

Network Working Group
Internet Draft
Intended status: Standards Track
Expires: July 28, 2014

Y. Xia
S. Jiang
X. Wang
B. Liu
Huawei Technologies
January 24, 2014

Network Abstract Interface (NAI) Modeling Language
draft-xia-nai-modeling-language-00.txt

Abstract

This document introduces a modeling language used to model network services by network operators and customers to create specific network service instances through the standardized Network Abstract Interface (NAI) and processed automatically by the network.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on July 28, 2014.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Internet-Draft [draft-xia-nai-modeling-language-00](#)

January 2014

Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
2.	Requirements Language and Terminology	3
3.	Requirements for a NAI Modeling Language	3
4.	NAI Modeling Language Overview	3
5.	NAI Modeling Language Syntax and Statements	5
5.1.	Lexical Rules	5
5.1.1.	Comments	5
5.1.2.	Tokenization	5
5.2.	Identifiers	6
5.3.	NML Statements	6
5.4.	Typedef Statements	7
6.	NAI Modeling Language Blocks Definition	7
6.1.	Entity Block	7
6.2.	Capability Block	10
6.3.	User Block	12
6.4.	Service Logic Block	14
6.5.	Open Profiles Block	15
7.	Security Considerations	15
8.	IANA Considerations	15
9.	References	15
9.1.	Normative References	15
9.2.	Informative References	15
10.	Acknowledgments	15
	Authors' Addresses	17

Internet-Draft [draft-xia-nai-modeling-language-00](#)

January 2014

[1.](#) Introduction

In [NAI-ARCH], a model-driven Network Abstract Interface architecture is described. The models are customizable through a model framework and described by network administrators using a NAI modeling language (NML), which is introduced in this document.

The modeling language used to model network services by network operators or customers to create specific network service instances through the standardized Network Abstract Interface (NAI) and processed automatically by the network.

[2.](#) Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [\[RFC2119\]](#) when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [\[RFC2119\]](#) key words.

[3.](#) Requirements for a NAI Modeling Language

For the characteristics of the NAI architecture in [NAI-ARCH], there are some requirements for the modeling language:

- o The language should provide different capabilities and visibility of the network for different users.
- o The language should be simple and easy to use for network administrators.
- o The syntax and keywords should be network related and intuitive.
- o The language should support service logic description as well as service content description.

[4.](#) NAI Modeling Language Overview

NML is used to design specific model for the specific network based on the requirement of opening the network. And the specific model automatically generates specific NAI for the network. The application developer also uses NML to design service logic to access network information and functions.

Different users can have different models to implement their customized service requirement and innovative service application. The common interface styles can be used, such as RESTful style [RESTFUL].

An NAI model is constructed by five blocks: entity block, capability block, user block, service logic block and open profile block. The following figure depicts the relation between the blocks.

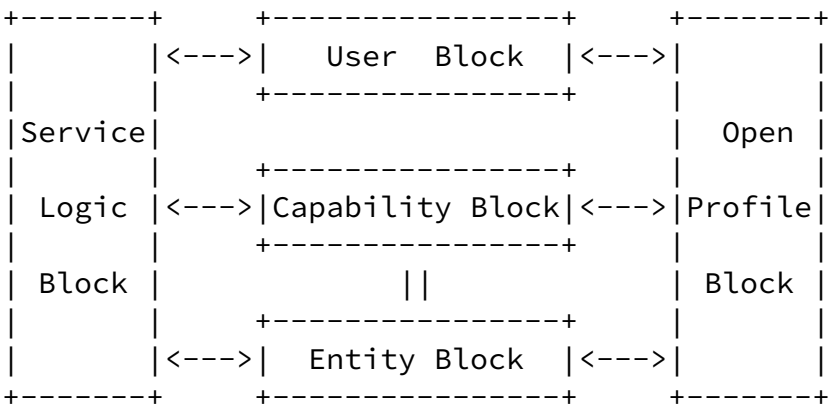


Figure 1 NAI Modeling Language Blocks

Entity Block: describes the network entity that is opened to the user, e.g. network device, device group, port, virtual network device and so on. The property of entity describes the static information of entity, and the statistics of entity describes the statistics and runtime information of entity.

NML provides an inheritance mechanism to define the reusable information about each entity. An inherited entity can be defined in another entity to include its definition information so as to avoid repetition.

Capability Block: describes the capability that is opened to the

user. Capability is a set of network functions and operations that encapsulated together and provided to the users e.g. VPN.

User Block: describes the information about the user group, including the basic user group information, authentication and account information. It distinguishes the user group classified profiles.

