N-SQUARED SOFTWARE

Introduction to N2TTG Subscriber Self-Mgmt.

N-SQUARED SOFTWARE

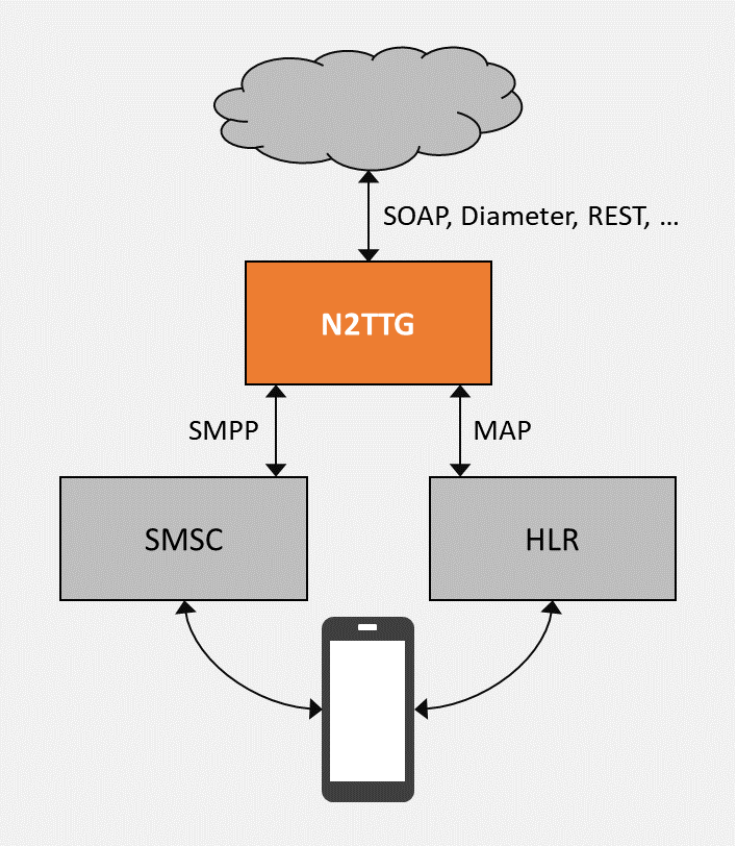
N-SQUARED SOFTWARE

What Is N2TTG?

- N2TTG is a subscriber interaction platform.
- The key features of N2TTG are:
  - Inbound request-response text messages via SMPP with a core network SMSC.
  - Outbound request-response text messages via SMPP with ASP.
  - Inbound request-response text messages via USSD/MAP with a core network HLR.
  - Inbound menu navigation and selection via USSD/MAP with a core network HLR.
  - Outbound USSD flash notification via USSD/MAP to a core network HLR.
  - Extensible communication and/or triggering to or from other network elements via Diameter, SOAP, REST, and more.

N-SQUARED SOFTWARE

A Subscriber Communication Gateway



- N2TTG controls subscriber interactions in both directions:
  - Both interception of subscriber-initiated messages from the core network, and
  - Handling of system-initiated communications to subscribers.
  - Brokering of information from multiple wider network elements as required for message enrichment.
  - Unified generation and translation of text for end-users.

N-SQUARED SOFTWARE

When To Use N2TTG?

- N2TTG is used for:
  - Translating system-triggered events into subscriber notifications.
  - Handling all text-based subscriber self-management from a mobile handset.
- Typical services using N2TTG are:
  - System-initiated outbound messages for low balance, call cost, account expiry, etc.
    - Either via USSD (flash) or SMS (flash or standard).
  - USSD menus for bundle/add-on purchases, product type swap, friend and family number selection, call history, etc.
  - Request-response messages for voucher top-up, balance enquiry, favourite destination selection, etc.
    - Selected via USSD codes (e.g. *123#) or SMS keywords (e.g. bal)

N-SQUARED SOFTWARE

What N2TTG Is Not

- N2TTG holds no data natively:
  - All data for processing requests is either from the input received, or
  - Retrieved from an external data source via any required method, e.g. DB lookup, REST query, Diameter message, etc.
- N2TTG deals only with text-based messaging:
  - No voice interaction (use the N2SRP instead).
  - No call control (use N2DSG, N2NP, or N2ACD instead).

