

CESNET Technical Report 3/2010

Supplementary Service Implementation in IP Telephony by DDDS Application

JAN RUDINSKÝ

Received 26.04.2010

Abstract

Technical report introduces a method for implementation of Supplementary Services in IP telephony. The services were implemented by different types of Intelligent Network in Public Switched Telephone Network, however such a unified solution for IP telephony is missing. The report proposes, specifies and describes implementation of DDDS application for supplementary services provision in IP telephony.

Keywords: DDDS, enum, IP telephony, supplementary service, intelligent network

1 Introduction

A concept of Next Generation Network (NGN) was born at ITU as a convergence of traditional telephony with packet networks. NGN represents unified, high-speed packet network that provides number of services with guaranteed quality. IP telephony and IMS architecture for mobile networks can be considered as implementations of NGN concept.

The primary goal of any telecommunication network is to provide basic services such as telephony. The network in addition provides supplementary services (e.g. call forwarding, screening, alternative charging, etc.) that increase user comfort and cause an intense market competition.

Supplementary services in traditional telephony were provided by Intelligent Network (IN) concept developed by ITU. The services were introduced in groups starting with Capability Set 1 (ITU-T Q.121x) to CS-4 (Q.124x) [1]. An important fact is that the recommendations, instead of providing exact service specification, give only elementary principles definition that can be used to build services.

IP telephony can be considered missing a global mechanism for supplementary service provision. One way to implement the services is by convergence with IN and traditional telephony network. Another more recent activity by IETF is represented by direct services implementation by protocol SIP [2]. Both solutions are either complex and require changes to be made at client/server side or implement only a relatively small subset of services. An alternative method can be the utilization of existing IP mechanisms.

2 ENUM

An observation has been made that number of IN supplementary services essentially convert identifiers of calling parties. Therefore a mechanism that translates

telephony and VoIP identifiers can be used for the service implementation in IP telephony. ENUM [3] has been proposed as a translation mechanism between E.164 telephony numbers [4] and identifiers used in the Internet (URIs). It thus enables the calls originating in the telephony network to be connected to the endpoints located in the Internet. The capability of string translation and route selection based on priority designates ENUM to be a mechanism representing IN services.

However analysis of ENUM exposes two constraints. First, the input is represented by a single telephone number in the form according to ITU-T E.164. But there are services that require processing of non-digit input (such as password). The processing of single input is the second limitation. But many services require more than one input parameter. Such an example is Abbreviated dialing, where both calling party number and abbreviated number are needed for conversion into called party identifier.

3 DDDS application design

Because ENUM mechanism is limited in the capability to implement maximum number of IN services, new DDDS application is proposed and specified in this text.

ENUM mechanism can be described as an application of Dynamic Delegation Discovery System (DDDS). DDDS application represents an abstract algorithm operating on a database with rewrite rules used by the application for string conversion. In order to design an alternative DDDS application to ENUM several parameters need to be defined:

- Algorithm
- Database
- Application specific parameters

3.1 The algorithm

General DDDS algorithm was already specified in RFC 3402 [5]. The service is provided as a string processing defined by the algorithm. The input is initially converted into a database search key used later to query the database. The key is then matched to database records in order to retrieve rewrite rules for input string conversion. In case a rule is not final, it is applied on the initial input and the search is repeated. Once the terminal rule is reached it is applied on the input string to produce output string for further call processing.

3.2 The Database

The database contains rewrite rules for string conversion. DDDS specification does not imply any specific database, however a DNS-based hierarchical system has been proposed in RFC 3403 [6]. Properties of well known DNS and its scalability make it desirable storage for application rewrite rules. The rules are stored in format of NAPTR resource records.

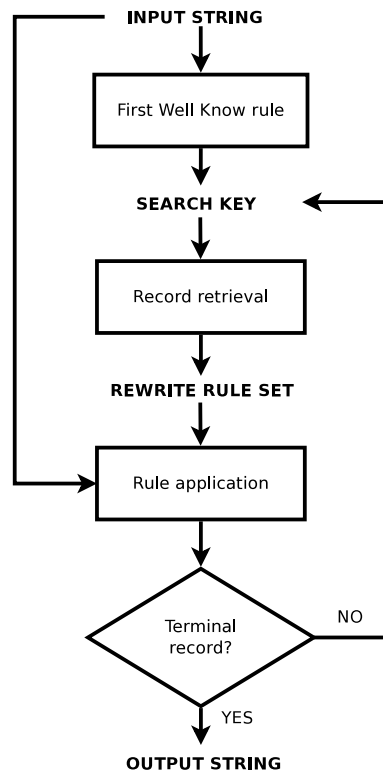


Figure 1. General DDDS algorithm.

