

# Metaconfiguration of the computer network

Miroslav Matuska

11.12.2004

## 1 Introduction

Computer network generally provides communication infrastructure for various important systems and services. The communication services have to fulfill their needs and thus must offer real-time capability of data transfer between network end points. The key question is realization of those services.

Today's computer networks are often complex systems of active devices and transmission media. Active devices must perform various tasks based on the type of device and its configuration; no single active device can perform the whole data transfer. The computer network offers its communication service by cooperation of all devices in it.

It is thus crucial to set up computer network active devices as an integrated system, not as the standalone devices. The current approach to device configuring one by one by the network administrator needs a lot of experience and the administrator must maintain the "big picture" of the network during the process. The change of one part of functionality often needs to be adjusted in multiple points of the network.

The Metaconfiguration concept focuses on the formalized approach to the network configuration, i.e. **to configure the functions and attributes of the whole network rather than functions and attributes of single devices**. The standard descriptive and configuration method of computer network should solve the problem with network settings inconsistencies and offers the way to automate various routine and repetitive tasks of this process.

### 1.1 Metaconfiguration - fundamentals

This document describes the network metaconfiguration system, its function and data structure. The aim of metaconfiguration is to create the system for configuring the network devices, mainly L3 ones (routers, L3 switches). The distinction against the usual approach is not to configure each device separately, but to configure the network as a whole.

The basic idea is to have data and function representation of the configuration of whole network, not just single router configurations. Having single router configurations, some information are redundant and some of them are specified by the (silently assumed) relations in the network. The metaconfiguration gathers all configuration data into one entity while removing redundancies and inferial attributes.

There is also possible to use these network metadata to provide automated operations, assistant mechanisms and "metaoperations" on metadata - though simplifying and facilitating the network design, administration, maintenance and documentation. It also enables to run various checks of the correctness of the designed network. These mechanisms provides the network designer/administrator possibility to automate the routine tasks.

Metaconfiguration system is designed to enable the single network description to be applied on various existing networks. This is realized by the means of reusable templates, which can be slightly modified by setting parameters and variables to them. There is also possible to use macro expressions to specify the attributes without the need of exact values at a design time.

The metaconfiguration system is closely bound to the CESNET Liberoouter/Netopeer project ([Lib04]), which ensures practical use of the metaconfiguration application (i.e. the final configurations could be applied on various physical boxes in the network). Metaconfiguration system is designed as one of the front-ends of the Netopeer system (front-end is the interface between the network administrator and core/backend of the Netopeer system)

The metadata representation form is in the standard XML format [W3C04], data schema language is RelaxNG [Oas01]. It is supposed that various XML standards (such as XSLT [W3C99]) and tools (both standard and newly created) facilitate the understanding and acceptance of the metadata and metaconfiguration systems.

The whole system is developed under open-source license. All development materials (specifications, papers, presentations, code) can be found in the CVS repository [Met04a] of the Liberoouter project or at the supporting website [Met04b]. Several industry standards, such as [ITU00] were studied, as well as similar research projects (i.e. [Rex03], [Net03]) in order to achieve the "big picture" of the problem area.

## **2 Terminology**

In order to achieve reader's understanding it is needed to mention the definition of some terms used in the metaconfiguration system.

**Node:** Router (L3 device) or aggregated network on lower hierarchy level

**Link:** L2&L1 devices and media used for connecting nodes (i.e. point or multipoint link).

**Binding:** conjunction of two entities (e.g. node and link)

**Network:** nodes and links that are bound to them

**Interface:** binding of a link to a node

**Feature:** simple attribute assigned to node, link or interface

### 3 Metadata schema with comments

Metadata are divided into following parts: *Topology&Addressing*, *Routing*, *Filtering/Firewalling* and *Other/Features*. The first three parts are subject to "metatisation" (redundance removal, inferiality simplification), the last one contains data which are tightly bound to specific routers from their nature (hostnames, descriptions and so on). All of them can be simplified by templates. Metadata are discussed in following sections in particular.

Current version of described metadata is 1.5

#### 3.1 Topology & Addressing data

The information about the network topology is stored in the *networks* group of elements. It contains the templates of simple networks (groups of nodes and links connecting them). One of the networks is the root one - where the network hierarchy starts and it is specified by the corresponding attribute.

```
<define name="networks-content">
  <oneOrMore>
    <element name="network">
      <ref name="network-content"/>
    </element>
  </oneOrMore>
  <attribute name="root"/>
</define>
```

```
<define name="network-content">
```

Each network template contains the basic identification (will be explained later) and the section for variable setting. Variables set in this area will be valid in the whole network template and all included nodes.

```

<ref name="identification-content"/>
<optional>
  <element name="set">
    <ref name="set-content"/>
  </element>
</optional>

```

The very important attribute is the information about inheritance - if this is original, new template (no inherit attribute) or if it inherits base network information from some other template. In this case, the template will include only the new data entities, in addition to parent template. Please distinct between *template inheriting* (this case) and *template using* (described later in the "node" section).

```

<optional>
  <attribute name="inherit"/>
</optional>

```

The following section contains the addressing information - how is the IP addressing space hierarchically divided into smaller segments. There can be one or more prefixes assigned to the network - either specified locally or delegated from the higher level of network. The prefixes defined here are later assigned to network links.

```

<optional>
  <element name="prefixes">
    <oneOrMore>
      <element name="prefix">
        <ref name="prefix-content"/>
      </element>
    </oneOrMore>
  </element>
</optional>

```

Prefix internal structure is specified here. The newly assigned prefix has its address and length of subnet mask specified here, along with its identification. If this prefix is delegated from somewhere else (from the higher level of network), only the *idref* reference is mentioned here.

```

<define name="prefix-content">
  <choice>

```

```

    <group>
      <ref name="identification-content"/>
      <attribute name="address"/>
      <attribute name="masklen"/>
    </group>
    <attribute name="idref"/>
  </choice>
  <ref name="make-subnets-content"/>
</define>

```

Each prefix can be divided into more smaller subnets. If so, the size of new subnets is recorded in the *subnetbits* attribute. This value specifies the shift if the boundary between the network and host part of the address. The newly created subnets are created under the previous *prefix* element.

```

<define name="make-subnets-content">
  <optional>
    <attribute name="subnetbits"/>
    <oneOrMore>
      <element name="subnet">
        <ref name="subnet-content"/>
      </element>
    </oneOrMore>
  </optional>
</define>

```

Each subnet has its unique number *subnetid* reflecting the order in the group of new subnets (cannot be higher than the number of the newly created subnets).