N-SQUARED SOFTWARE

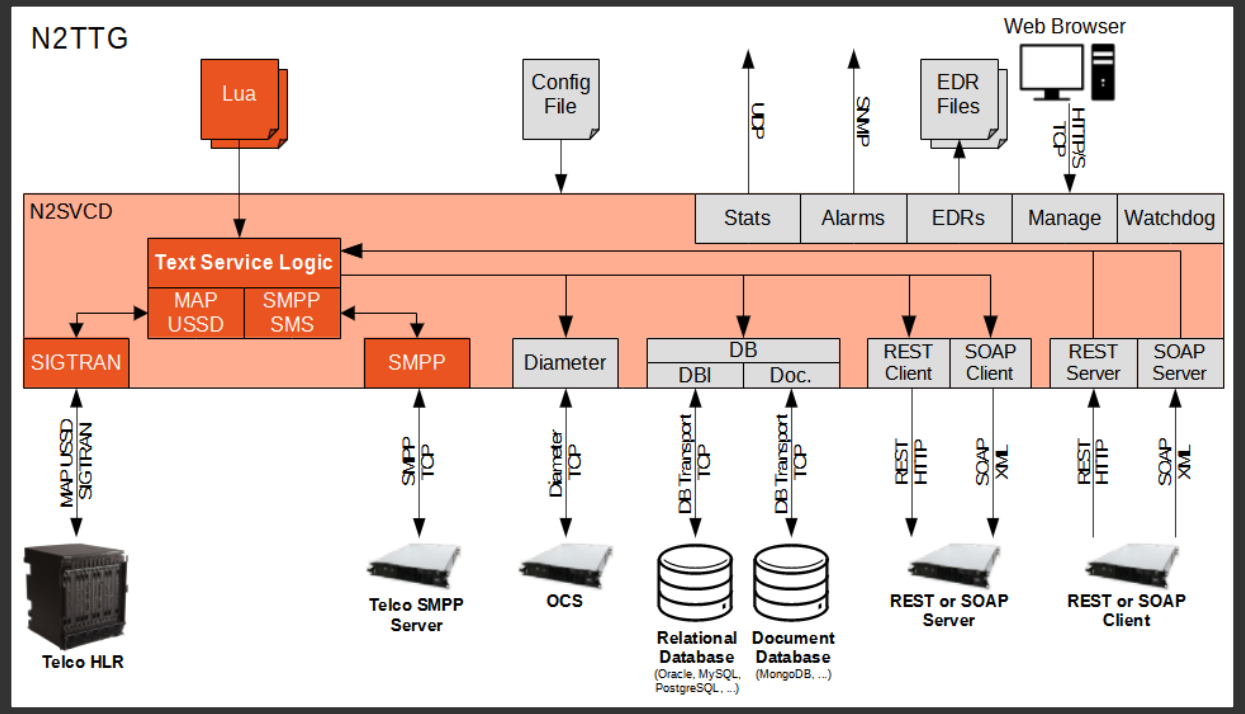
Features of the N-Squared N2TTG

- High availability (N+1 redundancy).
- Linear scalability for additional capacity and geo-distribution.
- Generic x86-64 hardware (virtualized or bare metal).
- Cost-effective, centralised subscriber interaction and notification.
- Infinitely extensible and configurable via Lua scripting.
  - Translate and transform received or configured text.
  - Retrieve and manipulate any amount of data from any amount of sources.
  - Notify subscribers, update external systems, collect statistics, etc.

Deployment

N-SQUARED SOFTWARE

Deployment Platform

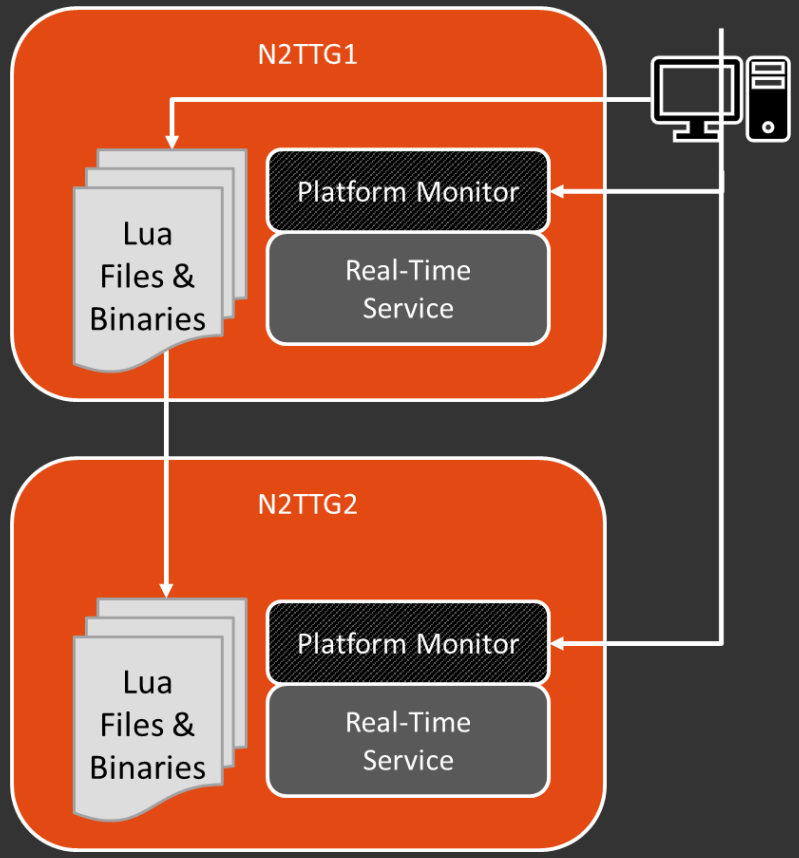


- Built on the N-Squared Service Daemon (N2SVCD) framework.
- Communicates with external entities via N2SVCD applications.
- Managed by N2SVCD configuration and management.
- Controlled by Lua files or precompiled binaries.

