Using TPNS as a Tool for Prototyping Software

By
Melvyn Feuerman
10/26/2001

Introduction

The classic problem that occurs in every major software project is very simply stated: How do you test individual software and hardware modules when connecting links are not available? For example, consider the requirements for testing ATM/Host acquirer and issuer software for a Shared ATM Network shown in Figure 1.

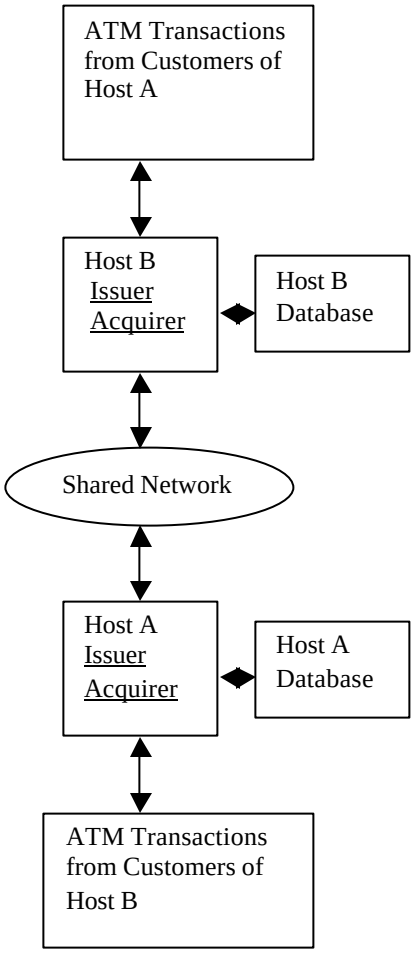

Figure 1
A Shared Network with real resources

The *acquirer* institution owns the ATM on which the customer enters the transaction while the *issuer* institution is the issuer of the customer's ATM card. Here is a summary of the transaction path for a customer "get cash" transaction.

- An ATM connected to a financial institution A (Host A) accepts a cash withdrawal request from a customer of financial institution B (Host B). In this path Host A is the *acquirer* host and Host B is the *issuer* host.
- The ATM sends the customer request to acquirer Host A in *ATM to Host Format* (ATF). Host A validates the ATM message, converts it into *Shared Network Format* (SNF), and sends it to Host B via the Shared Network.
- The Shared Network validates the format of the Host A acquirer SNF request before routing it to issuer Host B.
- Host B issuer software "decodes" the SNF cash request message from the Shared Network and uses the customer's account number, pin and account balance to accept or reject the cash request. The Host B issuer software returns an authorization or denial response to the Shared Network.
- The Shared Network matches the response from Host B against the original request from Host A and creates a response message to Host A.
- Host A receives the SNF response from the Shared Network and sends a response to the ATM in ATF.
- The ATM completes the transaction by either (a) dispensing cash to the customer with a display of an account balance or (b) displaying a negative response to the customer such as "insufficient funds" or pin error.
- Visa-versa, customers of Host A can enter transactions on the ATM connected to Host B. In this flow, Host B is the *acquirer* and Host A is the *issuer*.

The individual hardware and software components in this example are similar to links in a chain because all the components are required for successful execution of a transaction. The question: How to test the ATM software when ATM/Host A/Shared Network/Host B links are unavailable?

A similar problem for testing Host A software: Host A acquirer software developers need *both* messages from the ATM and a response from the Shared Network to test the Host A acquirer software. Host A issuer software developers need requests from Host B via the Shared Network to test issuer paths. The answer: Prototype terminals and hosts during testing using TPNS to simulate the missing links in the chain.

About Teleprocessing Network Simulator (TPNS)

Teleprocessing Network Simulator (TPNS) is the premier mainframe product developed and used by IBM for automated support of regression and stress testing - two important types of testing for all systems including the new world of web-enabled systems.

- **Regression testing of existing applications:** This testing validates that you did not "break" the current existing software with the rollout of the new software functionality. Regression testing involves re-testing the existing transactions and typically does not involve testing the new functionality of the software.
- **Stress and exception testing:** This testing creates and tests exceptional conditions, which may be difficult to create with real hardware. Examples of exception testing are "timeout" and "stress" testing for periods of high volumes.