The new subnets identified by identification attributes could be furthermore divided into smaller ones and so on. The RelaxNG schema reflect this situation by specifying the *subnet* element recursive. If there is a need to split the subnet into smaller ones, the *subnetbits* attribute is used and new *subnet* elements are created under it.

```

<define name="subnet-content">
  <ref name="identification-content"/>
  <attribute name="subnetid"/>
  <ref name="make-subnets-content"/>
</define>

```

Basic topology data of the network template are stored in the *nodes* and *links* elements as one or more of *node* and *link* elements.

```
<optional>
  <element name="nodes">
    <oneOrMore>
      <element name="node">
        <ref name="node-content"/>
      </element>
    </oneOrMore>
  </element>
</optional>
<optional>
  <element name="links">
    <oneOrMore>
      <element name="link">
        <ref name="link-content"/>
      </element>
    </oneOrMore>
  </element>
</optional>
```

Each node has its unique identification and information about its graphical representation (*visualization-content* is described later in this text).

```
<define name="node-content">
  <ref name="identification-content"/>
  <ref name="visualization-content"/>
</define>
```

The node can be either usual router or nested network on a lower hierarchy level. The distinction between these two types of node is assured by the *use-network* attribute. In the router case, it is not present. In the nested network case, this attribute is a reference to network template which is nested in this node.

```
<optional>
  <attribute name="use-network"/>
</optional>
```

The *set* element provides opportunity to set variable values for this node only (not for whole network).

```
<zeroOrMore>
  <element name="set">
    <ref name="set-content"/>
  </element>
</zeroOrMore>
```

Following element creates an interface on the node - it connects the active node to some link, defined in the *links* section (see below). There should be none, one or more bindings (aka interfaces) on the node.

```
<zeroOrMore>
  <element name="bind">
```

This attribute specifies the reference to the corresponding *link* element (that link is connected into this interface).

```
<attribute name="linkref"/>
```

The name of the interface is specified here. It is either the physical identification of the interface on the router or the symbolic name of the entry point in the nested network (if the node contains another network).

```
<optional>
  <attribute name="interface"/>
</optional>
```

Host ID address part of this node on bound link is specified in the attribute *hostid*. The rest of the network address can be achieved from the connected link (via the *linkref* reference).

```
<optional>
  <attribute name="hostid"/>
</optional>
```

If there are any specific features for this interface there will be specified here (see the *features* section below).

```
<optional>
  <element name="features">
    <ref name="interface-features-content"/>
  </element>
</optional>
```

The packet filters (firewall lists, access-control lists,...) can be specified in the *filters/filter* elements. There are two possibilities of filter specification - direct set of rules (see the *filter-attributes-content* section) or with the reference to ready made packet filter template (via the attribute *use-filter-template*). The definition of filter templates will be described later in this text.

```
<optional>
  <element name="filters">
    <oneOrMore>
      <element name="filter">
        <choice>
          <ref name="filter-attributes-content"/>
          <attribute name="use-filter-template"/>
        </choice>
      </element>
    </oneOrMore>
  </element>
</optional>
</element>
</zeroOrMore>
```

We have left the *bind* element now and we are continuing with the *node* element. The last two possible elements describe the reference to the *Other/Features* data set (see this section later in the text), thus specifying the templates of the network attributes/settings, which are neither routing nor filtering nor topology nor addressing related. But these attributes are highly needed to ensure the completeness of the metaconfiguration system and the possibility to create the exports into the Netopeer system. Example of these "features" is the AAA settings, DNS settings, NTP settings and so on.

Inheritable attributes are applicable to current node and all of its children (nested) networks. Local features are only for the actual node and those are not inherited to the nested networks.

```
<optional>
  <element name="features-inheritable">
    <ref name="features-content"/>
  </element>
</optional>
<optional>
  <element name="features-local">
    <ref name="features-content"/>
  </element>
```

```
</optional>  
</define>
```

The link (apart from its identification) can include the reference to the addressing data - to its assigned subnet

```
<define name="link-content">  
  <ref name="identification-content"/>  
  <optional>  
    <attribute name="subnetref"/>  
  </optional>
```

If there are any specific features for this link there will be specified here (see the features section below).

```
<optional>  
  <element name="features">  
    <ref name="link-features-content"/>  
  </element>  
</optional>
```

Estimated hosts value can be used for statistic purposes or for tailoring the subnets size in automatic addressing tool.

```
<optional>  
  <attribute name="estimated-hosts"/>  
</optional>
```

Some routing attributes can be applied to the link also (e.g. "exclude this link from the route propagation") and such are applied here, in the routing section of the *link* element.

```
<optional>  
  <element name="routing">  
    <ref name="routing-content"/>  
  </element>  
</optional>  
</define>
```

Each network template contains the links to the information of the other areas (routing and other features). Inheritable attributes are applicable to current network and all of its children (nested) networks. Local features are only for the actual network and those are not inherited to the nested networks.

```
<optional>
  <element name="features-inheritable">
    <ref name="features-content"/>
  </element>
</optional>
<optional>
  <element name="features-local">
    <ref name="features-content"/>
  </element>
</optional>
<optional>
  <element name="routing">
    <ref name="routing-content"/>
  </element>
</optional>
```

This is the end of the network template.

```
</define>
```

### **3.2 Routing data**

This section describes data regarded to routing processes in the network and the means of their configuration on each router.

```
<define name="routing-common-content">
```

Unique identification of each routing process in the network is stored in attribute *process-id*. This ID brings together various settings for the same routing process stored in various places of the metaconfiguration (e.g. RIP settings will be common for all RIP routers in the network and some links could be excluded - but it is needed to know from which routing process they will be excluded).

```
<attribute name="process-id"/>
```

Routing data can contain reference to previously defined routing template (see below).

```
<optional>
  <attribute name="use-routing-template"/>
</optional>
</define>
```

Each routing method (protocol) has its own properties. Those will be stored in appropriate elements. Values will be extracted from the Netopeer data schema [Lho03], because they will be the same and it is no good to duplicate them. For example, the static routing method will need parameters about possibility of supernetting and next-hop types (via interface or IP address), RIP routing process will need the timer values, protocol version and so on.

It is possible that some specific routing metadata will emerge later and those will be stored here as well as newly supported routing protocols will have their elements here.

```
<define name="routing-content">
  <oneOrMore>
  <choice>
    <element name="static">
      <ref name="routing-common-content"/>
      <!--Attributes from the Netopeer system will be here -->
    </element>
    <element name="rip">
      <ref name="routing-common-content"/>
      <!--Attributes from the Netopeer system will be here -->
    </element>
    <element name="ospf">
      <ref name="routing-common-content"/>
      <!--Attributes from the Netopeer system will be here -->
    </element>
    <element name="is-is">
      <ref name="routing-common-content"/>
      <!--Attributes from the Netopeer system will be here -->
    </element>
    <element name="bgp">
      <ref name="routing-common-content"/>
      <!--Attributes from the Netopeer system will be here -->
    </element>
  </choice>
</define>
```

```
    </choice>
  </oneOrMore>
</define>
```

All routing templates have common identification and will have common routing attributes such as enable/disable state or exporting/importing rules to/from other routing protocols. Those will be added into schema later and will be located here.

```
<define name="routing-templates-common-content">
  <attribute name="id"/>
</define>
```