N-SQUARED SOFTWARE

Example Redundant Platforms

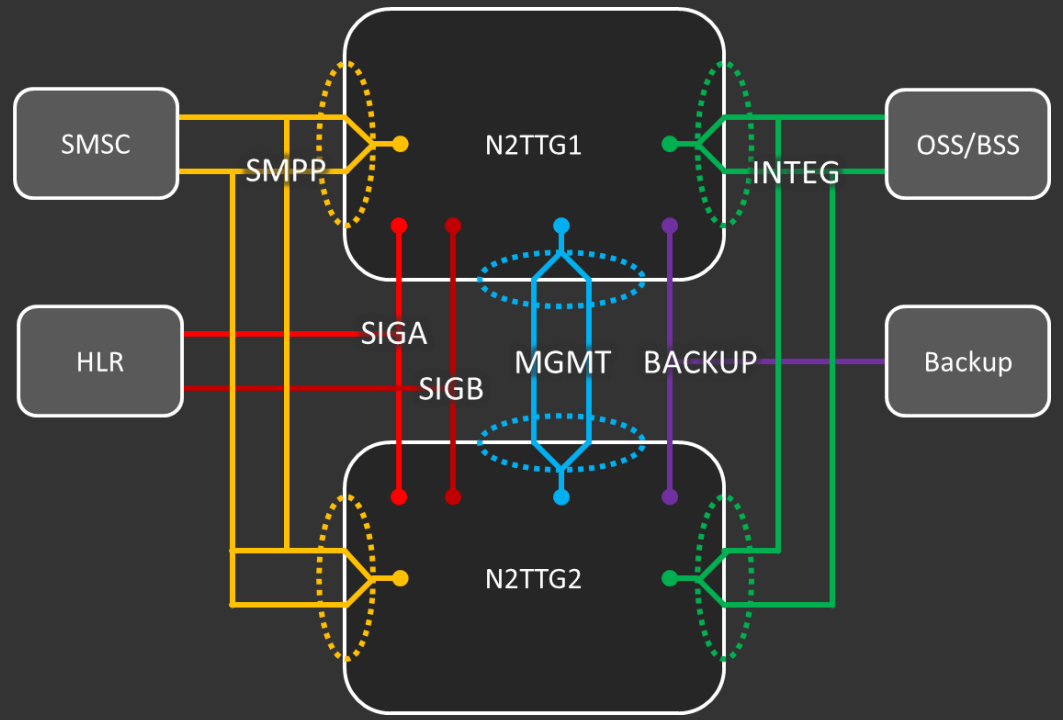
- Redundant hosts for HA
  - N+1 active/active
  - Linear scalable
- Single point of configuration
  - Convenient & consistent
  - Synchronised as required
- Direct real-time monitoring
  - Maximum responsiveness
  - Maximum resilience



N-SQUARED SOFTWARE

Example Redundant Network

- SMPP bonded network for real-time messaging.
- Data integration bonded network for message enrichment and updates.
- Management bonded network for synchronisation, statistics, and monitoring.
- SIGTRAN primary and secondary connections to core network for USSD.
- OOB backup network.



N-SQUARED SOFTWARE

Interfaces

Core network integration:

1. SMPP to SMSC (SMPP 3.4)
2. USSD/MAP to HLR (TS 29.002)
   - SCTP (RFC 2960)
   - M3UA (RFC 4666), or
   - SUA (RFC 3868)
   - TCAP (ITU-T Q.771-775)

OSS & BSS integration:

A. Alarms via SNMPv2 (RFC 3416)

B. Statistics via Etsy StatsD

C. Platform admin via HTTP/S

D. External messaging in/out via SOAP, REST, Diameter, …

E. Database integration via DBI or document storage

Note: Interfaces are implemented to the extent necessary to support advertised features.
Refer to the product Protocol Conformance Statement documentation for details.