An often-overlooked use of TPNS is its use as a tool for prototyping terminal/client/server/host software to support development testing. TPNS has a rich and flexible programming language called STL that can be used for prototyping new messages in almost any format. TPNS can simulate SNA devices, X.25 networks, as well as TCP/IP clients and servers.

TPNS is primarily a tool for creating transaction data, "under the hood", on the message level between a terminal and a host or between two hosts. TPNS has a logging feature that provides a trace of all messages sent and received by real or simulated terminals. This allows developers to review a message log to verify the format of real or simulated messages.

Using TPNS "Boiler Plates" to Test ATM and Host A Software for the Shared Network

Next, let us see how we can use TPNS to help test the ATM and Host A software for the Shared Network. The objective here is to allow developers to *move ahead with testing when a "missing link"* is found in the testing chain. There are three basic TPNS paradigms or "boiler plates" used to prototype software for Host A and ATM testing:

- **TPNS as a Responder for testing the ATM software**: TPNS simulates Host A acquirer returning account balances/return codes to the ATM. Here TPNS simulates Host A accepting and returning a response to the ATM in ATF.
- **TPNS as a Requestor for testing the Host A issuer software**: TPNS simulates Host B sending issuer requests to Host A via the Shared Network. Here TPNS sends messages to Host A as issuer in SNF.
- **TPNS as both a Requestor and Responder for testing the Host A acquirer software:** TPNS prototypes both an ATM requesting information to Host A and the Shared Network responding to Host A. TPNS simulates messages to Host A in ATF format and responses to Host A in SNF.

Using TPNS to Test the ATM Software

We begin by using TPNS to simulate the Host A acquirer software for testing the ATM messaging and screen transition software. The key to using a prototyping tool such as TPNS is to only prototype the software **directly connected to the modules** under test. For example, in Figure 2a, the transaction entered by a customer of Host B (1) results in four messages generated between (2) - (4). Note however, to test the ATM/Host A interface we only need to simulate a response from the **Host A acquirer software** (2) > (3). Software components (4) and (5) do not have to be simulated, as **they do not directly interface** with the ATM under test.

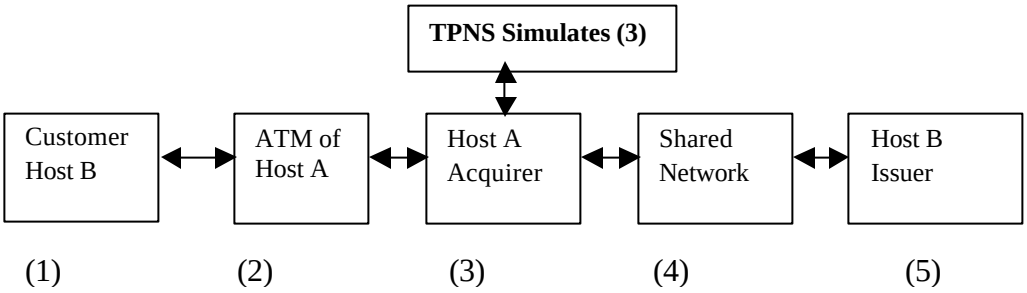

Figure 2a

TPNS Simulates Host A Acquirer Software

```
THOSTA:  msgtxt
onin then recvdata=data
wait until onin
do forever
  log recvdata
  act  = substr (recvdata,6)
  bal  = '100,000'
  rc  =  'YES'
  response = act ||bal ||rc
  type response
  transmit and wait until onin
end
endtxt
```

Test of the Real ATM

Send request to Host A with:

- Account
- PIN
- Transaction code
- Dollar amount
- Display balance
- Dispense cash

Figure 2b
Test of the real ATM using TPNS to simulate Host A

ATM Software Testing Normal Transaction Paths

Figure 2b illustrates the use of TPNS to test the "normal" ATM path with a simulated Host A. The ATM developers have set up a connection from the ATM to THOSTA, a prototype of Host A. THOSTA returns balances to the ATM **(ATF)** "as if they came from Host B".