This routing template section contains similar properties as direct (non-templated) routing properties (see above). Majority of them will be derived from the Netopeer data schema. All of them will have the possibility to include the variables and macro expressions to extend the descriptive abilities of the routing metadata.

```
<define name="routing-templates-content">
  <oneOrMore>
    <choice>
      <element name="static-template">
        <ref name="routing-templates-common-content"/>
        <!--Attributes from the Netopeer system will be here-->
      </element>
      <element name="rip-template">
        <ref name="routing-templates-common-content"/>
        <!--Attributes from the Netopeer system will be here-->
      </element>
      <element name="ospf-template">
        <ref name="routing-templates-common-content"/>
        <!--Attributes from the Netopeer system will be here-->
      </element>
      <element name="bgp-template">
        <ref name="routing-templates-common-content"/>
        <!--Attributes from the Netopeer system will be here-->
      </element>
      <element name="is-is-template">
        <ref name="routing-templates-common-content"/>
      </element>
    </choice>
  </oneOrMore>
</define>
```

```

        <!--Attributes from the Netopeer system will be here-->
    </element>
</choice>
</oneOrMore>
</define>

```

### 3.3 Filtering/Firewalling data

The filtering metadata schema is small nowadays due to strong bindings to the Netopeer system. Definition of packet filter chain template contains of the set of packet filter rules (in the Netopeer style, this part of metaconfiguration schema will be derived from the Netopeer data schema) in conjunction of the optional element direction. It specifies if this is a inbound or outbound filter (if applied on the interface).

The packet filter rules in addition to Netopeer system can include metaconfiguration variables and macro expressions. This will facilitate creating packet conditions like "apply on all internal networks", "apply on the subnet of this link", "apply on all networks but neighboring" and so on. The result of this approach is the possibility of creation of general filtering templates applicable in various networks regardless of actual network addresses.

```

<define name="filter-attributes-content">
    <ref name="identification-content"/>
    <optional>
        <attribute name="direction"/>
    </optional>
    <!--Attributes from the Netopeer system will be here -->
</define>

```

### 3.4 Other/Features

Apart from already mentioned metadata (addressing, topology, routing, filtering) there also exist network attributes, which are not able to be easily simplified and applied on whole network. But those data can still be grouped into common templates and applied on network parts (single routers, interfaces, single links, groups of those entities and so on). For example, if there is need to specify NTP settings which will be common on all routers in the network, common NTP template applied to all routers (or to corresponding network template) can solve this problem.

The features template consists of its identification and the set of features.

```

<define name="feature-templates-content">
  <ref name="identification-content"/>
  <ref name="features-content"/>
</define>

```

The network features are one of these three types - applicable to *routers*, applicable to *links* and applicable to *interfaces* (bindings of routers to links). Each type groups different set of features - not all attributes are applicable to all entities (e.g. NTP settings has no sense for link now).

```

<define name="features-content">
  <zeroOrMore>
    <ref name="router-features-content"/>
  </zeroOrMore>
  <zeroOrMore>
    <ref name="link-features-content"/>
  </zeroOrMore>
  <zeroOrMore>
    <ref name="interface-features-content"/>
  </zeroOrMore>
</define>

```

Specific router/link/interface features will be derived from the Netopeer data schema. It would be useless to duplicate the schema here. The important difference between metaconfiguration and Netopeer will be in data fields - it is possible to include variables in metaconfiguration and thus create reusable templates. The bigger templates can be composed from the smaller ones or the smaller ones can be applied in groups. So it is possible to create one common (e.g. NTP, DNS, ...) template and to apply it on all routers where desirable. The change in this template affect settings on all assigned routers.

```

<define name="router-features-content">
  <ref name="feature-template-use-content"/>
  <!--Attributes of Netopeer system (e.g., hostname) -->
</define>

<define name="link-features-content">
  <ref name="feature-template-use-content"/>
  <!--Attributes of Netopeer system (e.g., speed)-->
</define>

```

```

<define name="interface-features-content">
  <ref name="feature-template-use-content"/>
  <!--Attributes of Netopeer system (e.g., interface description) -->
</define>

```

This element *feature-template-use* specifies the reference to feature template. It is possible to assign various kinds of features (router/link/interface) to various entities (e.g. to all objects in some network). If the features applied are not compatible with current object (e.g. if link features are applied on router), they are ignored.

```

<define name="feature-template-use-content">
  <zeroOrMore>
    <element name="feature-template-use">
      <attribute name="idref"/>
    </element>
  </zeroOrMore>
</define>

```

Some templates may include reference to template parameters (as *\$VARIABLE\_NAME* - please see the section below). If it is desirable, it is possible to specify metaconfiguration variables. Those variables (or parameters) can be substituted with real values at the end. The substitution manages this element with two mandatory attributes, *name* of the variable/parameter and its *value*. The value can contain another reference to other variable. The optional attribute is *index* in the array (applicable only if the variable is array, e.g. more memory fields with the same name distinct by index value).

```

<!--Parameters/variables common data -->
<define name="set-content">
  <attribute name="name"/>
  <attribute name="value"/>
  <optional>
    <attribute name="index"/>
  </optional>
</define>

```

The identification of various elements (as described above in the text) is defined by mandatory *id* attribute (it is used as a target of various *idrefs*). There are also

optional attributes like *name* and *description* - and they can be used at various places where more description is desirable.

```
<define name="identification-content">
  <attribute name="id"/>
  <optional>
    <attribute name="name"/>
  </optional>
  <optional>
    <attribute name="description"/>
  </optional>
</define>
```

Visualization data are not the integral part of the metadata system, but it is required by the GUI application. Its purpose is to specify layout of the network entities (nodes and links between them) and later their symbols, color - anything what will be needed.

```
<define name="visualization-content">
  <optional>
    <attribute name="vi-posx"/>
    <attribute name="vi-posy"/>
  </optional>
</define>
```

The derivation of the some metaconfiguration attributes from the Netopeer data schema mentioned above will be realized by the means of standard XML concepts and routines. It includes namespaces (metaconfiguration marks in the Netopeer schema) and XSLT stylesheets extracting specific elements and joining them with metaconfiguration schema.

## **4 Soft and hard rules**

The hard and soft rules will be formalized by two techniques - XML related tools and procedure programming languages.

Once the metadata are captured in XML form, they can be checked by standard XML tools. The simplest check is the validity verification against the schema. But complex rules cannot be included in the schema, so there is possible to

use XPath language [W3C99a] together with operators (is equal, is greater, exist and others). XPath expression specifies the location of metadata and its value can be compared with other data using operators. If this complex expression (which can include multiple XPath expressions) is not fulfilled, then the network design must be adjusted - the values must be changed, the elements must be added or removed.

If the rule is not rigid (hard), those expressions must be valuated with probability values specifying "how soft this rule is". The user can apply those soft rules (complex XPath expressions) and see the probability level of the results (i.e. if the network is built in one manner it can achieve the high level, if something is changed the level can decrease). Then the user can choose if the network design will be retained or changed. Those probabilities will also reflect specific network administrator preferences, which can differ from network to network (fastest convergence vs. routing stability, bandwidth effectiveness vs. higher security and so on).