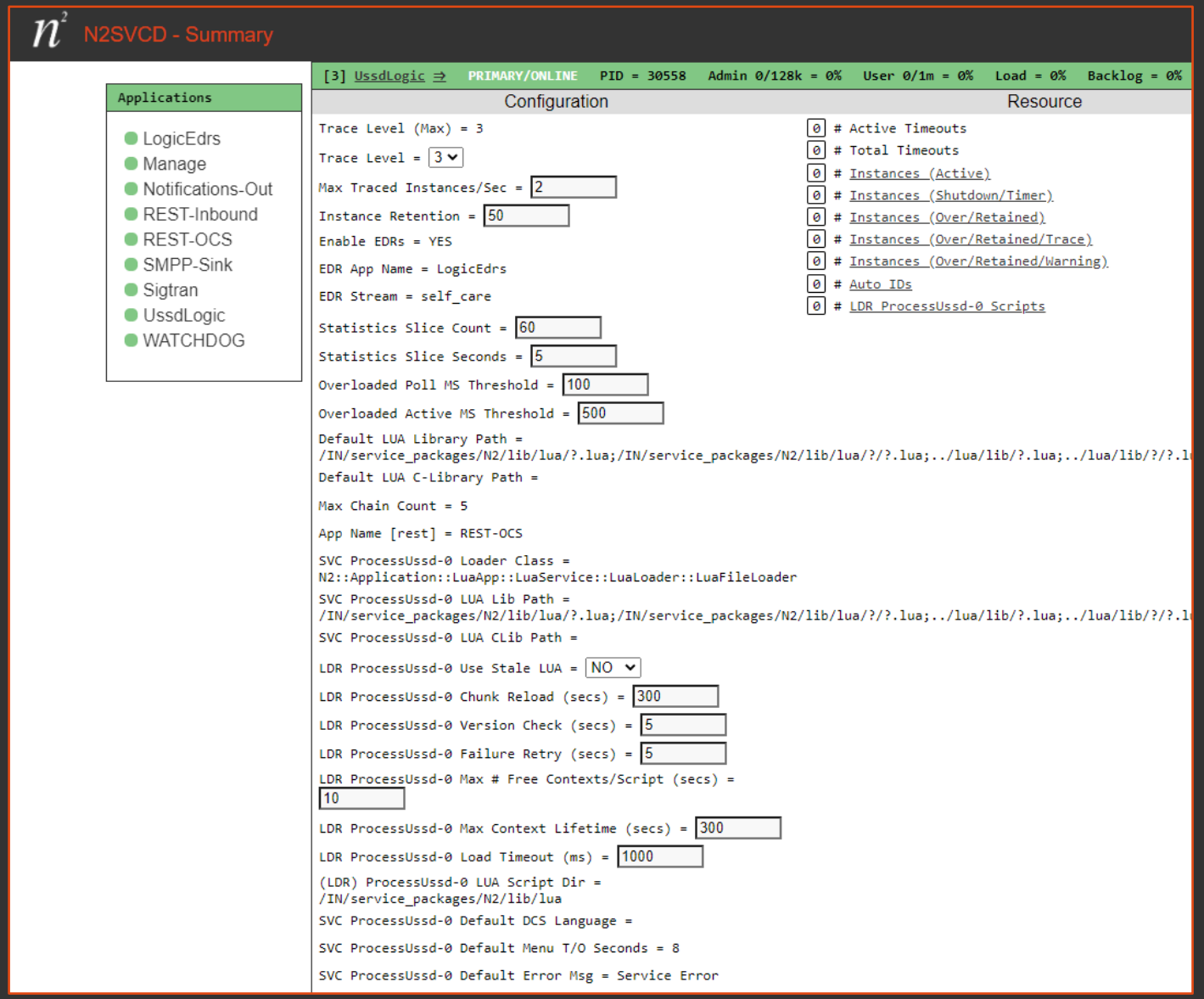
User Interfaces

Monitoring
User Interface

The run-time service execution environment for N2TTG offers an HTTP/S port to perform monitoring and system administration activities using any modern web browser.

The interface allows access to:

- In-progress interaction instances.
- Working configuration.
- Current statistics.
- Trace logs for interaction instances.



Monitoring UI (cont.)

Tracing mode can be activated for interaction instances. Tracing logs are stored in memory and can be accessed over the HTTP/S monitoring UI.

The trace output shows:

- Protocol messages in/out.
- Application and script debug and dump-level output.
- Warnings/errors.
- Timestamps & statistics.