The objective here is to validate the following ATM software functionality:

- Formatting and transmitting customer request messages to Host A
- Receiving and formatting the response from Host A
- Translation of the response message into screen readable format
- Proper display of balance information and text returned by Host A
- Correct dispense of cash to a customer on a "good" response
- Screen transitions after receiving balance and return code from Host A.

The TPNS simulation of Host A acquirer software illustrates some of the basic components of the TPNS STL language, much like BASIC, with special verbs for sending, receiving and parsing messages. In this example, the TPNS paradigm is of type "**responder**" because TPNS prototypes a host responding to requests from a terminal.

- MSGTXT defines THOSTA as the name of the "executable" message deck
- WAIT UNTIL ONIN – wait for an input message
- DO FOREVER -- perform statements between Do and End
- The input message stored in RECVDATA is logged
- ACT, a 6-digit account number is parsed from RECVDATA
- TPNS variables -- BAL: A ledger balance of 100,000 and RC: a return code are concatenated and stored in Response
- TYPE sets up Response in the output Buffer
- TRANSMIT returns a response to the ATM.

ATM Software Testing Exception Paths

A very useful feature of TPNS is its use in testing terminal and host timeout software. In this section, I show how to use the TPNS suspend statement to test the ATM timeout and reversal software as shown in Figure 3a.

When an ATM sends a cash request to a host, it also sets up a timeout value of 60 seconds for example, that specifies how long the ATM will *wait* for a host response. If the timer expires, the ATM reversal software will "kick in" and send a reversal of the original cash withdrawal transaction. The ATM will continue to send cash reversals until receiving an acknowledgment of the reversal from Host A.

A slightly revised version of THOSTA, to help test ATM timeout of 60 seconds and cash reversal functionality, is illustrated in Figure 3b. The basic idea is to set a delay in the TPNS program greater than the ATM timeout value.