Sample of the XPath soft rules is can look like this - If the dynamic routing (RIP for example) is enabled on some routers then:

- See if there are no stub links with route propagations enabled (70% if any present, 90% if all present)
- See how many routers are in the RIP area and if the network is not too big (40% for minimum value, i.e. 7 routers, 50% for standard value, i.e. 10 routers, 80% for 20 routers)
- See if update intervals are not too short or too long (70% for standard value ...)
- See if holddown timers are adjusted to the router count (i.e. update timer multiplied by router count in possible loops) (90% for standard value ...)
- See if RIP redistributes its routing information with neighboring routing methods and if it is not overlaying with other ones (70% for standard value ...)
- Any hard rule would be described with 100% probability

Another standard available for these purposes is Schematron assertion language [Jel02]. It is also based on XPath expressions and includes the reports which will be displayed (or for the metaconfiguration purpose the actions that should be performed). This feature will be also useful in automatic assistance mechanisms to specify what elements/attributes should be added/modified/deleted if some

situation in the schema occurs. Probability values for the soft rules apply also here.

The last instrument for hard/soft rule checking is ordinary procedural language (such as C or Java). It will process the XML data and can use all means of computing including heuristic mechanisms and artificial intelligence to produce decisions. This is the "last-resort" option for the cases when the rule is too complex to be implemented in XPath with probabilities or Schematron assertions. Therefore it is possible to implement almost any checking routine that may be needed in the future. Such example can be the rule based on real-time traffic simulation characteristics - if the network is not congested in this design, if the network fulfills the maximum end-to-end transport delay time or similar.

## 5 Variables and special macro expressions

The elements and attributes can contain either actual value or the expression, which will be substituted with real values later (at a configuration generation time). Those general expressions will be useful in templates because they will be applicable in more places of the network without the need to adjust or modify. It will also save time in specifying actual values because it will not be necessary to count or derive them extra by hand.

The basic expression in the metadata is the variable reference, indicated by the dollar sign and the variable name, like this: *\$variableA*. The variable can be used as a plain reference or as a part of complex expressions like:

```
$variableA+1  
"Administrator name: "+$variable1.
```

The complex expressions are made using usual operators. It is not needed to specify the variable type (string or integer), it will be detected automatically. The variables are defined in the set element (see the data schema above).

It is also possible to include following macros in the data fields:

- First possible value from the possible range
- Last possible value from the possible range
- The number of the occurrence of the template if it is used more than once
- The current level of hierarchy (of nesting)

- The actual number of nodes/links in the template
- The unique ID of the current entity
- and the others...

There are also designed macros for specifying prefix and link values:

- The highest level prefix
- All assigned prefixes
- All point-to point (2 host only) prefixes
- All internal links
- All border links
- All neighboring links
- All but neighboring links
- and the others...

It will be possible to include the XPath expressions referring to specific metadata in the XML tree and to specify simple *iterations* (like "repeat those values until...") and *condition* ("apply these data only if...") in case no other solution is possible.

The macro expressions are easily extensible in the need of another formalized configuration expression. It is highly possible that the needs of network administrators will require adding of various useful macros.

## 6 Functions

### 6.1 Application architecture

The applications/functions of metaconfiguration system provide the network administrator method to create, edit and delete the metadata, import them from other systems (mainly Netopeer), export them and run various checking&assistance mechanisms on them.

The core of function structure is the graphical user interface application, which comprises all basic editing functions. The user can create templates of all kinds - mainly the topological ones with nodes and links and he is able to see them in

graphical representation (e.g. ellipses connected with lines, see the GUI screen prototypes below). The GUI application integrates and calls all other functions of the metaconfiguration system, although they can be provided also as standalone "supporting" applications. All kinds of templates (e.g. routing, filtering...) have their own user interface facilitating the editing process - for example the chart of subnets distribution in assigned address space.

The second part of function structure is the sum of exporting routines into other systems. It will mainly provide the conversion of metadata into any desirable form of normal data, either configuring or documenting. The purpose of exports is to have possibility of up-to-date documentation available for network administrator or his manager without the need to frequently elaborate boring and routine descriptive tasks of managed network. Other export target is the Netopeer system - set of simple XML configurations of all routers in the network. Those configurations exported (derived) from the metadata can be applied to physical routers in the network via the Netopeer backend systems. The exporting routines will use XSLT stylesheets or procedural routines.

Importing functions from the Netopeer system is the reverse task than described in the previous paragraph. It is much more complex because there is need to extract the metadata (remove redundancies, simplify deducible elements, create common templates and so on) automatically. Therefore methods of boundary scanning, heuristic analysis and maybe artificial intelligence will come in place because no rigid approach works here. It is possible that user will have to assist procedural mechanisms in importing process.

The very important part is the automated assistance mechanisms. Those will run through metadata and will extend the design with new features in order to be correct. For example if the administrator adds new links into networks, those mechanisms offer him available addressing subnets to be assigned to them - and if there is only one option they will assign them. Its realization will be procedural and probably integrated into editing process of the GUI application (i.e. it will periodically check the metadata during template creation).

Closely related to assistance mechanisms are the checking routines which can validate the network design according to hard and soft rules. It will check if it is possible to create the functioning network from the given templates (hard rules) and if the new network will be optimized for traffic flow, routing, address space allocation and so on (soft rules).

## **6.2 Metaconfiguration use-case**

Network administrator/designer can either create new network or apply the metaconfiguration system to his existing networks.

In the first case, the admin can use ready-made topology templates from the

other networks, which will contain also the knowledge and best practices of successful network building. Automatic mechanism enables him to fill other data (such as addressing ones), otherwise which must have been added manually. Administrator can solely focus on the network design and he can omit the routine configuration tasks like subnet counting, common features copying, repetitive "inventing" of security measures - packet filters. Graphical user interface provides him the pleasant environment to deal with real creative tasks.

In the second case the importing functions will be used. If the existing network is simple, the importing system can find all needed entities and their attributes itself. It can gather all existing configurations from the physical routers using the Netopeer system and create simple templates by finding common attributes (i.e. same subnets, routing area boundaries, same DNS settings and so on). If the existing network is too complex, there may be some user input needed to achieve fundamental points of the network design and the correct approach. But the system can also use for example autonomous systems areas, packet filters protected interfaces and similarity of other router settings as important points and use them to separate flat space into hierarchic templates.