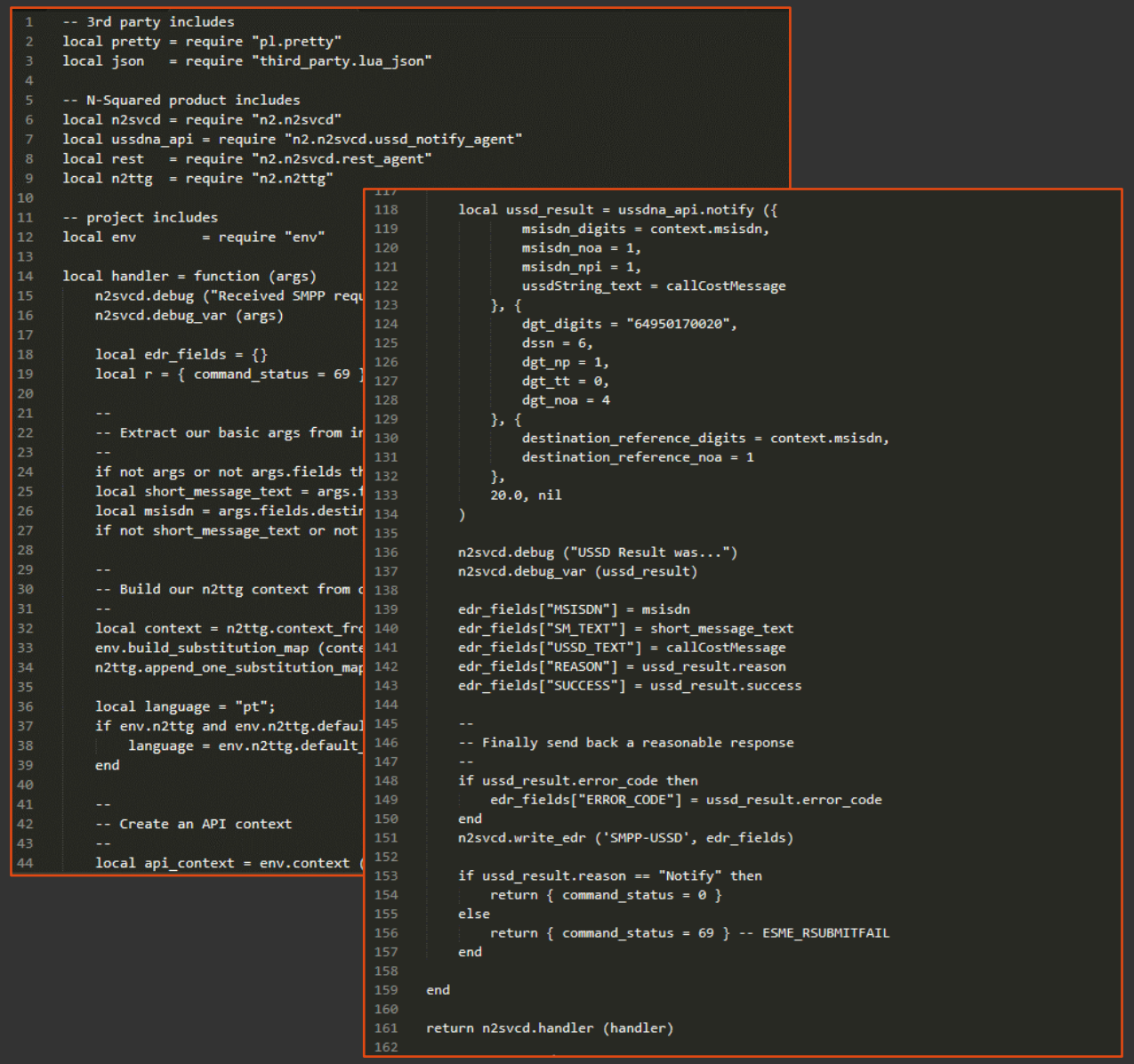
```
2022-01-20 22:32:24.745836 LuaService::LuaLoader    [trace.debug] Instance already has a script_key defined.  Proceed to phase 2 of loading.
2022-01-20 22:32:24.745927 LuaService::LuaLoader    [trace.debug] Loading instance for script key 'submit_sm' (mtime <undef>).
2022-01-20 22:32:24.745953 LuaService::LuaLoader    [trace.debug] No cached chunk.  Load chunk.
2022-01-20 22:32:24.745968 LuaService::LuaLoader    [trace.debug] No cached chunk (or too stale).  Force reload.
2022-01-20 22:32:24.746072 LuaLoader::LuaFileLoader [trace.debug] Have .lua but not .lc.
2022-01-20 22:32:24.746272 LuaService::LuaLoader    [trace.debug] Load complete for script key 'submit_sm' mtime = 1641801812.
2022-01-20 22:32:24.746300 LuaService::LuaLoader    [trace.debug] Returned loader response includes main script LUA chunk (5027 bytes).
2022-01-20 22:32:24.746319 LuaService::LuaLoader    [trace.debug] Returned mtime 1641801812.  Had no cached mtime.
2022-01-20 22:32:24.746388 LuaService::LuaLoader    [trace.debug] Create new context.  Lib Path =
'/IN/service_packages/N2/lib/lua/?.lua;../lua/lib/?.lua;../lua/lib/?.lc;../../n2ttg/lua/lib/?.lua;../../n2ttg/lua/lib/?.lua', C-Lib Path = '<undef>'.
2022-01-20 22:32:24.754353 LuaApp::LuaInstance      [trace.debug] Entering LUA after '<start>'.
2022-01-20 22:32:24.754545 LuaApp::LuaInstance      [trace.debug] Received SMPP request:
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug] {
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]   "bytes" => "\0\0\0=\0\0\0\4\0\0\0\0\0\0\5\0\1\001244123123\0\1\001244950100200\0
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]   "command_id" => 4,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]   "connection" => {
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "far_system_id" => "ncc",
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "remote_ip" => "127.0.0.1",
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "remote_port" => 60630
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]   },
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]   "fields" => {
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "data_coding" => 0,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "dest_addr_npi" => 1,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "dest_addr_ton" => 1,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "destination_addr" => "244950100200",
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "esm_class" => 0,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "priority_flag" => 0,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "protocol_id" => 0,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "registered_delivery" => 0,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "replace_if_present_flag" => 0,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "schedule_delivery_time" => "",
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "service_type" => "",
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "short_message" => "'\0020.28'",
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "short_message_text" => "'\$0.28'",
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "sm_default_msg_id" => 0,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "sm_length" => 7,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "source_addr" => 244123123,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "source_addr_npi" => 1,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "source_addr_ton" => 1,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "tlvs" => {},
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]     "validity_period" => ""
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]   },
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]   "pdu" => "submit_sm",
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]   "sequence_number" => 5,
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug]   "test_details" => {}
2022-01-20 22:32:24.755264 LuaApp::LuaInstance      [trace.debug] }
```

