemplate>
        </NemTemplate>

        <NemTemplate type="ETH" definition="commeffects.xml" group="true"/>

    </PlatformTemplate>

</EmaneTemplate>
```

The following example illustrates a slightly more complex *EmaneTemplate* that describes a configuration of "centralized" NEM platforms that can run on a subset of the available EMANE cluster while "transport hosts" are specified in a separate <HostPoolDefinition>. The "transport hosts" in this case will actually correspond to the "Node" instances from a *NetworkPlan* document. So, in this example, the NEMs will be instantiated on the "nemMachines" of "emane1" through "emane10" with a maximum of ten NEMs per platform while the associated transport endpoints for the NEMs will be assigned to machines in the range of "emane11" through "emane25". This configuration of ten NEM platform machines with up to ten NEMs each means a maximum of 100 "interfaces" are allowed with up to 15 "Node" instances (assigned to the "transportMachines" set) in a mobile network scenario.

```
<EmaneTemplate>

    <HostPoolDefinition name="nemMachines">
        <hosts>emane1-emane10</hosts>
    </HostPoolDefinition>

    <HostPoolDefinition name="transportMachines">
        <hosts>emane11-emane25</hosts>
    </HostPoolDefinition>

    <PortPoolDefinition name="emanePorts">
        <ports>12000-13000</ports>
    </PortPoolDefinition>

    <PlatformTemplate nemlimit="10">
        <ParamTemplate>
            <param name="otamanagerfroup" value="224.1.2.8:45702"/>
            <param name="eventservicegroup" value="224.1.2.8:45703"/>
        </ParamTemplate>

        <HostPool>nemMachines</HostPool>
        <PortPool>emanePorts</PortPool>
```

```
        <TransportTemplate definition="transvirtual.xml">
            <HostPool>transportMachines</HostPool>
            <PortPool>emanePorts</PortPool>
        </TransportTemplate>

        <NemTemplate type="WIFI" definition="80211abg.xml" >
            <ParamTemplate>
                <param name="bitrate" units="kbps" value="1000"/>
                <param name="txpower" units="dBm" value="23"/>
                <param name-"frequency" units="MHz" value="2400"/>
            </ParamTemplate>
        </NemTemplate>

        <NemTemplate type="WIMAX" definition="rfpipe.xml" >
            <ParamTemplate>
                <param name="bitrate" units="kbps" value="5000"/>
                <param name="txpower" units="dBm" value="37"/>
                <param name-"frequency" units="MHz" value="3500"/>
            </ParamTemplate>
        </NemTemplate>

        <NemTemplate type="RADIO" definition="rfpipe.xml" >
            <ParamTemplate>
                <param name="bitrate" units="kbps" value="1000"/>
                <param name="txpower" units="dBm" value="48"/>
                <param name-"frequency" units="MHz" value="4200"/>
            </ParamTemplate>
        </NemTemplate>

        <NemTemplate type="ETH" definition="commeffects.xml" group="true"/>

   </PlatformTemplate>

</EmaneTemplate>
```

Finally, the "`emaneTemplateExample.xml`" file included in this distribution provides a "hybrid" EMANE configuration with one <`PlatformTemplate`> specifying "distributed" NEM operation for some of the NEM/interface "types" and and a second <`PlatformTemplate`> specifying "centralized" NEM operation for the "RADIO" interface type.

7. Ancillary File Formats

This section describes some ancillary file formats that are described for use where more compact representation than XML may be useful. For example, a simple, linear format that conveys node locations over time or other aspects of the emulation may be useful for logging or other purposes. These formats might also be used as intermediate or "output" formats for tools that process *EmulationScript* documents.

7.1. Emulation Event Log (EEL) Format

This file format is a linear, text file format that can be used to convey the value of properties or parameters identified by a keyword. This file allows for "events" affecting modeling system components and/or their properties that occur over time to be expressed (e.g. as a file format to "drive" event generation over time) or to be logged (e.g. as a log file format for "capturing" run-time events for replay or post-processing analysis). The EEL file is a text format consisting of lines (a.k.a. "sentences") that each contain a timestamp, some "module identifier" and an event type "keyword" that implies the format and interpretation of the remainder of the line. The "keyword" approach allows a mixture of event types to be included within an EEL file and expanded over time as needed. Tools that process EEL file may choose to process a subset of event types as needed. The format also lends itself to simple filtering by event type, module identifier, etc using commonly-available tools (e.g., "grep", etc).

The linear, time-ordered format also allows it to be incrementally processed such that even very bulky files can be handled as needed. Note that, in the interest of compactness, it is typically expected that the events included will represent "deltas" (i.e. changes) to any previously-established state. However, one could choose to have each time epoch (or at some less granular interval such as once per minute) include the complete modeling system state (e.g. all current node locations, adjacencies, etc). This would result in a more bulky EEL file but could enable processing tools to "skip" to desired sections of the file without need to process the entire file from its beginning. This specification does not dictate or preclude such either usage.

Thus, the skeleton format of lines within the EEL format is:

```
<time> <moduleID> <eventType> <type-specific fields ...>
```

The section below describe the format of the <time> and <moduleID> fields and specify currently-defined <eventType> values and their respective "sentence" formats. Note that event lines are delimited by end-of-line characters including '\n' and/or '\r' (line feed and/or carriage return). Blank lines are permitted and should be ignored. Also, a '#' character as the first non-whitespace character of a line indicates a comment and the line should be ignored.

The individual fields of lines are typically delimited by white space and thus white space characters in fields (e.g. <moduleID> strings) is discourage, but strings that contain white space can be include if they are encapsulated in double quote '"' characters. Note that *no nesting* of double quotes or encapsulated strings that extend beyond multiple lines are permitted in this format. The <eventType> keywords SHALL NOT be defined that include white space characters. Note that the type-specific content of "sentences" MAY specify additional delimiters such as commas or other characters, but MAY NOT allow multi-line content.

A maximum line length of 256 characters (including end-of-line delimiters) MUST be observed for EEL files.

The following provides a simple example of an EEL file excerpt with some "location" and "pathLoss" events:

```
0.0 nem:1 location gps -76.950000,38.800000
0.0 nem:2 location gps -76.950000,38.820000
0.0 nem:1 pathLoss nem:2,90.0
1.0 nem:1 location gps -76.950103,38.800041
1.0 nem:1 pathLoss nem:2,89.0
2.0 nem:1 location gps -76.950205,38.800082
2.0 nem:1 pathLoss nem:2,88.0
3.0 nem:1 location gps -76.950308,38.800123
3.0 nem:1 pathLoss nem:2,87.0
4.0 nem:1 location gps -76.950410,38.800164
4.0 nem:1 pathLoss nem:2,86.0
```

7.1.1. <time> Field

The <time> value is a floating point number in units of seconds with respect to the start of emulation (or simulation) execution. Note that negative values MAY be allowed to reflect ordering of configuration-related parameters prior to execution. I.e., the negative time value simply reflects ordering and not actual time. A <time> value of "-10.0" would indicate a configuration event preceding one with a <time> value of "-9.0" but those configuration events could be executed in rapid succession rather than waiting 1.0 second between. It is possible that future revision of this file specification may allow for additional <time> formats if deemed necessary.

7.1.2. <moduleID> Field

EEL lines contain a <moduleID> fields that is a string may have system-specific interpretation and/or formats. The colon character ':' is reserved as a delimiter for potential hierarchical <moduleID> or other named identifier fields. The comma character ',' is reserved as a delimiter for multi-part values (e.g. "x,y,z" coordinates, etc). Thus, <moduleID> and property or parameter names SHOULD not contain these characters. White space characters ' ' are also reserved as overriding field delimiters for this format, but names MAY be encapsulated within double quotes (e.g., "my name")

such that spaces (and even commas or colons) MAY be used within names. However, for clarity, use of spaces and other reserved characters is discouraged where possible.

By default, tools that process EEL files should treat the <moduleID> simply as a "string" for some processing purposes. However, some system-specific parsers may choose to interpret any hierarchical (or other) naming convention that is used for the <moduleID>. This specification does suggest some specific naming conventions for some modeling systems such as EMANE.

7.1.2.1. Default *EmulationScript* <moduleID> Format