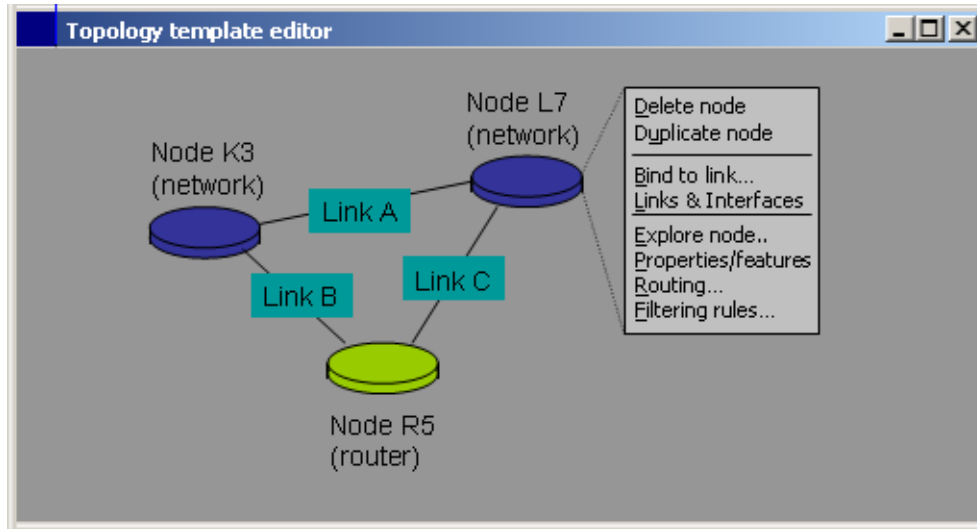
When the process of the metadata network creation is over, the administrator will need to maintain its network - change its transport characteristics if needed, replicate networks, add new branch networks, provide reports & documentation and more. All this can be achieved by editing features of the GUI (metadata changes with assistance and checking, possibility to reuse existing design from the other branch network) or the exporting routines (applying the changes on physical routers, generating documentation and various statistics and reports).

The main aim of this usage is to have the primary version of the configuration in the metadata representation and all other data forms including physical router configuration will be derived from it. There will be no need to synchronize data from various sources and maintain integrity among them. User friendly representation of all kinds of data will shorten the time needed for network design and maintenance. It also enables to work with network data not only experienced administrators but also less qualified ones, thus sharing knowledge and saving needed human resources.

## **7 GUI prototypes**

The key factor of the acceptance of the metaconfiguration system among current network administrators is the user-friendliness and intuitivism of the applications. In order to achieve that goal, some screen prototypes were designed and made available to further discussion. Those prototypes depict various situations and template types editing.

This is a projected design for topology editor. You can see simple triangle network with three point-to-point links and three nodes. References into various other template editors will be realized by the means of properties menu - node menu is displayed here

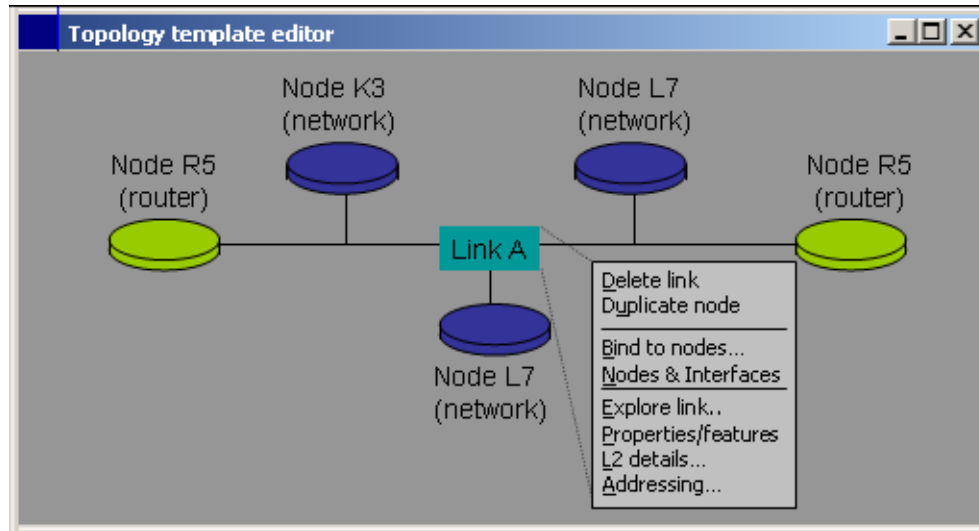


**Figure 1:** The topology editor with triangle network

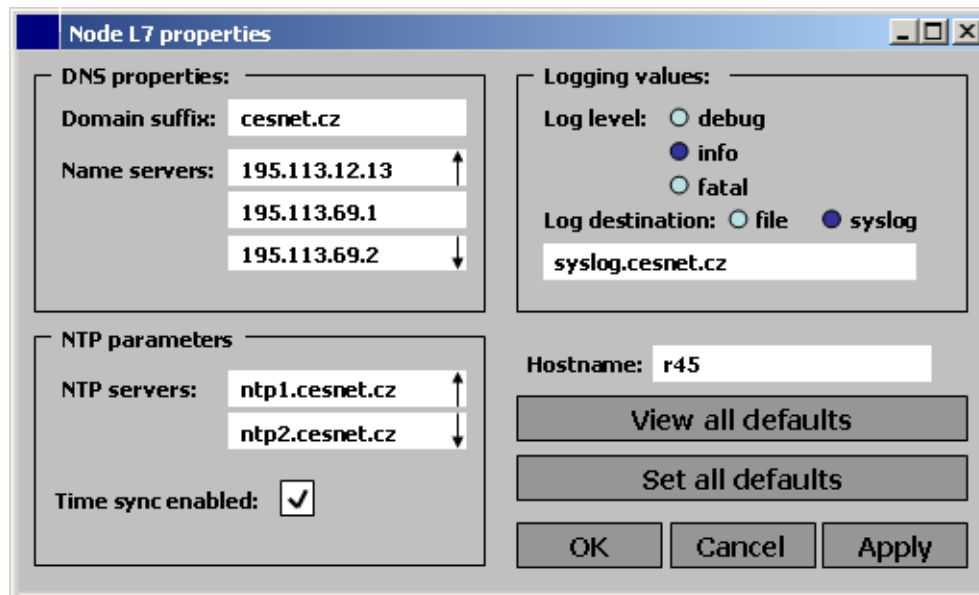
The next picture is similar example of the topology editor with one multipoint link and five nodes. Two of them are router type, three of them are nested network type - they include other network templates in them. Those nested templates can be displayed in new window. The link has specific properties also - see the link menu.

The node properties window (details) can look like this. Specific feature templates (created elsewhere, see below) are put together into one bigger feature set. Its specific values regarding to chosen router could be modified here. The network administrator can return the to original default values. Various types if input fields (text box, checkbox, radio buttons, combo box) can be present here. Those will reflect the nature of the data element or attribute.