Lua Scripting

Lua Scripting

All subscriber interaction is driven by scripting using Lua, the lightweight, fast, simple, and widespread scripting language:

- Extensive library of fully-documented N2TTG and N2SVCD APIs to simplify and extend scripts.
- Access to all inbuilt features of Lua.
- Freedom to use any third-party or custom Lua library or API as necessary.
- Hot reload of changed scripts.

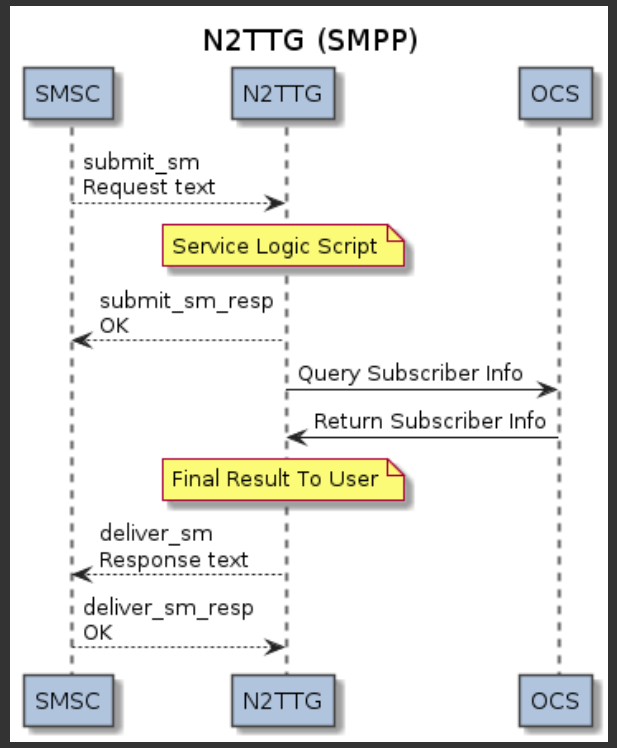


```lua
-- 3rd party includes
local pretty = require "pl.pretty"
local json   = require "third_party.lua_json"

-- N-Squared product includes
local n2svcd = require "n2.n2svcd"
local ussdna_api = require "n2.n2svcd.ussd_notify_agent"
local rest   = require "n2.n2svcd.rest_agent"
local n2ttg  = require "n2.n2ttg"

-- project includes
local env        = require "env"

local handler = function (args)
    n2svcd.debug ("Received SMPP requ
    n2svcd.debug_var (args)

    local edr_fields = {}
    local r = { command_status = 69 }

    --
    -- Extract our basic args from ir
    --
    if not args or not args.fields th
    local short_message_text = args.f
    local msisdn = args.fields.destir
    if not short_message_text or not

    --
    -- Build our n2ttg context from c
    --
    local context = n2ttg.context_fro
    env.build_substitution_map (conte
    n2ttg.append_one_substitution_map

    local language = "pt";
    if env.n2ttg and env.n2ttg.defaul
        language = env.n2ttg.default_
    end

    --
    -- Create an API context
    --
    local api_context = env.context (

    local ussd_result = ussdna_api.notify ({
            msisdn_digits = context.msisdn,
            msisdn_noa = 1,
            msisdn_npi = 1,
            ussdString_text = callCostMessage
        }, {
            dgt_digits = "64950170020",
            dssn = 6,
            dgt_np = 1,
            dgt_tt = 0,
            dgt_noa = 4
        }, {
            destination_reference_digits = context.msisdn,
            destination_reference_noa = 1
        },
        20.0, nil
    )

    n2svcd.debug ("USSD Result was...")
    n2svcd.debug_var (ussd_result)

    edr_fields["MSISDN"] = msisdn
    edr_fields["SM_TEXT"] = short_message_text
    edr_fields["USSD_TEXT"] = callCostMessage
    edr_fields["REASON"] = ussd_result.reason
    edr_fields["SUCCESS"] = ussd_result.success

    --
    -- Finally send back a reasonable response
    --
    if ussd_result.error_code then
        edr_fields["ERROR_CODE"] = ussd_result.error_code
    end
    n2svcd.write_edr ('SMPP-USSD', edr_fields)

    if ussd_result.reason == "Notify" then
        return { command_status = 0 }
    else
        return { command_status = 69 } -- ESME_RSUBMITFAIL
    end

end

return n2svcd.handler (handler)
```