Service Logic Block: describes a set of network management functions based on the user's requirement. A service logic block provides the ability to specify automatic, logic control, monitoring, configuration and maintenance for the administrator and NAI application designer. For example, a NAI application can modify the

VPN routing automatically in response to the network load of specified network link, or configure different links for one important enterprise customer between two sites in day or night to meet the SLA requirement. A service logic block designed by one designer can be opened to others to reuse its features, improve NAI ecosystem, support and accelerate growing and innovative network applications.

Open Profile Block: describes the different authorization and visibility of entity and capability opened for different user.

The following section will introduce syntax and keywords for the five blocks.

[5. NAI Modeling Language Syntax and Statements](#)

The NML syntax is similar to the programming languages like C because of the support of service logic.

[5.1. Lexical Rules](#)

[5.1.1. Comments](#)

A comment is text that the compiler ignores but that is useful for programmers. Comments are normally used to annotate code for future reference. The compiler treats them as white space. The comment is written in one of the following ways:

- The /* (slash, asterisk) characters, followed by any sequence of

characters (including new lines), followed by the */ characters. This syntax is the same as ANSI C. Therefore, it is commonly called a "block comment."

- The // (two slashes) characters, followed by any sequence of characters. A new line not immediately preceded by a backslash terminates this form of comment. Therefore, it is commonly called a "single-line comment."

[5.1.2. Tokenization](#)

A token is the smallest element of a NML that is meaningful to the compiler. The NML parser recognizes these kinds of tokens: identifiers, keywords, literals, operators, punctuators, and other separators. A stream of these tokens makes up a translation unit. Tokens are usually separated by "white space." White space can be one or more: Blanks, Horizontal or vertical tabs, New lines, Form feeds.

[5.2. Identifiers](#)

An identifier is a sequence of characters used to denote different kinds of NML items by name. Each identifier starts with an uppercase or lowercase ASCII letter or an underscore character, followed by zero or more ASCII letters, digits, underscore characters, hyphens, and dots.

Each identifier has its namespace. Each identifier is valid in a namespace that depends on the type of the NML item being defined. All identifiers defined in a namespace MUST be unique.

- All entity names share the same global entity identifier namespace.
- All capability names share the same global capability identifier namespace.
- All user names share the same global user identifier namespace.
- All service-logic-names share the same global service logic identifier namespace.
- All property-names share the same identifier namespace defined

in the entity's property statement.

- All statistics-names share the same identifier namespace defined in the entity's statistics statement.
- All para-names share the same identifier namespace defined in the capability parameters statement.
- All output-names share the same identifier namespace defined in the capability outputs statement.

[5.3.](#) NML Statements

A NML model file contains a sequence of statements. Each statement starts with a keyword, followed by zero or one argument, followed either by a semicolon (";") or a block of substatements enclosed within braces ("{ }"):

```
statement = keyword [argument] (";" / "{" *statement "}")
```

Expression statements, Selection Statements, Compound Statements, Iteration Statements are used in NML-script-logic statement body.

[5.4.](#) Typedef Statements

The typedef Statement defines a new derived type from base data type. It is followed by the derived type name immediately.

- The description statement

The "description" statement provides the human-readable descriptive string by the modeler for the NBI application designer to understand the new derived type.

- The basetype statement

The basetype statement defines the base type name from which the new type is derived. The base type can be primitive type or derived type.

- The default statement

The default statement defines the default value for the new derived

type. There may be default value conflict between the base type and derived type. Only the default value with conforming to the type definition is valid. The valid derived type default has higher priority than that of base type. If base type is provided with default value and derived type is not, the default value of base type is used for derived type when it is valid for derived type.

6. NAI Modeling Language Blocks Definition

6.1. Entity Block

Entity block contains the following statements:

- o Class: defines entity's classification. It is enumeration type.
- o Description: takes a string that contains a human-readable textual description of this definition. The text is provided in a language (or languages) chosen by the module developer.
- o Version: defines the version number of current entity.
- o Import: provides a reference mechanism to refer the property and statistics information of imported entity to the importing entity. The data part of the imported data structure is stored in the imported entity.

- o Inherit: provides a include mechanism to combine the property and statistics information of inherited entity into the inheriting entity.
- o Property: defines a data structure of property of the entity. Its sub-statements include property-name, type, description, default. The statements (property name, type, description, default) are combined into one property data structure body. One "property" can have one or more than one property data structure body.
 - Property-Name: defines the name of the property item. The property name must be unique in one "property" statement. This document does not define its naming rule, but it recommends that the name should be simple, human-readable and human-

understandable.