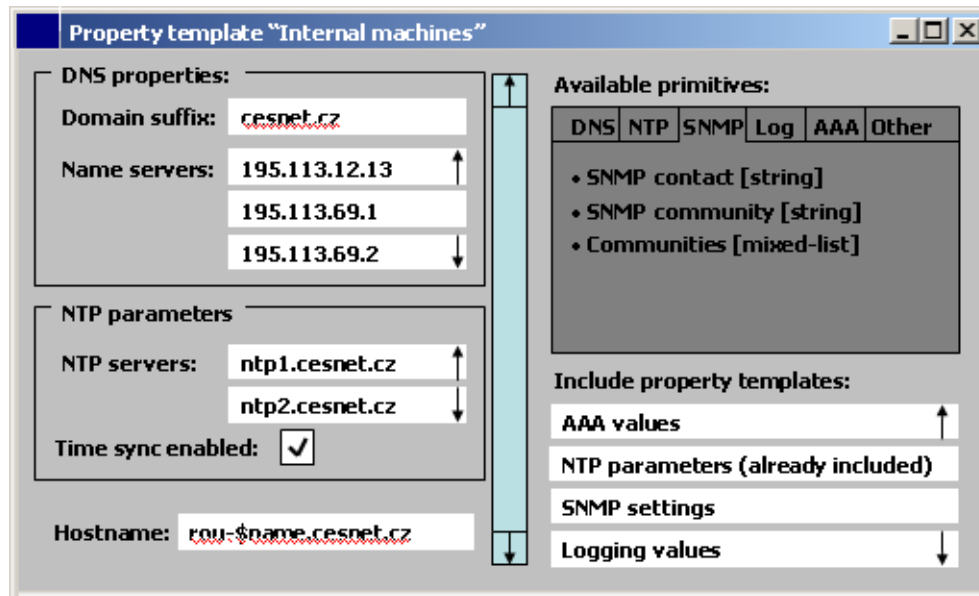
Features template editor is presented here. In the left side of the window there is list of so called "primitives" which are basic building blocks of the feature sets. They are corresponding to simple elements and attributes from the metadata schema. The small sets of those primitives (grouped by common purpose) can be further grouped into bigger sets and therefore very complex feature templates can be created. The values entered here will be taken as the default ones and they can include variables and macro expressions. Those templates will be assigned to various entities like routers, links or interfaces - in a manner presented above.



**Figure 2:** The topology editor with the multipoint link



**Figure 3:** The node properties window



**Figure 4:** Features template editor

Existence of variables and macro expressions can make configuration system very complex and incomprehensive which is undesirable. The variables summary is needed - which variables were inherited from the topology template, which variables are applied from the upper level of network hierarchy and which variables were created locally in this node. The important fact is which variable is currently valid if more than one type of variable is present - the actual value is in last column.

The first draft of the routing template editor layout is below. In the left part of the window there is a combo box with routing methods (mainly protocols and static routes) and in the right half there are detail parameters for the selected method such as timers, version, redistribution rules and others. Those parameters will be different for each method.

Filtering rules editor will be similar to existing firewall management software. The major difference will be the possibility of so called "macro conditions" - conditions based on general rules, not on exact numbers. It is inevitable in order to achieve portability of packet filter template on various networks. These conditions include "this link subnet", "all subnets of internal links", "all external subnetworks", "all subnets used in this template/node", "all point to point networks" and others. It is also possible to use variables as a reference to prefix/subnet list and to use this variable as a parameter.

Addressing space allocation window layout is displayed below. The big graphical chart on the right side of the window represents the available address space.

**Variables for Node R1**

Variables:

Name	New value	From upper	Inherited	Actual value
\$name	RRG	<none>	<none>	RRG
\$description		Whole net	<none>	
\$l1	<none>	link1	<none>	link1
\$l2	<none>	<none>	extern1	extern1
\$l3	<none>	up-l1	extern2	up-l1
\$l4	up-l3	up-l2	extern3	up-l3

Add variable Remove variable OK Cancel

Figure 5: Variables summary

**Routing template ROU-RIP**

Routing type: RIP

- Static routing
- RIP
- OSPF
- IS-IS

Type specific properties >>>

OK Cancel

RIP parameters

Version out: 1

Version in: 1&2

Activity at border: Passive listener

Update timer: 30

Holddown timer: 90

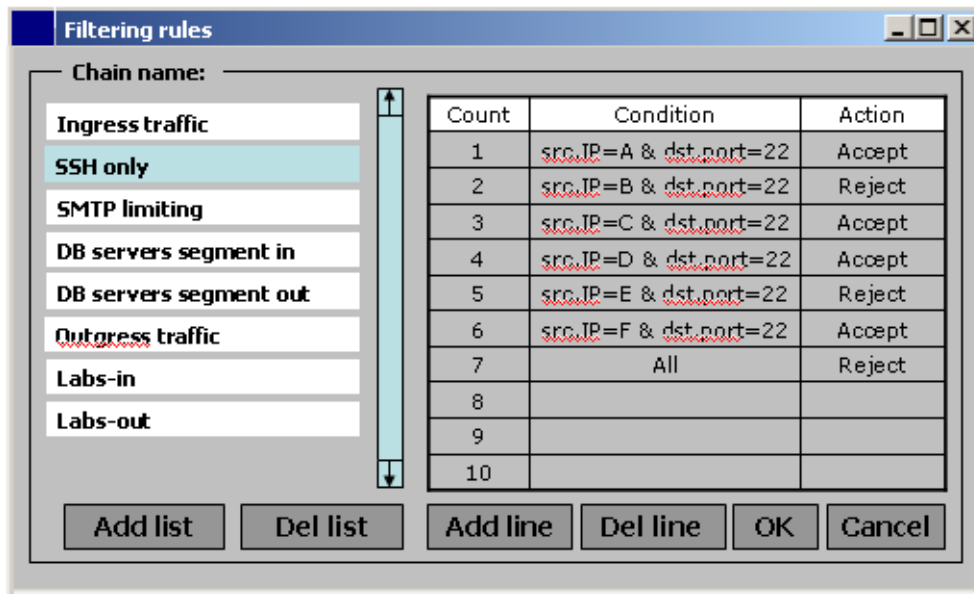
Redistribute from:

ROU-STA ROU-OSPF ROU-ISIS

Redistribute to:

ROU-STA ROU-OSPF

Figure 6: Routing template editor



**Figure 7:** Filtering rules editor

The columns represent the borders between network and host part of the IP address. The user will be able to adjust the columns - to adjust the subnet size and count. Each created subnet will have its unique sequential number and each subnet will be subject to further subnetting into smaller and smaller parts. Each created subnet can be assigned to either link (directly as its subnetwork address) or to node (as a parameter value to be used in nested networks).

The final graphical user interface may differ from these prototypes, but the basic functions, data fields and user activities on templates will remain similar.

## 8 Future L2 data extensions

Current metadata data and function design deals mostly with L3 network attributes; it does not cover lower layer issues. To provide complete solution for network administrator there is need to provide him similar tools for configuring specific link parameters and L2 devices. Modern computer networks include rich featured switches with various options and functionality and there is need to configure them. One link from L3 perspective can be whole bunch of L2 devices and transmission media with complex technologies such as virtual circuits or VLANs.