3.3 The Application-Specific Parameters

DDDS application implements the algorithm and makes use of selected database. The application has to specify four parameters:

- Unique input
- First well known rule
- Select database
- Output

3.3.1 Unique input

As the application is aimed at identifier translation following input types encoded in UTF-8 are proposed:

- Non-digit string
- Single E.164 number
- Couple of E.164 numbers

An input string must be unique to identify single search path and it has to include complete information to reach desired output. The unique input is used to create a search key and to apply rewrite rule upon to obtain application output.

Input can be divided into fixed and dynamic part. Compulsory fixed part is known prior to service execution, while dynamic part changes with every call. Thus the fixed part can be stored in the database as well as used during the search process. Both fractions need to be separated by a delimiter. It should not be any character that appears neither in fixed nor in dynamic input and it should not play primary

role in regular expressions (as it is processed by regexp.). Therefore “&” is selected as delimiter.

The unique input has been defined by ABNF:

```

uniq-input= fixed-part delim-char dyn-part
delim-char= "&"
fixed-part= uri / e164number
dyn-part= uri / e164number
uri= <URI as defined in RFC 2396>
e164number= <E.164 phone number>

```

3.3.2 First well known rule

Initial FWKR is applied on the input and provides a database search key. The specification of the rule conforms to rewrite rules and is represented in the form of regular expression below. FWKR represents an association between the unique input fixed part and the operator domain for service provider identification. When applied the rule provides domain name search key for DNS query.

```
!^[ (^&]*)&!\1.OPERATOR_DOMAIN!
```

3.3.3 Database

As previously mentioned, proposed DDDS application makes use of DNS to store and query the NAPTR records. The records include a rewrite rule set for string translation based on regular expressions. These rules applied to the unique input provide either consecutive search key or final output. The use of the rules represents the service provision in one of three forms:

- Translation of involved party identifier
- Verification of identifier presence
- Output selection for routing purpose

Each rule composes of priority, flags, rewrite rule and service parts. Priority identifies the order in which rules are processed and it can be used for output selection to route the call. Flag indicates, whether a rule is terminal or not and what the following process should be. Rewrite regular expression specifies the translation of string identifier. The regular expression part of NAPTR record for the proposed DDDS application has the ABNF form below. It is composed of two parts. First is regular expression (“ere”) that identifies part of the input string intended for subsequent processing. The second part is substitution string (“repl”) that contains selected input and an additional part. Application output string is a result of applying the substitution on unique input string.

```

subst-expr = delim-char ere delim-char repl delim-char *flags
delim-char = "/" / "!" / <char except 'POS-DIGIT' and 'flags'>
ere = <POSIX Extended Regular Expression>
repl = *(string / backref)

```

```

string = *(anychar / escapeddelim)
anychar = <any character other than delim-char>
escapeddelim = "\" delim-char
backref = "\" POS-DIGIT
flags = "i"
POS-DIGIT = "1"/"2"/"3"/"4"/"5"/"6"/"7"/"8"/"9"

```

Finally service field of NAPTR identifies meaning of the record, which has to be unique for every record within a domain (represented by call-party identifier). DDDS application defines two services kinds, where “type” represents a service identifier as specified in ITU recommendation:

- E2E+type = E.164 numbers translation
- E2U+type = E.164 number-to-URI translation

3.3.4 Output

Output of the application represents the unique input processed by the rewrite rules. Depending on the service type result can be either E.164 number or URI. However some services require an information whether an input string is present in the database instead. Therefore an error channel is added. DNS provides three response types (below), where the first type is used for return of E.164 number or URI and name error type is used for service check of string presence.

- Record corresponding to queried name and type
- Name error
- Data not found error

4 DDDS application specification

Specification and Description Language, the graphical representation (SDL/GR) is used for formal specification DDDS application and its behavior. The application is realized by a tree structure composed of “system” as root, “block” as middle layer and “process” as lowest functional element.