- Type: defines the data type of the property in property data structure body. The built-In types and derived types can both be used.
- Description: provides the human-readable descriptive information by the modeler for the NAI application designer to understand the property item in the property data structure body.
- Default: defines the default value for the property item in the property data structure body. If no value for this property is provided when the entity is instantiated or data record is inserted into the entity, the default value will be used. One property data structure body can have zero or one default statement. If no default statement exists, no default value is provided.

o Statistics: defines a data structure of statistics information of the entity. Its sub-statements include statistics-name, type, description, default. The statements (statistics-name, type, description, default) are combined into one statistics data structure body. One "statistics" can have one or more than one statistics data structure body.

- Statistics-Name: defines the name of the statistics item. The statistics-name must be unique in one "statistics" statement. This document does not define its naming rule, but it recommends that the name should be simple, human-readable and human-understandable.
- Type: defines the data type of the statistics item in the statistics data structure body. The built-In types and derived types can be used in this statement.
- Description: provides the human-readable descriptive information by the modeler for the NBI application designer to understand the statistics item in the statistics data structure

body.

- Default: defines the default value for the statistics item in the statistics data structure body. If no value for this statistics item is provided when the entity is instantiated or data record is inserted into the entity, the default value will be used. One statistics data structure body can have zero or one default statement. If no default statement exists, no default value is provided.

o Description: takes a string that contains a human-readable textual description of this definition. The text is provided in a language (or languages) chosen by the module developer.

Following is an example of entity block definition.

```
enum class /*define some classes of entity*/
{
    ip_device;
    optical_device;
    ...;
}

entity DCI_PE // entity name
{
    class ip_device; // class is enum type
    description "this is a model for DCI PE device";
    version 1.0;
    inherit device;
    property
    {
        property_name loc;//the name of the property
        {
            type uchar;
            description "the location of the entity";
        }
        property_name ipaddr;//the name of the property
        {
            type ulong;
            description "the ip address of the entity";
        }
        ... ;
    }
    statistics
    {
        statistics_name inpacket_num;//the name of the property
        {
            type ulong;
            description "the number of the inpacket";
        }
        ... ;
    }
}
```

```
        description "the number of the inpacket";
    }
    ... ;
}
```

}

6.2. Capability Block

Capability block contains the following statements:

- o Class: defines capability's classification. It is enumeration type.
- o Description: provides the human-readable descriptive string by the modeler for the NAI application designer to understand the capability.
- o Version: defines the version number of current capability.
- o Parameters: defines the input parameters for network to execute the capability. It provides a mean for NAI application designer to implement his special service requirement through customized network parameters in his service case. Its sub-statements include para-name, type, description, default. The statements (para-name, type, description, default) are combined into one parameters structure body. One "parameters" statement can have zero, one or more parameters structure body.
 - Para-Name: defines the name of the parameter. The parameters must be unique in one parameters structure body. This document does not define its naming rule, but it recommends that the name should be simple, human-readable and human-understandable.
 - Type: defines the data type of the parameter in the parameters structure body. The built-In types and derived types can be used in this statement.
 - Description: provides the human-readable descriptive information by the capability provider for the NAI application designer to understand the parameter in the parameters structure body.
 - Default: defines the default value for the parameter in the parameters structure body. If no value for this parameter is provided when the capability is executed, the default value will be used. One parameters structure body can have zero or one default statement. If no default statement exists, no default value is provided.

- o Outputs: defines the output results from network after executing the capability. It provides a mean for NAI application designer to

get the network entity property, statistics, or other response in his service application case. Its sub-statements include output-name, type, and description. The statements (output-name, type, description) are combined into one outputs structure body. One "outputs" statement can have zero, one or more outputs structure body.

- Output-Name: defines the name of the output. The output-name must be unique in one outputs structure body. This document does not define its naming rule, but it recommends that the name should be simple, human-readable and human-understandable.
- Type: defines the data type of the output parameter in the outputs structure body. The built-In types and derived types can be used in this statement.
- Description: provides the human-readable descriptive information by the capability provider for the NAI application designer to understand the output parameter in the outputs structure body.

o Execution-Link: takes one argument of string as the name of the execution link and two other statement to define its type and description.

- Type: defines the type of the execution link in the capability. It is enumeration type.
- Description: provides the human-readable descriptive information by the capability provider for the NAI application designer to understand the execution link.