All this data can be included into current metadata design as an additional element and attribute set. Integrated L2 & L3 dataset will provide the desired goal to configure all active devices (excluding application gateways). Current

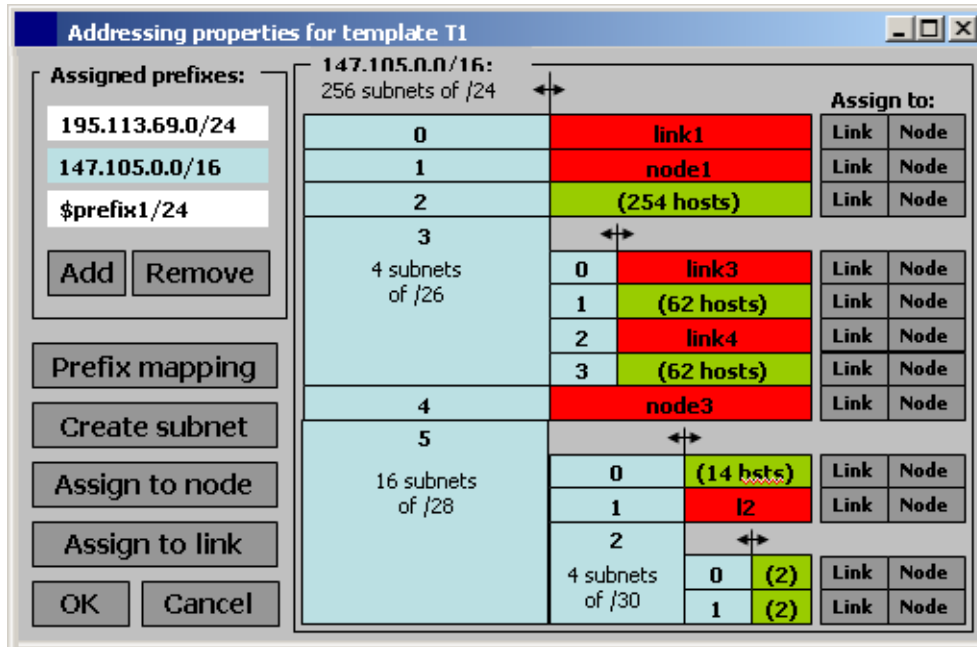


Figure 8: Prefix allocation editor

function and data design does not include those features due to initial phase of metaconfiguration idea and complexity which could make the main idea impossible in reasonable time. But the future possibility of L2 extension is possible and metadata system is ready for that step.

## 9 Full metaconfiguration RelaxNG schema

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">

  <start>
    <element name="metaconfiguration">
      <ref name="metaconfiguration-content"/>
    </element>
  </start>

  <!--Global metaconfiguration data -->
  <define name="metaconfiguration-content">
    <optional>
      <attribute name="name"/>
    </optional>
  </define>

```

```

<optional>
  <element name="networks">
    <ref name="networks-content"/>
  </element>
</optional>
<optional>
  <element name="filter-templates">
    <oneOrMore>
      <element name="filter-template">
        <ref name="filter-attributes-content"/>
      </element>
    </oneOrMore>
  </element>
</optional>
<optional>
  <element name="routing-templates">
    <ref name="routing-templates-content"/>
  </element>
</optional>
<optional>
  <element name="feature-templates">
    <oneOrMore>
      <element name="feature-template">
        <ref name="feature-templates-content"/>
      </element>
    </oneOrMore>
  </element>
</optional>
</define>

```

```

<!--Filtering information section -->
<define name="filter-attributes-content">
  <ref name="identification-content"/>
  <optional>
    <attribute name="direction"/>
  </optional>
  <!--Attributes from the Netopeer system will be here -->
</define>

```

```

<!--Routing information section -->
<define name="routing-common-content">

```

```

    <attribute name="process-id"/>
    <optional>
        <attribute name="use-routing-template"/>
    </optional>
</define>

<define name="routing-content">
    <oneOrMore>
    <choice>
        <element name="static">
            <ref name="routing-common-content"/>
            <!--Attributes from the Netopeer system will be here -->
        </element>
        <element name="rip">
            <ref name="routing-common-content"/>
            <!--Attributes from the Netopeer system will be here -->
        </element>
        <element name="ospf">
            <ref name="routing-common-content"/>
            <!--Attributes from the Netopeer system will be here -->
        </element>
        <element name="is-is">
            <ref name="routing-common-content"/>
            <!--Attributes from the Netopeer system will be here -->
        </element>
        <element name="bgp">
            <ref name="routing-common-content"/>
            <!--Attributes from the Netopeer system will be here -->
        </element>
    </choice>
    </oneOrMore>
</define>

<define name="routing-templates-common-content">
    <attribute name="id"/>
</define>

<define name="routing-templates-content">
    <oneOrMore>
    <choice>
        <element name="static-template">
            <ref name="routing-templates-common-content"/>
            <!--Attributes from the Netopeer system will be here-->
        </element>
    </choice>
</define>

```

```

        </element>
        <element name="rip-template">
          <ref name="routing-templates-common-content"/>
          <!--Attributes from the Netopeer system will be here-->
        </element>
        <element name="ospf-template">
          <ref name="routing-templates-common-content"/>
          <!--Attributes from the Netopeer system will be here-->
        </element>
        <element name="bgp-template">
          <ref name="routing-templates-common-content"/>
          <!--Attributes from the Netopeer system will be here-->
        </element>
        <element name="is-is-template">
          <ref name="routing-templates-common-content"/>
          <!--Attributes from the Netopeer system will be here-->
        </element>
      </choice>
    </oneOrMore>
  </define>

<!--Topology data -->
<define name="networks-content">
  <oneOrMore>
    <element name="network">
      <ref name="network-content"/>
    </element>
  </oneOrMore>
  <attribute name="root"/>
</define>

<define name="network-content">
  <ref name="identification-content"/>
  <optional>
    <element name="set">
      <ref name="set-content"/>
    </element>
  </optional>
  <optional>
    <attribute name="inherit"/>
  </optional>
  <optional>

```

```

    <element name="prefixes">
      <oneOrMore>
        <element name="prefix">
          <ref name="prefix-content"/>
        </element>
      </oneOrMore>
    </element>
  </optional>
  <optional>
    <element name="nodes">
      <oneOrMore>
        <element name="node">
          <ref name="node-content"/>
        </element>
      </oneOrMore>
    </element>
  </optional>
  <optional>
    <element name="links">
      <oneOrMore>
        <element name="link">
          <ref name="link-content"/>
        </element>
      </oneOrMore>
    </element>
  </optional>
  <optional>
    <element name="features-inheritable">
      <ref name="features-content"/>
    </element>
  </optional>
  <optional>
    <element name="features-local">
      <ref name="features-content"/>
    </element>
  </optional>
  <optional>
    <element name="routing">
      <ref name="routing-content"/>
    </element>
  </optional>
</define>

```