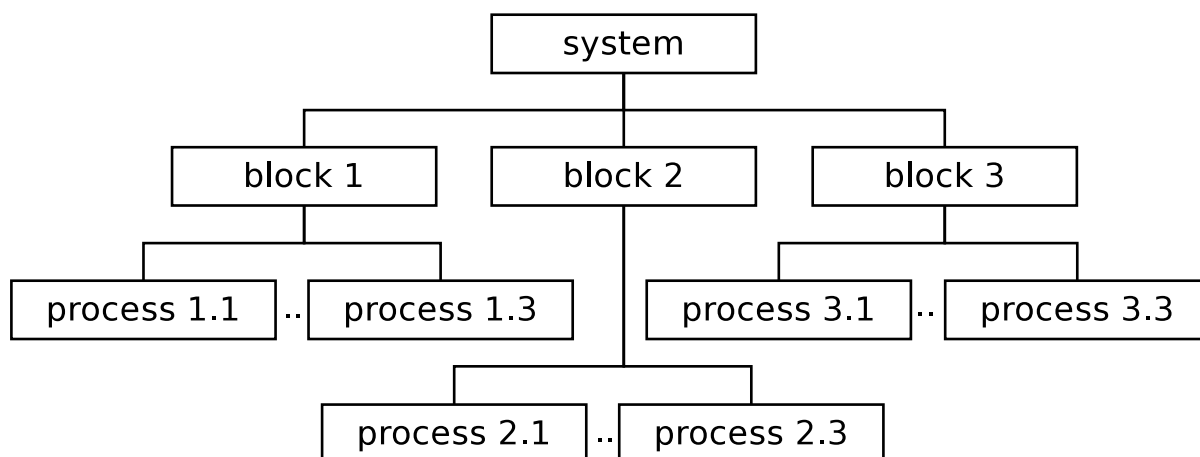


Figure 2. SDL/GR specification of the DDDS application.

The system represents application interaction with environment through input and output channels. The blocks represent functional elements with simplest interface to each other. The processes define system behavior and are composed of Extended Finite State Machines (EFSM).

The system element makes border between the application and the environment. The application is a stand alone process with start up parameters as input channel and a return value as the output. The system element includes three blocks.

Input processing block (“ZPRACOVÁNÍ VSTUPU”) checks the input parameters such as service type or connection identifiers. Record look up block (“VYHLEDÁNÍ ZÁZNAMU”) creates the unique input, applies the first well known rule onto input to produce a search key and finally queries the rewrite rules from the database. Rule application block (“APLIKACE PRAVIDEL”) then applies the acquired rules on unique input to produce output. The output is checked on format and returned by system. Each block contains three processes, whose interaction defines the system behavior. For details on process specification, please refer to [7].

5 Implementation

The application is implemented for functionality verification in IP telephony environment. Out of numerous IP telephony software, Asterisk PBX is selected as a test platform for following reasons. Asterisk supports SIP, the most common signalling protocol today. PBX is often used by small operators and the possibility to implement IN services can increase their competitiveness on the market. Finally the software has many programmable interfaces that can be used for application implementation.

BIND name server is chosen as DNS representative in connection with MySQL database as resource record storage. The architecture is depicted on the picture.

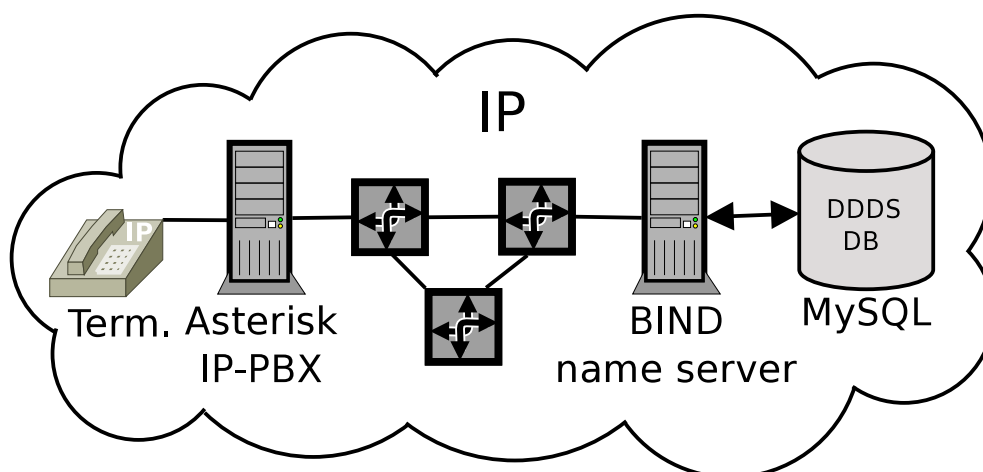


Figure 3.