Message Flows

N-SQUARED SOFTWARE

SMPP Request/Response



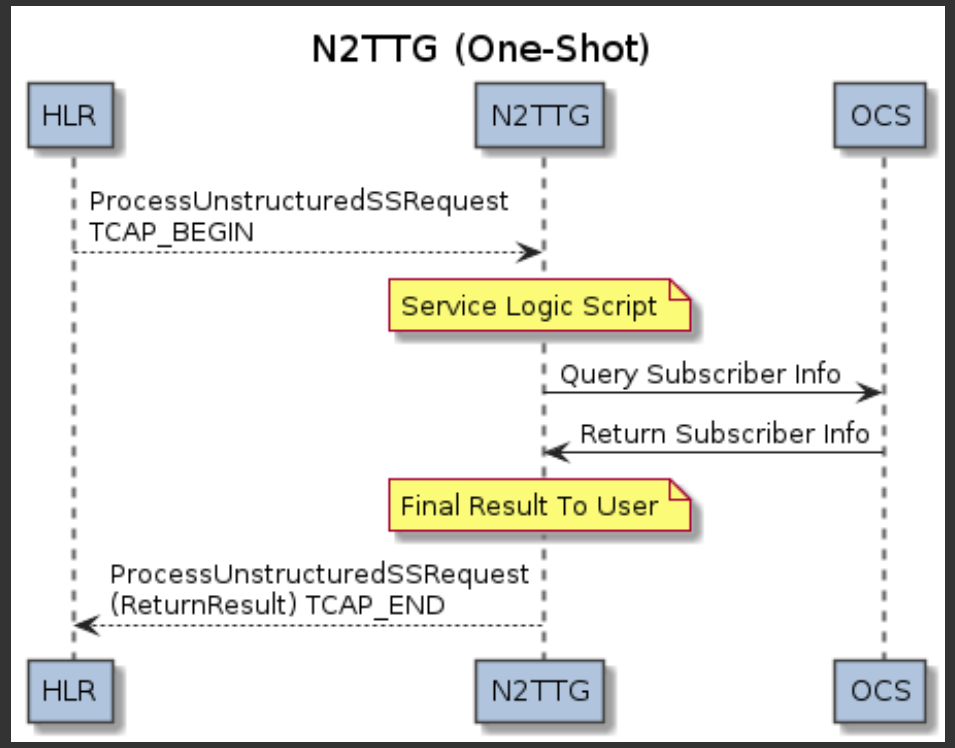
The subscriber sends a request SMS to a service number.

N2TTG determines the appropriate script to run and executes it.

Additional information is gathered and a response SMS is sent to the subscriber.

N-SQUARED SOFTWARE

USSD Single Request/Response



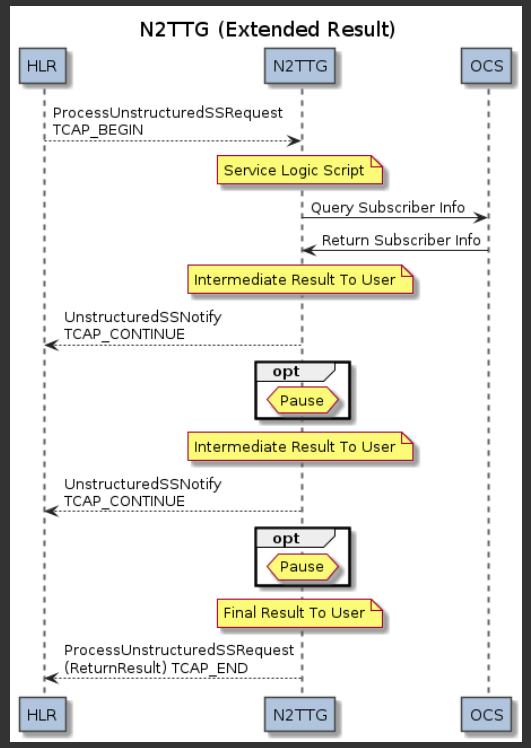
The subscriber sends a USSD request to a service code.

N2TTG determines the appropriate script to run and executes it.

Additional information is gathered and final response USSD text is sent to the subscriber.