Following is an example of capability block definition.

```
enum class /*define some classes of capability*/
{
    network cap;
    Service cap;
    ...;
}

capability vpn // capability name
{
    class network cap; // class is enum type
    description "this is a model for creating vpn capability";
    version 1.0;
    parameter
    {
        para name pe; //the name of the property
```

```
{
    type ulong;
    description "the IP address of the PE";
}
para name connection;//the name of the property
{
    para name src;
    {
        type ulong;
        description "the source ip address of the connection";
    }
    para name dst;
    {
        type ulong;
        description "the destination ip address of the
                    connection";
    }
    ...
}
... ;
}
output
{
    output name http link;
    {
        type string;
        description "the output is a http link";
    }
    ...
}
execution link
{
    exec name exec link;
    {
        type string;
        description "the name of execution body";
    }
    parameter pe;
    parameter connection;
    ...
}
```

[6.3.](#) User Block

User block contains the following statements:

- o Usergrp: defines user group classification to which administrator wants to open entities and capabilities. The "usrgrp" statement takes one argument of string as the name of the entity and one usrgrp structure body to describe the details of the usrgrp.
- o Description: takes a string that contains a human-readable textual description of this definition. The text is provided in a human language (or languages) chosen by the module developer.
- o Information: defines a data structure of information of the usrgrp. Its sub-statements include info_name, type, description, default. The statements (info_name, type, description, default) are combined into one information data structure body. One "information" statement can have one or more than one information data structure body.
 - info_name: defines the name of the information item. The info_name must be unique in one "information" statement. This document does not define its naming rule, but it recommends that the name should be simple, human-readable and human-understandable.
 - type: defines the data type of the information in information data structure body. The built-In types and derived types can be used in this statement.
 - default: defines the default value for the information item in the information data structure body. If no value for this information is provided when the usrgrp is instantiated or data record is inserted into the usrgrp, the default value will be used. One information data structure body contains zero or one default statement. If no default statement exists, no default value is provided.
- o Authentication: defines the authentication mode of the usrgrp, which is followed by an "enum" variable. So administrator needs to define all the allowed authentication mode with an "enum" variable ahead.
- o Charge: defines the charge mode of the usrgrp, which is followed by an "enum" variable. So administrator needs to define all the allowed charge mode with an "enum" variable ahead.

Following is an example of user block definition.

```
enum authes
{
```

```
        non;
        ipsec;
        ...;
    }

enum charges
{
    buflow;
    plan1;
    ...;
}

usergrp OTT user // usergrp
{
    description "this is a model for OTT user";
    information
    {
        info name name;//the name of the property
        {
            type uchar;
            description "the name of the entity";
        }
        ... ;
    }
    authentication authes;
    charge charges;
}
```

[6.4.](#) Service Logic Block

Service logic block contains the following statements:

- o Selection: mainly includes "if-else" and "switch".
- o Expression: causes expressions to be evaluated. No transfer of control or iteration takes place as a result of an expression

statement.

- o Compound statement consists of zero or more statements. A compound statement can be used anywhere a statement is expected.

- o Iteration: causes statements (or compound statements) to be executed zero or more times, subject to some loop-termination criteria. When these statements are compound statements, they are executed in order. NML provides three iteration statements -"while", "do" and "for".

Liu, et al.

Expires July 28, 2014

[Page 14]

Internet-Draft

[draft-xia-nai-modeling-language-00](#)

January 2014

- o Break: causes the innermost enclosing loop or a switch statement to exit immediately.

- o Continue: is similar to the "break" statement. Instead of exiting a loop, however, "continue" restarts a loop in a new iteration.

- o Operators: NML support binary arithmetic operators, assignment operators, comparison operators, bitwise operators and string operators.

(An example needs to be filled in the future.)

[6.5](#). Open Profiles Block

The open model is a bit map which the manager pictures based on open requirements, which is fixed and needn't to be modeled.

[7](#). Security Considerations

Since customers can order services directly through the NAI interface, authentication and authorization is strongly needed.

[8](#). IANA Considerations

None.

[9](#). References

[9.1](#). Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.

[9.2](#). Informative References

[NAI-ARCH] Xia Y., Jiang S., and X. Wang, "Customizable Network Abstract Interface (NAI) Architecture based on Model Description", Work in Progress, January 2014

[RESTFUL] Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures",
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000

[10](#). Acknowledgments

Valuable comment was received from Brian E Carpenter.

Liu, et al.

Expires July 28, 2014

[Page 15]

Internet-Draft

[draft-xia-nai-modeling-language-00](#)

January 2014

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Yinben Xia (editor)
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: xiayinben@huawei.com

Sheng Jiang (editor)
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: jiangsheng@huawei.com

Xuwei Wang
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095

P.R. China

Email: wangxuewei@huawei.com

Bing Liu

Huawei Technologies Co., Ltd

Q14, Huawei Campus, No.156 Beiqing Road

Hai-Dian District, Beijing, 100095

P.R. China

Email: leo.liubing@huawei.com