```

<define name="node-content">
  <ref name="identification-content"/>
  <ref name="visualization-content"/>
  <optional>
    <attribute name="use-network"/>
  </optional>
  <zeroOrMore>
    <element name="set">
      <ref name="set-content"/>
    </element>
  </zeroOrMore>
  <zeroOrMore>
    <element name="bind">
      <ref name="bind-content"/>
    </element>
  </zeroOrMore>
  <optional>
    <element name="features-inheritable">
      <ref name="features-content"/>
    </element>
  </optional>
  <optional>
    <element name="features-local">
      <ref name="features-content"/>
    </element>
  </optional>
</define>

<define name="link-content">
  <ref name="identification-content"/>
  <optional>
    <attribute name="subnetref"/>
  </optional>
  <optional>
    <element name="features">
      <ref name="link-features-content"/>
    </element>
  </optional>
  <optional>
    <attribute name="estimated-hosts"/>
  </optional>
  <optional>
    <element name="routing">

```

```

        <ref name="routing-content"/>
    </element>
</optional>
</define>

<define name="bind-content">
    <attribute name="linkref"/>
    <optional>
        <attribute name="interface"/>
    </optional>
    <optional>
        <attribute name="hostid"/>
    </optional>
    <optional>
        <element name="features">
            <ref name="interface-features-content"/>
        </element>
    </optional>
    <optional>
        <element name="filters">
            <oneOrMore>
                <element name="filter">
                    <choice>
                        <ref name="filter-attributes-content"/>
                        <attribute name="use-filter-template"/>
                    </choice>
                </element>
            </oneOrMore>
        </element>
    </optional>
</define>

```

```

<!--Addressing data -->
<define name="prefix-content">
    <choice>
        <group>
            <ref name="identification-content"/>
            <attribute name="address"/>
            <attribute name="masklen"/>
        </group>
        <attribute name="idref"/>
    </choice>
</define>

```

```

    <ref name="make-subnets-content"/>
</define>

<define name="make-subnets-content">
  <optional>
    <attribute name="subnetbits"/>
    <oneOrMore>
      <element name="subnet">
        <ref name="subnet-content"/>
      </element>
    </oneOrMore>
  </optional>
</define>

<define name="subnet-content">
  <ref name="identification-content"/>
  <attribute name="subnetid"/>
  <ref name="make-subnets-content"/>
</define>

<!--Parameters/variables common data -->
<define name="set-content">
  <attribute name="name"/>
  <attribute name="value"/>
  <optional>
    <attribute name="index"/>
  </optional>
</define>

<!--Features (simple attributes) section -->
<define name="feature-templates-content">
  <ref name="identification-content"/>
  <ref name="features-content"/>
</define>

<define name="features-content">
  <zeroOrMore>
    <ref name="router-features-content"/>
  </zeroOrMore>
  <zeroOrMore>
    <ref name="link-features-content"/>
  </zeroOrMore>
</define>

```

```

        </zeroOrMore>
        <zeroOrMore>
            <ref name="interface-features-content"/>
        </zeroOrMore>
    </define>

    <define name="router-features-content">
        <ref name="feature-template-use-content"/>
        <!--Attributes from the Netopeer system will be here (e.g., hostname) -->
    </define>

    <define name="link-features-content">
        <ref name="feature-template-use-content"/>
        <!--Attributes from the Netopeer system will be here (e.g., speed)-->
    </define>

    <define name="interface-features-content">
        <ref name="feature-template-use-content"/>
        <!--Attributes from the Netopeer system will be here (e.g., interface des
    </define>

    <define name="feature-template-use-content">
        <zeroOrMore>
            <element name="feature-template-use">
                <attribute name="idref"/>
            </element>
        </zeroOrMore>
    </define>

    <!--Common identification of various elements -->
    <define name="identification-content">
        <attribute name="id"/>
        <optional>
            <attribute name="name"/>
        </optional>
        <optional>
            <attribute name="description"/>
        </optional>
    </define>

    <!--Supporting content (symbols, colors, position...) -->

```

```

<define name="visualization-content">
  <optional>
    <attribute name="vi-posx"/>
    <attribute name="vi-posy"/>
  </optional>
</define>

</grammar>

```

## References

- [Met04a] *Metaconfiguration CVS*  
<http://www.liberouter.org/cgi-bin2/cvsweb.cgi/liberouter/netopeer/metaconf/><sup>1</sup>
- [Met04b] *Metaconfiguration materials repository*  
<http://vseedu.vse.cz/metaconf/><sup>2</sup>
- [Lho03] Lhotka Ladislav: *XML schema for router configuration data: An annotated DTD*, CESNET Technical Report 2/2003  
<http://www.ten.cz/doc/techzpravy/2003/netopeer-dtd/><sup>3</sup>
- [Lib04] *The Liberouter project*  
<http://www.liberouter.org><sup>4</sup>
- [Oas01] The Organization for the Advancement of Structured Information Standards 2001: *Relax NG Specification*  
<http://www.relaxng.org/spec-20011203.html><sup>5</sup>
- [Jel02] Jeliffe R.: *The Schematron Assertion Language 1.5*, 2002  
<http://xml.ascc.net/resource/schematron/Schematron2000.html><sup>6</sup>
- [ITU00] ITU-T Recommendation: *G.805* (not available to public)  
<http://www.itu.int><sup>7</sup>
- [Rex03] Rexford J.: *The cutting EDGE of IP router configuration*  
<http://www.research.att.com/jrex/papers/hotnets03.pdf><sup>8</sup>

---

<sup>1</sup><http://www.liberouter.org/cgi-bin2/cvsweb.cgi/liberouter/netopeer/metaconf/>

<sup>2</sup><http://vseedu.vse.cz/metaconf/>

<sup>3</sup><http://www.ten.cz/doc/techzpravy/2003/netopeer-dtd/>

<sup>4</sup><http://www.liberouter.org>

<sup>5</sup><http://www.relaxng.org/spec-20011203.html>

<sup>6</sup><http://xml.ascc.net/resource/schematron/Schematron2000.html>

<sup>7</sup><http://www.itu.int>

<sup>8</sup><http://www.research.att.com/jrex/papers/hotnets03.pdf>

- [Net03] Bellogini A., Santarelli I.: *NetML, Network Markup Language*  
<http://giga.dia.uniroma3.it/ivan/NetML/><sup>9</sup>
- [W3C99] *XSLT: Extensible Styleheet Transformations*  
<http://www.w3.org/TR/xslt><sup>10</sup>
- [W3C04] *XML: Extensible Markup Language*  
<http://www.w3.org/XML/><sup>11</sup>
- [W3C99a] *XPath: XML Path Language*  
<http://www.w3.org/TR/xpath><sup>12</sup>

---

<sup>9</sup><http://giga.dia.uniroma3.it/ivan/NetML/>

<sup>10</sup><http://www.w3.org/TR/xslt>

<sup>11</sup><http://www.w3.org/XML/>

<sup>12</sup><http://www.w3.org/TR/xpath>