N-SQUARED SOFTWARE

USSD Extended Request/Response

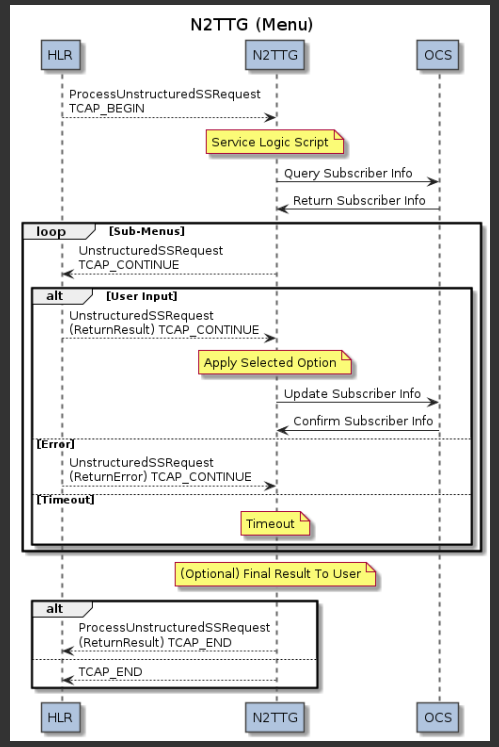


As for the single request/response case, but multiple response messages with additional information are sent.

Pauses are used to give the subscriber time to read each message.

N-SQUARED SOFTWARE

USSD Menu



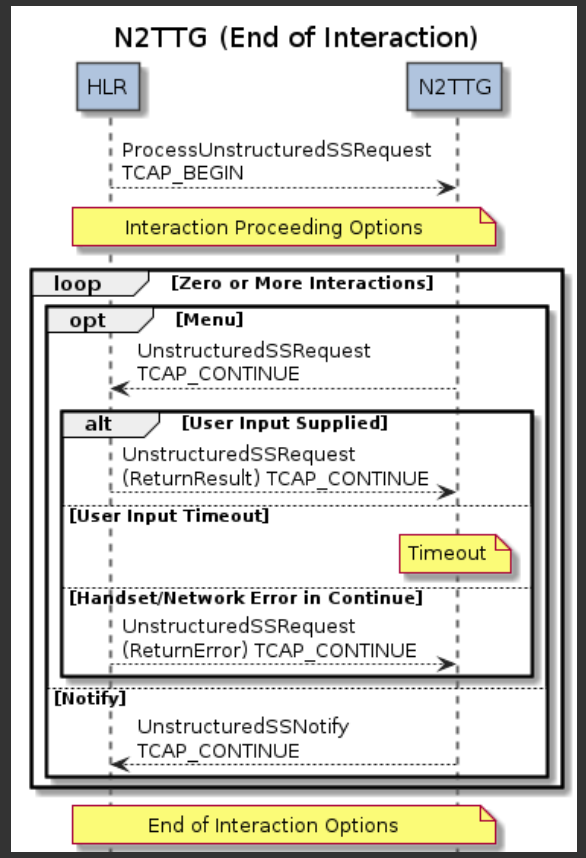
USSD message flows may also be interactive menus.

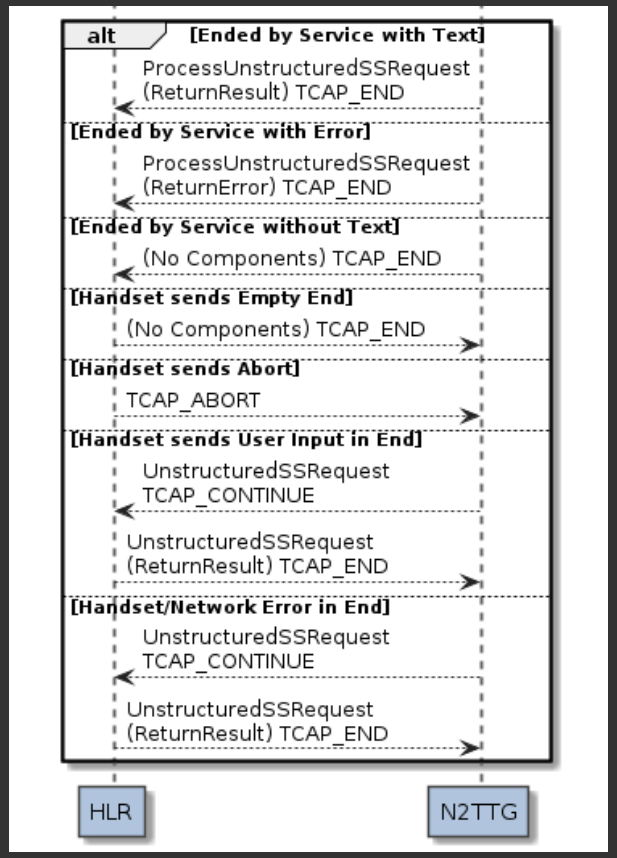
Each menu that is presented allows the subscriber to enter single- or multi-character responses in order to drive the request forward.

There is no limit to the depth or re-use of menus in interaction.

N-SQUARED SOFTWARE

USSD Interaction End





There are multiple ways that a USSD session with a subscriber may be ended:

- Subscriber side for no response, error, etc.
- N2TTG side for bad input, notification, etc.

Conclusion