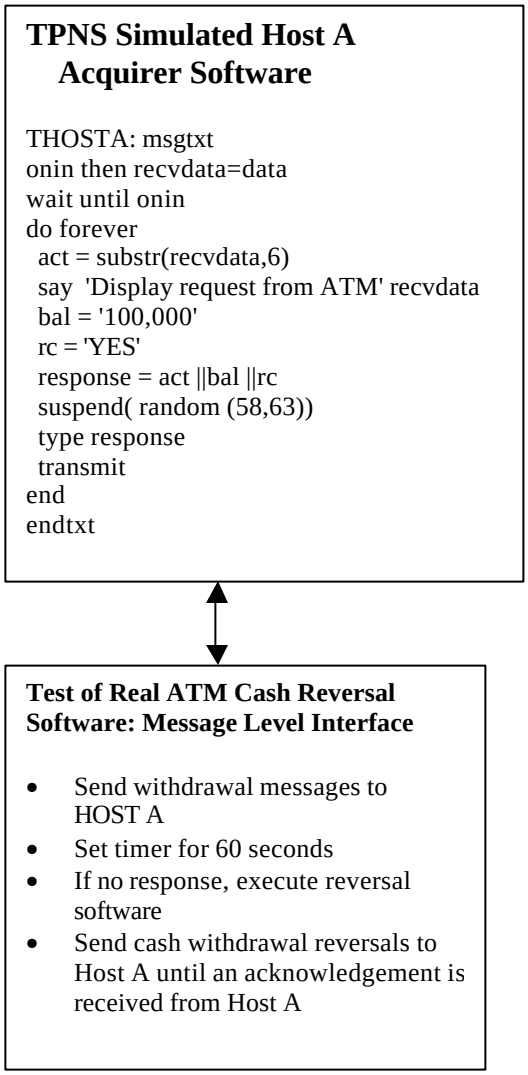

Figure 3b
Test of the ATM timeout and reversal code using TPNS

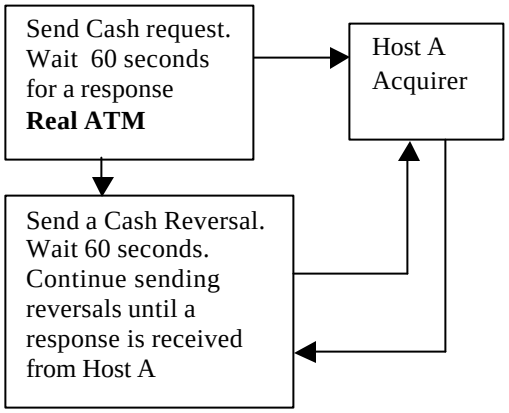

Figure 3a
ATM Timeout and Cash Reversal Flow

The test plan for testing the ATM reversal code using TPNS to simulate a late response from Host A is shown in Figure 3b.

Using THOSTA to test ATM timeout/reversal

- The ATM sends a cash withdrawal to THOSTA
- The ATM sets a timeout "window" of 60 seconds
- The THOSTA SAY statement displays the ATM input request
- THOSTA uses the suspend(random (58,63)) statement to randomly simulate delayed responses between 58 and 63 seconds to the ATM.

Expected Results of the ATM timeout reversal test

- The ATM should send a cash withdrawal reversal whenever THOSTA does not respond within the 60 second timeout window (suspend > 60)
- The ATM should ignore a THOSTA response from the original withdrawal transaction for suspend > 60
- The ATM should send reversals until an acknowledgment of a reversal is received from THOSTA for suspend <60.

Using TPNS to Test the HOST A Issuer Software

Next we show how to use TPNS to test the Host A issuer software. Consider the issuer flow in Figure 4a from the entry of a transaction (1) by a customer of Host A on an ATM connected to Host B. Again the key to using TPNS is to only simulate **a message from a host or terminal directly connected to the software under test**. In order to test HOSTA issuer software we only need to simulate a request from the Shared Network (4) > (5). Software components (1), (2) and (3) do not have to be simulated, as **they do not directly interface** with the Host A issuer software under test.

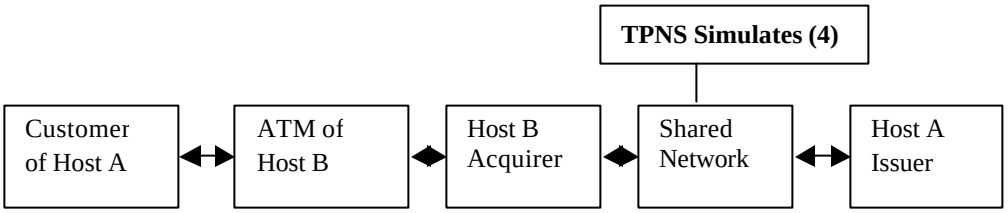

Figure 4a

Host A Database

Account	Balance	PIN
345678	100,000	ABCD

Test of Host A Issuer Software

- Accept a message request from the Shared Network in Shared Network Format **(SNF)**
- Validate the pin in the input against the customer database
- Inquiry/Update account database
- Create a response and transmit the response to the Shared Network

TPNS Simulates Host B Issuer Requests via the Shared Network

```
issuer: msgutbl
  '345678,INQ, ,ABCD'
  '345678,CASH,50.00,ABBB'
  '345678,CASH,50.00,ABCD'
  '345678,CASHRV,50.00,ABCD'
endutbl

THOSTB: msgtxt
do i = 0 to utblmax(issuer)
  message =  utbl(issuer,i)
  type message
  transmit
end
endtxt
```

Figure 4b
TPNS simulates issuer requests from Host B via the Shared Network

The Test of Host A Issuer Normal Paths

The TPNS program THOSTB shown in Figure 4b simulates a Shared Network request message as if it originated from Host B. The TPNS paradigm is of type "**requestor**" because TPNS is simulating requests to the Host A issuer software.