DDDS application is implemented by shell script (see Appendix A) and it is portable to other platforms. The script is called by SHELL() function in Asterisk configuration file extensions.conf. The function receives input that contains script path and DDDS application input parameters (such as call party identifiers). It

returns the output (identifier), where to connect the call or information about accepting or rejecting the call. Here is an example of Free Phone service:

```
exten => _800XXXXXX,1,NoOp(FPH service called, A=${CALLERID(num)}, B=${EXTEN})
exten => _800XXXXXX,n,Set(Result=${SHELL(/opt/asterisk/bin/ddds_application \
    E2E+FPH ${CALLERID(num)} ${EXTEN}))})
exten => _800XXXXXX,n,GotoIf(["${Result}" != ""]?default,${Result},1)
exten => _800XXXXXX,n,Macro(incept,1)
```

The format of NAPTR resource record stored by BIND name server and including rewrite rules can have following form:

```
$ORIGIN blue.comtel.cz. 800111112 \
    NAPTR 10 100 "U" "E2E+FPH" "!(^800)(.*)"(&.*$)!222\2!"
```

DDDS application implementation is verified on Free Phone service. The output from Asterisk follows:

```
-- Executing [800111112@default:1] NoOp("SIP/602336334-084cac8b",
    "FPH service called, A=602336334, B=800111112") in new stack
-- Executing [800111112@default:2] Set("SIP/602336334-084cac8b",
    "Result=222111112") in new stack
-- Executing [800111112@default:3] GotoIf("SIP/602336334-084cac8b",
    "1?default,222111112,1") in new stack
-- Goto (default,222111112,1)
```

6 Conclusion

It is shown that a number of supplementary services realized in the traditional telephony by Intelligent Network can be transferred to IP telephony. Basically these services process identifiers of calling parties for routing purposes and user verification. This can be realized by ENUM that enables telephony (E.164 numbers) and VoIP (URIs) identifiers manipulation. However limitation of ENUM mechanism leads to proposal of DDDS application capable of implementing more supplementary services, while keeping the advantages of the mechanism.

The proposed DDDS application is defined in the form of parameters: unique input, first well known rule, rewrite rules and output. The application process is formally specified using SDL Graphical Representation that defines structural components and behavior. The application functionality is verified by implementation in IP telephony system. DDDS application is build as shell script called by Asterisk PBX connected to BIND name server. A Freephone service validates the functionality of proposed application with positive result.

7 Acknowledgment

This work has been carried out in cooperation of Cesnet z.s.p.o., Multimedia transmissions and collaborative environment group and Czech Technical University in Prague, Faculty of Electrical Engineering.

References

- [1] International Telecommunications Union. Recommendations Q.121x, Q.122x, Q.123x and Q.124x.
- [2] JOHNSTON, A. (Ed.) et al. *Session Initiation Protocol Service Examples*. RFC 5359¹, IETF, October 2008.
- [3] FALTSTROM, P.; MEALLING, M. *The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)*. RFC 3761², IETF, April 2004.
- [4] International Telecommunications Union. Recommendation E.164.
- [5] MEALLING, M. *Dynamic Delegation Discovery System (DDDS). Part Two: The Algorithm*. RFC 3402³, IETF, October 2002.
- [6] MEALLING, M. *Dynamic Delegation Discovery System (DDDS). Part Three: The Domain Name System (DNS) Database*. RFC 3403⁴, IETF, October 2002.
- [7] RUDINSKÝ, J. *Alternativní přístupy k řešení implementace služeb Inteligentní sítě v prostředí IP telefonie*. (In Czech: Alternative methods of Intelligent Network service implementation in IP telephony.) Dissertation, Praha: Czech Technical University, Faculty of Electrical Engineering, October 2009.