When an EEL file is generated from an *EmulationScript* document and no mapping from the "Node:host:interface" naming scheme to modeling system-specific identifers (e.g. EMANE NEM IDs) via an *EmulationDirectory* or other means, the <moduleID> SHOULD use the colon-delimited "Node:host:interface" name. However, when the "host" portion has a value of "`default`", that portion is optional (i.e. a format of "Node:interface" MAY be used for a more succinct <moduleID>). Also, if no time zero <`Node:interface`> Events are included in the EmulationScript, then only the "Node" name will be used as the <moduleID>. By adhering to this hierachical naming convention, visualization tools (e.g. SDT) can be driven by EEL files and intelligently determing appropriate "Node" instances and display multiple links properly when Nodes possess multiple "host" sub-systems and/or interfaces.

7.1.2.2. EMANE <moduleID> Format

The <moduleID> format for EEL files containing events for the EMANE system should use the hierarchical naming convention of "<moduleType>:<id>". Supported EMANE <moduleType> values include "`nem`" and "`platform`". Most typically, event in EEL files will pertain to EMANE NEM instances and the "nem" <moduleType> value will be used. This convention allows additional EMANE <moduleType> keywords to be defined in the future and in cases where the EMANE framework is just part of a modeling system using multiple subsystems (e.g. CORE emulation, ns-3 simulation-in-the-loop, etc), this richer naming convention (as compared to just using a numeric <moduleID>) will allow for distinction among subsystem types. Similar formats will be developed for support other modeling systems within the EEL format.

Note the example given above uses the EMANE "`nem`" <moduleID> format.

7.1.3. "location" Events

For mobile network modeling, one of the most commonly occurring dynamic events will be changes in location as nodes move. The "location" event provides a means for specifying (or recording) this information in a few different coordinate system types. The skeleton format of EEL "location" event lines is:

```
"<time> <moduleID> location <locationType> <a>,<b>[,<c>[,{msl | agl}]]"
```

In this format, the <locationType> keyword identifies the coordinate system, and the three fields <a>, <b>, and <c> correspond to coordinate values for the given coordinate system. Note that that the third coordinate is optional as it represents the z-axis for Cartesian coordinates or altitude (in meters) for Geodetic (`gps`) coordinates. A fourth optional, comma-delimited field can specify a keyword value of `msl` (mean-sea-level) or `agl` (above-ground-level) to specify the intended interpretation of the z-axis or altitude value if it is given. When this fourth keyword field is NOT specified it is RECOMMENDED the default altitude interpretation be `agl` (above-ground-level). When the altitude (or z-axis) is omitted, a default value of "0.0,agl" should be assumed.

Supported <locationType> values include:

gps Geodetic coordinates of "<latitude>,<longitude>,<altitude>" where <altitude is in units of "meters".

cart Cartesian coordinates of "<x>,<y>,<z>". No specific units are assumed, but if translation to/from "gps" coordinates is supported, units of "meters" should be assumed unless otherwise specified.

utm Universal Transversal Mercator (UTM) coordinate system (support for this is TBD).

The following is a simple example of a EEL "location" event using the "gps" coordinate system to convey the location for "nem:12" at 23.0 seconds into the experiment:

```
23.0 nem:12 location gps -74.123,38.456,1000.0,agl
```

7.1.4. Propagation "pathLoss" Events

Radio frequency (RF) propagation loss is a significant determining factor that affects communication quality and range in mobile, wireless network systems. A number of factors can contribute to the propagation loss and the calculation of it can be complex depending upon the desired fidelity and degree of environmental information used. To support real-time operations via a priori computation of path loss for a scenario, or to log loss values computed in real-time during experiment execution, the EEL format provides the "pathLoss" event type to convey or record this information. In this case, the normative <moduleID> field is the assumed transmission "source" for a list of one or more path loss values (in decibels (dB)) to corresponding "destination" <moduleID> receivers. The skeleton format for the"pathLoss" event "sentence is:

```
<time> <moduleID> pathLoss <dstModuleID1>,<dB>[,dBrev] <dstModuleID2>,<dB>[,dBrev] <dstModuleID3>,<dB>[,dBrev] ...
```

Note that the event-specific content of this line includes a space-delimited list of fields where each field is of the comma-delimited format of:

```
<dstModuleID>,<loss dB>[,<reverse loss dB>]
```

The <dstModuleID> is module identifier to which the given path loss value applies. The <loss dB> is the path loss in decibels (dB). An optional third field of <reverse loss dB> can be provided when asymmetric propagation loss occurs and this value expresses the "destination-to-source" path loss. If it is omitted, the propagation loss can be assumed to be symmetric. Note that the loss values in a "pathLoss" event MAY or MAY NOT include any transmitter or receiver antenna gains or other system gains or losses. It is expected that the "pathLoss" values given for a specific module type will or will not include such additional system gains or losses as appropriate.

Note that the list given for a single "pathLoss" event SHOULD NOT be assumed to be comprehensively inclusive for the given "source" <moduleID>. I.e., multiple "pathLoss" lines MAY be used to express the path loss values for an entire set of adjacencies to the given "source". This may sometimes be necessary for processes to observe the maximum EEL line length of 256 characters per line. It is RECOMMENDED that, if no "pathLoss" event is ever expressed (i.e. in an entire EEL or set of EEL files) for a particular <moduleID> pair, then the path loss SHOULD for that pair be assumed to be infinite.

The following lines provide path loss information for a "source" module "`nem:12`" to eight of its neighbors:

```
23.0 nem:12 pathLoss nem:1,62 nem:4,43 nem:8,57.2 nem:3,19.0
23.0 nem:12 pathLoss nem:13,73 nem:14,25.4 nem:18,83.62 nem:23,43
```

Note in this example that the path loss information is given on two different lines. If this is the first line that references path loss values for "`nem:12`" then the path loss for any unlisted neighbors to/from "`nem:12`" should be assumed to be infinite up to this point in time in the experiment.

7.1.5. Module "address" Events

In network modeling, network addresses are associated with interface devices modeled in experiment scenarios. In many cases, address assignments are fixed, but it is possible that address assignments may be dynamic during execution of a scenario. It is desirable to have events that can set or record such address configurations or changes. It is also possible that multiple addresses per interface may be configured, including addresses of different protocol layers, families, or functional purposes. The following "address" event type "sentence" skeleton allows a list of one or more addresses for a given <moduleID> to be set, updated, or removed.

```
<time> <moduleID> address <addrType1>,<addrValue1>[,{add|remove|replace}] <addType2>,<addr-
Value2>[,{add|remove|replace}] ...
```

The event-specific content is a space-delimited list of item that specify the addition, removal, or replacement of address assignments for the given <moduleID>. Each list item is a comma-delimited tuple in the form of:

```
<addressType>,<addressValue>[,{add|remove|replace}]
```

where the first field, <addressType>, specifies the type of address contained in the second field, <addressValue>. The third, optional, field in the tuple is a <command> field to indicate how the address type/value is applied to the addresses associated with the given <moduleID>. When the command is "`add`", the given address type/value is added to a list of addresses associated with the <moduleID>. And when the command is "`remove`", the given address type/value is removed from the same list. When the command value is "`replace`" (and this is the default command assumption when no command field is specified), the given address type/value is to replace any address(es) in the list of the same <addressType> value. Note the <addressType> implies which protocol layer to which the <addressValue> applies.

Supported <addressType> values that imply specific <addressValue> formats include:

ipv4 Indicates the <addressValue> is a layer-3 IPv4 address in dotted decimal notation. A "slash" character '/' and numeric value in the range 0 to 32 MAY be appended at the end of the address portion to convey an associated address prefix length (in bits) for the given address.

ipv6 Indicates the <addressValue> is a layer-3 IPv6 address in the usual colon-delimited text format. A "slash" character '/' and numeric value in the range 0 to 128 MAY be appended at the end of the address portion to convey an associated address prefix length (in bits) for the given address.

mac Indicates the <addressValue> is a layer-2 Media Access Control (MAC) address. This will usually be a colon-delimited set of 6 hexadecimal values (48-bit) to convey a IEEE 802 address, but may also include other formats such as the similar EUI-64 format, if applicable, or simply a decimal value. The interpretation is context and/or system specific.

A special <addressValue> of "none" is RESERVED to indicate (when the "`replace`" command is used or implied) that address assignment(s) of the corresponding <addressType> should be removed (i.e. dissassociated with the given <moduleID>).

The following example EEL lines illustrate some of the possible uses of the "address" event type:

```
# This associates a single IPv4 address with "nem:12"
# (All previously-assigned IPv4 addresses are removed
#  since "replace" is implied by command omission)
0.0 nem:12 address ipv4,192.168.1.1

# This "adds" an IPv6 address association to "nem:12"
1.0 nem:12 address ipv6,2001:0db8:85a3:08d3:1319:8a2e:0370:7334,add

# This "removes" _all_ IPv4 and IPv6 address assignments from "nem:12"
2.0 nem:12 address ipv4,none ipv6,none

# This adds two IPv4 addresses to "nem:12"
5.0 nem:12 address ipv4,192.168.1.2,add ipv4,192.168.100.2,add
```

7.1.6. Module "param" Events

A general purpose "param" event type is provide that can be used to list and express one or more parameters and associated values for a given <moduleID>. The event-specific content is a space-delimited list of comma-delimited "<name>,<value>" tuples each with an optional third comma-delimited "units" field. Thus, the skeleton format for the "param" event "sentence" is:

```
<time> <moduleID> param <name1>,<value>[,units] <name2>,<value>[,units] <name3>,<value>[,units]
..."
```

The "name" portion MUST uniquely identify a parameter associated with the component identified by the <moduleID>. It is further expected that the named parameter has single value that is set (or overridden) by the occurrence of a "param" event. Additional EEL event types may be defined in the future for parameters that can have multiple values to support addition/removal of values as needed. This "param" event allows EEL files to contain parameter information for which a "fully-defined" EEL event type has not been specified. In some cases, "fully-defined" EEL event types may be specified in the future for parameters that currently can be conveyed only in the "param" event type. Some example parameters include radio transmission "frequency" and "power" and other attributes that network modeling components may possess.

In many cases, the "param" event will be used to script or log modeling system-specific parameter settings or changes. Thus, processing of "param" events in EEL files will usually be done in a system-specific context.

The following example illustrates the use of the "param" event to convey the radio transmission "txpower" and "frequency" configuration at time 0.0 for module "`nem:12`":

```
0.0 nem:12 param txpower,37,dBm frequency,2400,MHz
```

7.2. NRL Scripted Display Tool (SDT) Format

The NRL Scripted Display Tool (SDT) suite specifies a format intended for general purpose visualization, but with some focus on visualization of mobile network scenarios. When visualization of planned scenarios is required, this format may be a useful option for tools that process the document types described here. Further information on SDT can be found at http://pf.itd.nrl.navy.mil/protools/sdt.html. It should be also noted that the SDT format also essentially encompasses a "dynamic graph" format since it can establish and de-establish "link" instances among nodes over time. It also can currently support visualization of multiple links among node pairs.

7.3. MITRE Mobility Format (MMF)

The MITRE Mobility Format (MMF) is a simple (but potentially quite bulky) text file format that has been used to convey mobile node positions and pre-computed radio frequency (RF) propagation path loss and distances among the nodes. Each line of the text file is described as follows:

```
<time> <txNodeId> <rxNodeId> <loss>[/<~loss>] <dist> <txUTMx> <txUTMy> <txUTMz> <rxUTMx>
<rxUTMy> <rxUTMz>
```

The `<time>` is in integer units of seconds. The `<txNodeId>` and `<rxNodeId>` are integer node identifiers. The `<loss>` is a floating-point value that gives the path loss (in dB) from sender (*txNode*) to the receiver (*rxNode*). This path loss is assumed to be bi-directional unless the optional comma-delimited `<~loss>` is given which specifies a different path loss value from the *rxNode* to the *txNode*. The `<dist>` specifies the pre-computed distance between the *txNode* and *rxNode* locations. Finally, the remainder of the line specifies the *txNode* and *rxNode* locations, respectively, in Universal Transverse Mercator (UTM) coordinates with the "*UTMz*" value indicating altitude in meters.

Here is an excerpt from an example MMF file:

```
0 1 2 78.195327 90.132864 325900 4299310 0 325974 4299259 0
0 1 3 80.013243 100.076260 325900 4299310 0 325898 4299210 0
0 1 4 100.210723 1018.440683 325900 4299310 0 325056 4298972 460
0 1 5 95.185714 571.064644 325900 4299310 0 325593 4299167 460
0 2 3 78.197395 90.143596 325974 4299259 0 325898 4299210 0
0 2 4 100.600519 1065.186205 325974 4299259 0 325056 4298972 460
0 2 5 95.671124 603.887149 325974 4299259 0 325593 4299167 460
0 3 4 99.943938 987.635097 325898 4299210 0 325056 4298972 460
0 3 5 94.910163 553.232589 325898 4299210 0 325593 4299167 460
0 4 5 95.185686 571.062809 325056 4298972 460 325593 4299167 460
```

```
0 4 1 100.145221 1010.789275 325059 4298982 460 325898 4299309 0
0 4 5 95.223276 573.539590 325059 4298982 460 325603 4299164 460
1 2 1 78.066028 89.464490 325970 4299256 0 325898 4299309 0
1 2 3 78.197399 90.143615 325970 4299256 0 325894 4299208 0
1 2 4 100.485118 1051.127786 325970 4299256 0 325062 4298992 460
1 2 5 95.463742 589.639650 325970 4299256 0 325614 4299161 460
1 3 1 80.227050 101.315585 325894 4299208 0 325898 4299309 0
1 3 4 99.826103 974.327023 325894 4299208 0 325062 4298992 460
1 3 5 94.711128 540.699540 325894 4299208 0 325614 4299161 460
1 4 1 100.095321 1004.999068 325062 4298992 460 325898 4299309 0
1 4 5 95.267188 576.446505 325062 4298992 460 325614 4299161 460
1 5 1 95.026092 560.665946 325614 4299161 460 325898 4299309 0