The TPNS program has two components consisting of a table definition and executable code:

- **MSGUTBL** defines a table of constants similar to the BASIC DIM statement for defining arrays:
  - Account number
  - INQ = Inquiry, CASH = Get Cash , CASHRV = Reversal
  - Amount for debit transaction, zero amount for inquiries
  - PIN entered by the customer.

- **MSGTXT** defines the THOSTB executable deck
  - DO I statement with a matching END statement causes the program to loop through all the elements in msgutbl
  - TYPE message sets up message in the transmit buffer
  - Transmit sends the issuer request to Host A in shared network format

- **THOSTB** reads the table defined in ISSUER and creates/sends an issuer message in SNF to Host A. The script contains the following transactions for the customer with account number 345678:
  - Account inquiry for pin ABCD (normal transaction)
  - Cash withdrawal 50.00 for pin ABBB (test of BAD pin)
  - Cash withdrawal 50.00 for pin ABCD
  - Cash withdrawal reversal of 50.00 for pin ABCD

TPNS Shared Network Responses

```
TSHARE: msgtxt
onin then recvdata=data
wait until onin
do forever
  act  = substr (recvdata,6)
  bal  = '100000'
  rc  = 'YES'
  response = act ||bal ||rc
  type response
  transmit and wait until onin
end
endtxt
```

Test of Host A Acquirer

- Validate ATM Input
- Create request to the Shared Network
- Accept/Match response with the original ATM request
- Respond to ATM

TPNS Simulated ATM

```
atmtxs: msgutbl
   '345678,INQ, ,ABCD'
   '345678,CASH,50.00,ABCD'
   '222222,INQ, ,DBCA'
   '222222,CASHRV,50.00,ABCD'
endutbl


TATM: msgtxt
do i = 0 to utblmax(atmtxs)
  message = utbl(atmtxs,i)
  type message
  transmit
end
endtxt
```

Figure 5b
TPNS simulation of both

- ATM Requests
- Shared Network Responses

Using TPNS to Test the Host A Acquirer Software

The TPNS setup for testing Host A acquirer software is a bit more complex than testing ATM or issuer software because it involves simulating both ATM requests and responses from Host B via the Shared Network.

Figure 5a
Host A acquirer flow

Test of Host A Acquirer Normal Transaction Paths

The test of the Host A acquirer involves validation of the following:

- Receiving and "decoding" the ATM cash request for a customer of Host B
- Formatting a request to issuer B via the Shared Network
- Receiving and formatting the response from Host B via the Shared Network
- Matching the Host B response received with the original ATM(B) cash request

TPNS Simulates the ATM and the Shared Network

- TATM simulates ATM requests to Host A in ATM format
- TSHARE simulates responses to Host A

TPNS Simulation of the ATM Requests

TATM is a requestor paradigm similar in structure to the Host B issuer **requestor** program in Figure 4b.

- ATMTXS defines the transaction table
- TATM sends the transactions to Host A

The first transaction simulates an ATM inquiry request for customer account 345678 with a pin of ABCD.

TPNS Simulation of Shared Network Responses

The second component in the network is called TSHARE, which simulates Shared Network responses to Host A:

- Do Forever - The network runs continuously receiving requests from Host A
- Wait Until an input message is received from Host A
- Copy the account number from the input message
- Prepare and send a response in shared network format

Note, TSHARE is a **responder** and has the same basic structure of THOSTA (Figure 2a) which responds to ATM requests. Here TSHARE provides responses to Host A in SNF while THOSTA provides responses to the ATM in **ATF.**

Conclusion

Using a prototyping tool such as TPNS can be very useful in a development environment when a project includes parallel and interdependent software efforts. A generic TPNS prototype is particularly useful during the initial stages of software testing because it allows parallel development groups to move ahead independently on workstations, servers, and HOST systems. I have presented the basic TPNS "boiler plates" (requestor and responder) that will help you build your own prototypes of almost any financial system.

More information on TPNS can be found on the IBM web site:
http://www-4.ibm.com/software/network/tpns/

Comments and questions about this article are welcome. Send me a note at TPNS@earthlink.net.

Mel Feuerman