¹ <http://tools.ietf.org/rfc/rfc5359.txt>

² <http://tools.ietf.org/rfc/rfc3761.txt>

³ <http://tools.ietf.org/rfc/rfc3402.txt>

⁴ <http://tools.ietf.org/rfc/rfc3403.txt>

Appendix A. DDDS Application Shell Script

```
#!/bin/bash
# A simple telephone service DNS query tool
# Searches a NAPTR record of a specified type (like E2E) and performs
# transformation of entered number(s) according to the gathered rule.
# Input: $1: Service type(E2E+FPH, U2U+NP, etc.)
# $2: Calling number(used for lookup)
# $3: Called number(used for lookup)
# $4: Card number(used for lookup)
# $5: PIN(used for lookup)
# Some global variables may come handy
# export DNSSERVER=ns.blue.comtel.cz
export DNSSERVER=212.24.135.15
# URI format check
# Input: $1: URI
function checkuri() {
  uri=`echo $1 | cut -d":" -f1`
  id=`echo $1 | cut -d":" -f2`
  if [ "$uri" == "tel" -o "$uri" = "sip" -o "$uri" = "h323" ] ; then
    echo $id
  fi
}
# Check if string contains specified number of digits
# It is a common code to following check functions
# Input: $1: Checked string
# $2: Minimum digits
# $3: Maximum digits
function checkdigits() {
  num=$1
  len=${#num}
  if [ "$len" -ge "$2" -a "$len" -le "$3" ] ; then
    echo $1 | grep "^[0-9]*$"
  fi
}
# Check of E.164 phone number format
# number should be 15 digits maximum
# Input: $1: the number
function checkE164() {
  echo `checkdigits $1 1 15`
}
# Checks credit card format
# card number is 12-19 digits long
# Input: $1: card number
function checkcard() {
  echo `checkdigits $1 12 19`
}
# Checks PIN format
# PIN is set to be 8 digits
# Input: $1: PIN
function checkpin() {
  echo `checkdigits $1 8 8`
}
}
```

```

# This function verifies the global input parameters. It fails if there is
# not enough parameters, or if the parameters are not valid
# Input: $1: Global $1
# $2: Global $2
# $3: Global $3
# $4: Global $4
# $5: Global $5
# Output: Nothing
# Special: This function terminates the whole command if an error is found.
function checkinput() {
  servtype=$1
  # extract DDDS application service type (E2E, U2U)
  dddsserv=`echo $servtype | cut -d"+" -f1`
  # extract IN service type (FPH, NP,..)
  inserv=`echo $servtype | cut -d"+" -f2`
  case $inserv in
    "CD" | "CF" | "CRD" | "FMD" | "SCFB" | "CFB" | "CFN" | "UPT" | "NP" | "CS")
    # check called party identity
    if [ "$dddsserv" == "E2E" ]; then
      konpar3=`checkE164 $3`
    elif [ "$dddsserv" == "U2U" ]; then
      konpar3=`checkuri $3`
    fi
    if [ -z "$konpar3" ]; then
      echo >&2 Parameter check failed.
      exit 5
    fi
    ;;
    "DCR" | "FPH" | "SCF" | "TCS" | "UAN" | "ABD" | "OCS" | "UDR" | "VPN")
    # check calling and called party identity
    if [ "$dddsserv" == "E2E" ]; then
      konpar2=`checkE164 $2`
      konpar3=`checkE164 $3`
    elif [ "$dddsserv" == "U2U" ]; then
      konpar2=`checkuri $2`
      konpar3=`checkuri $3`
    fi
    if [ -z "$konpar2" -o -z "$konpar3" ]; then
      echo >&2 Parameter check failed.
      exit 5
    fi
    ;;
    "ACC" | "AAB" | "CCC")
    # check called party identity, card number and pin
    if [ "$dddsserv" == "E2E" ]; then
      konpar3=`checkE164 $3`
      konpar4=`checkcard $4`
      konpar5=`checkpin $5`
    elif [ "$dddsserv" == "U2U" ]; then
      konpar3=`checkuri $3`
      konpar4=`checkcard $4`
      konpar5=`checkpin $5`
    fi
  fi
}

```

```

if [ -z "$konpar3" -o -z "$konpar4" -o -z "$konpar5" ]; then
echo >&2 Parameter check failed.
exit 5
fi
;;
"CON")
# check called party identity and pin
if [ "$dddsserv" == "E2E" ]; then
konpar3=`checkE164 $3`
konpar5=`checkpin $5`
elif [ "$dddsserv" == "U2U" ]; then
konpar3=`checkuri $3`
konpar5=`checkpin $5`
fi
if [ -z "$konpar3" -o -z "$konpar5" ]; then
echo >&2 Parameter check failed.
exit 5
fi
;;
"SEC")
# check calling party identity and pin
if [ "$dddsserv" == "E2E" ]; then
konpar2=`checkE164 $2`
konpar5=`checkpin $5`
elif [ "$dddsserv" == "U2U" ]; then
konpar2=`checkuri $2`
konpar5=`checkpin $5`
fi
if [ -z "$konpar2" -o -z "$konpar5" ]; then
echo >&2 Parameter check failed.
exit 5
fi
;;
*)
echo >&2 Invalid service type input.
exit 5
;;
esac
}
# Prepare Application Unique String
# It composes of fixed and dynamic parts separated by "&"
# Parts are assigned from caller identities, card number and PIN
# Input: $1: Global $1
# $2: Global $2
# $3: Global $3
# $4: Global $4
# $5: Global $5
function makeaus(){
servtype=$1
inserv=`echo $1 | cut -d"+" -f2`
case $inserv in
"CD" | "CF" | "CRD" | "FMD" | "SCFB" | "CFB" | "CFN" | "UPT" | "NP" | "CS")
# assign Fixed-part=B