...
```

8. Example Utilities

The *EmulationScript* schema distribution also includes source code for a number of useful command-line utilities that implement the network planning, system configuration, and motion planning and generation behaviors that are defined. The principal utlities include:

mp “motion planner” tool that parses MotionPlan XML documents and generates or amends existing “EmulationScript” XML. This tool is implemented in C++.

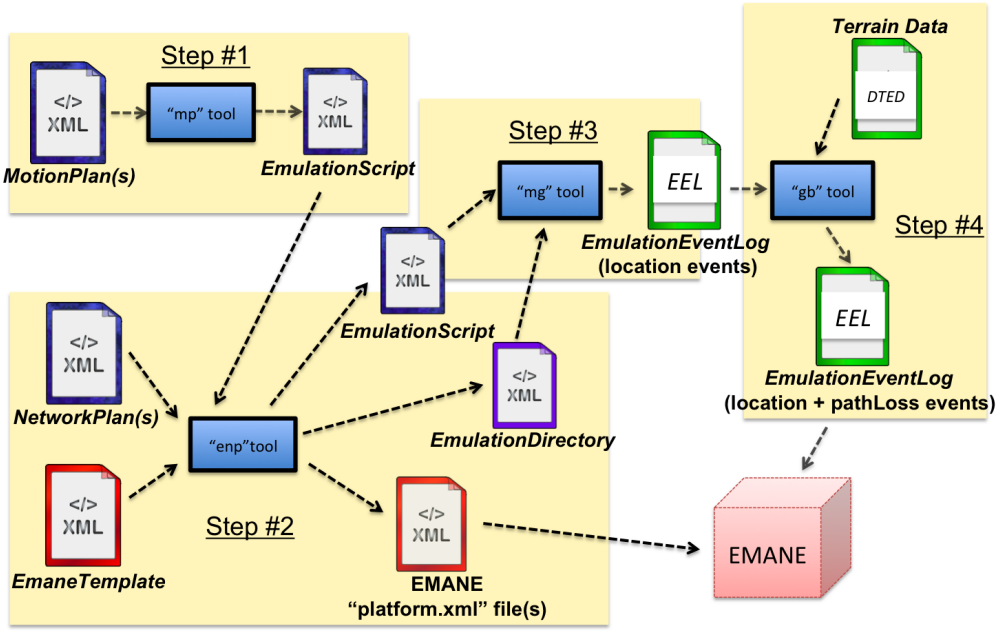
mg “motion generator” tool that parses EmulationScript XML documents and can generate location updates in various formats (SDT, EEL) at a specified interval. This tool is implemented in C++.

enp "EMANE network planner" tool that parses NetworkPlan XML document(s) and an EmaneTemplate document and generates corresponding EMANE "platform.xml" configuration files and an EmulationDirectory document. A future revision will also generate an (or amend an existing) EmulationScript document that sets <Node:[host:]interface> properties. This tool is implemented as a Python script with dependencies on the Python "netaddr" and "lxml" libraries.

gb "graph builder" tool that parses formats such as SDT, MMF, and EEL that contain location (and possibly interface property) events and references radio propagation models, etc to construct an appropriately connected (or not) dynamic graph in output formats of SDT, MMF, or EEL. For formats that support it, the "path loss" values for adjacencies are included with (or as) the graph connectivity information. This tool is implemented in C++ with dependencies on "*libgdal*" for terrain file parsing.

There are some additional tools for converting to/from some different file formats used by some mobile network modeling tools. An example tool chain that can be realized with these tools is shown in Figure 4, “Scriptools Tool Chain Example”. The accompanying "`ScriptToolsUserGuide.pdf`" document describes the usage of the *mp*, *mg* and these other utiliities. The suite of tools and their capabilities will be expanded over time.

Figure 4. Scriptools Tool Chain Example



9. Usage Notes

(TBD - Describe some example usages of the EmulationScript document types.)

10. "ToDo" List

This "todo" list is a set of items that will be addressed in future revisions of this document, the schema, and accompany tools:

- In *NetworkPlan* and *MotionPlan* documents, allow <Node> elements to be specified with a "ranged" name attribute (e.g. "node1-node50") to allow for automated enumeration of groups of nodes with equivalent Network and/or Motion properties. To support a form of "clustered" mobility for Nodes enumerated in this fashion in MotionPlan documents, a way to specify a bounded random "locationOffset" and "timeOffset" will be created. Note that a group of nodes enumerated in this fashion and using non-fixed MotionPlan:randpoint primitives or patterns patterns would assume independent motion from a common *MotionPlan* specification.

  When this Node enumeration technique is supported, it may also be useful to allow the Nodes to be subsequently referenced (by name) to modify *NetworkPlan* attributes and/or expand their respective *MotionPlan*.

- Define and implement a <follow> motion primitive that allows the location/motion of one node to be dependent upon another. The node <locationOffset> and/or <timeOffset> can come into play here. It might also be interest to define a <follow:positionOffset> where the <positionOffset> is like a <locationOffset> but the azimuth / elevation is with respect to the target node's "heading" (or "bearing"). This would let us specify group motion "formations".

10.1. Comments and Questions

This section documents some outstanding questions and comments raised as this schema is developed. This may be resolved into "todo" after some thought and discussion.

1. If an <interface> references a "net" but specifies a different interface "type", should it still inherit the parameters of the referenced <NetworkDefinition:interface>?

2. Should the <NetworkDefinition> be called <LinkDefinition> instead?