```

```

echo -n "$3&"
;;
"DCR" | "FPH" | "SCF" | "TCS" | "UAN")
# assign Fixed-part=B and Dynamic-part=A
echo -n "$3&$2"
;;
"ABD" | "OCS" | "UDR" | "VPN")
# assign Fixed-part=A and Dynamic-part=B
echo -n "$2&$3"
;;
"ACC" | "AAB" | "CCC")
# assign Fixed-part=CN+PIN and Dynamic-part=B
echo -n "$4$5&$3"
;;
"CON")
# assign Fixed-part=B+PIN
echo -n "$3$5&"
;;
"SEC")
# assign Fixed-part=A+PIN and Dynamic-part=B
echo -n "$2$5&$3"
;;
esac
}
# This function changes the number onto the key used for DNS query
# Input: $1: Application Unique String
# $2: The domain
# Output: stdout
function mkkey() {
# Pozn: je spravne zapsane vyjmuti fixni slozku z AUS ?
# - jsou to vsechny znaky po prvni "&"
# Odp: Nefungovalo to, pouzijeme obvykly zpusob, který je navíc rychlejší
# a homogenni s ostatním kódem
# Dale je zde připsáno ev. odstranění tel: či obdobných URI prefixu.
fixpart=`echo $1 | cut -d"&" -f1 | cut -d":" -f 2`
echo -n "$fixpart.$2"
}
# This function makes the query itself and returns data from the DNS
# It takes just the first entry, if there are more than one.
# Input: $1: The string to search for
# $2: The service type (E2E+FPH)
# Output: stdout
function dnsquery() {
host -a $1 $DNSSERVER | \
grep NAPTR | \
grep $2 | \
gawk '{print $9}' | \
head -n 1
}
# This function applies the rule to the original number
# Input: $1: Number (string) to process
# $2: The rule
# Output: stdout

```

```

function applyrule() {
  srch=`echo $2 | cut -d! -f2`
  repl=`echo $2 | cut -d! -f3`
  echo "$1" | gawk --re-interval "{print(gensub(\\\"$srch\\\",\\\"$repl\\\",1))}"
}
# This function verifies the result from all the previous steps.
# If there is a result starting with tel: URI, it is considered valid.
# The tel: specifier is removed from the beginning, to return just the pure
# number.
# Input: $1: The service type
# $2: The result preceeding process
# Output: stdout
function checkoutput() {
  servtype=$1
  dddsserv=`echo $servtype | cut -d"+" -f1`
  uri=`echo $2 | cut -d":" -f1`
  num=`echo $2 | cut -d":" -f2`
  tel=`echo $2 | grep "^[0-9]*$"`
  if [ "$dddsserv" = "U2U" -a "$uri" = "tel" ] ; then
    echo $num
  fi
  if [ "$dddsserv" = "E2E" -o -z "$tel" ] ; then
    echo $tel
  fi
}
# This is the "main" part of the script.
# Verify the input first.
checkinput $1 $2 $3 $4 $5
# Create Application Unique String
myaus=`makeaus $1 $2 $3 $4 $5`
# Get the key for the DNS query
mykey=`mkkey $myaus blue.comtel.cz`
echo $mykey
# Search the DNS to find the rule
myrul=`dnsquery $mykey $1`
# Terminate if we didn't get a result.
if [ -z "$myrul" ]; then
  echo >&2 DNS entry not found.
  exit 5
fi
# Apply the rule on the input.
myres=`applyrule $myaus $myrul`
# Make the final check.
final=`checkoutput $1 $myres`
# If our result is empty (check failed), terminate with non-zero as for other
# errors.
if [ -z "$final" ]; then
  echo >&2 Number processing failed.
  exit 5
fi
# And finally, print the final result to the stdout!
echo -n $final
# End of dnsquery script

```