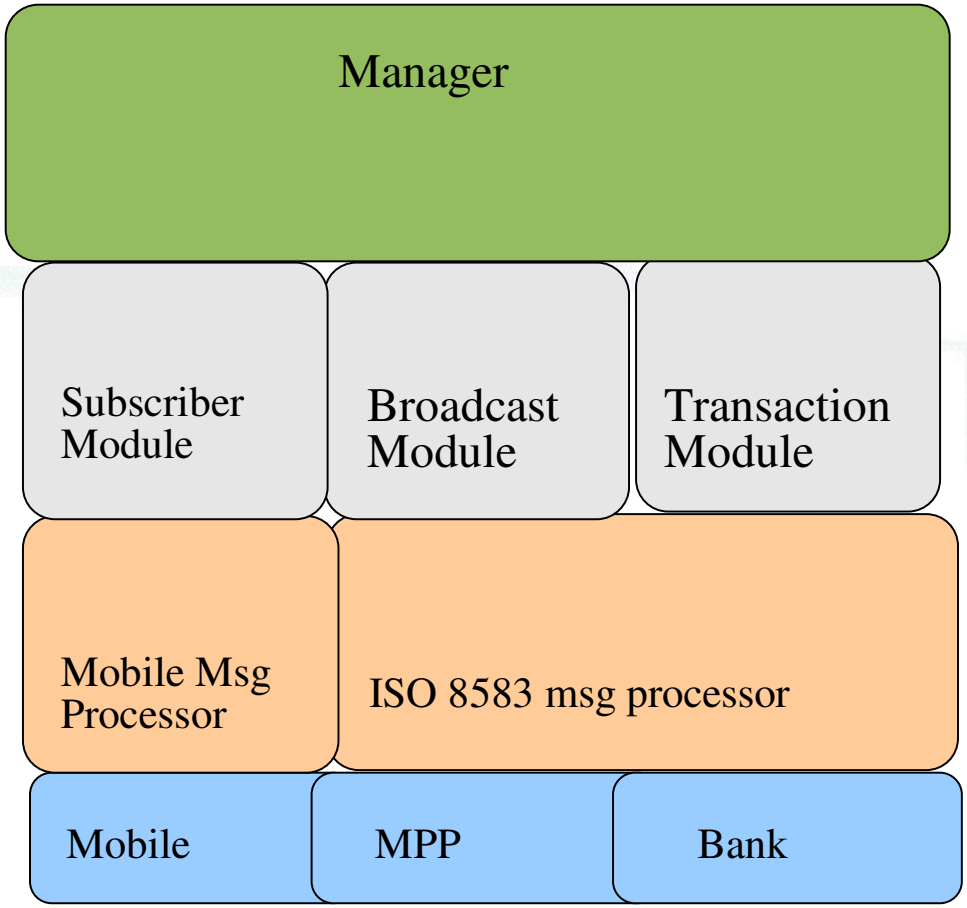
MPP/Bank Reference Design

T.A. Gonsalves

TeNeT Group
Dept of CSE, IIT Madras

2nd December 2009

Mobile Payment Provider Software



Layer 4: Application Manager

Layer 3: Functional Modules

Layer 2: Message Processor

Layer 1: Network Connections

Network Layer Functions

- The modules at this layer establish an encrypted TCP connection with the Bank and the other MPP
- Responsibility of re-establishing a connection in case of connection failure
- Performs key exchange and enables encryption of data
- Implementation of queues to prevent overflow of messages
- Input not standardised in case of the mobile transport module

Message Processor Layer Functions

- Mobile Message Processor
  - Receives messages via 'Mobile' interface of Layer 1
  - Parses and converts to a relevant message object
  - Forms and transmits messages to the mobile via Layer 1

- ISO Message Processor
  - Receives messages via 'Bank' and 'MPP' interfaces of Layer 1
  - Parses and sends to the upper layer
  - Data received from the upper layer is constructed into an ISO message and sent to Layer 1

Functional Layer

- Transaction Module
  - Receives messages from the lower layer and creates a transaction object
  - The object keeps track of the transaction state
  - The transaction object may be kept in memory (volatile) until the request reaches the state where the Bank is informed. At that time the object is written to disk (persistent)
- Broadcast Module
  - Handles registration details, conflict resolution
  - Performs lookup for MPP id corresponding to a given mobile number
  - Manages the cache

… Functional Layer

- Subscriber Module
  - Handles messages from the Mobile Message Processor module in Layer 2
  - Receives messages from other modules to send to the mobile via the Mobile Message Processor

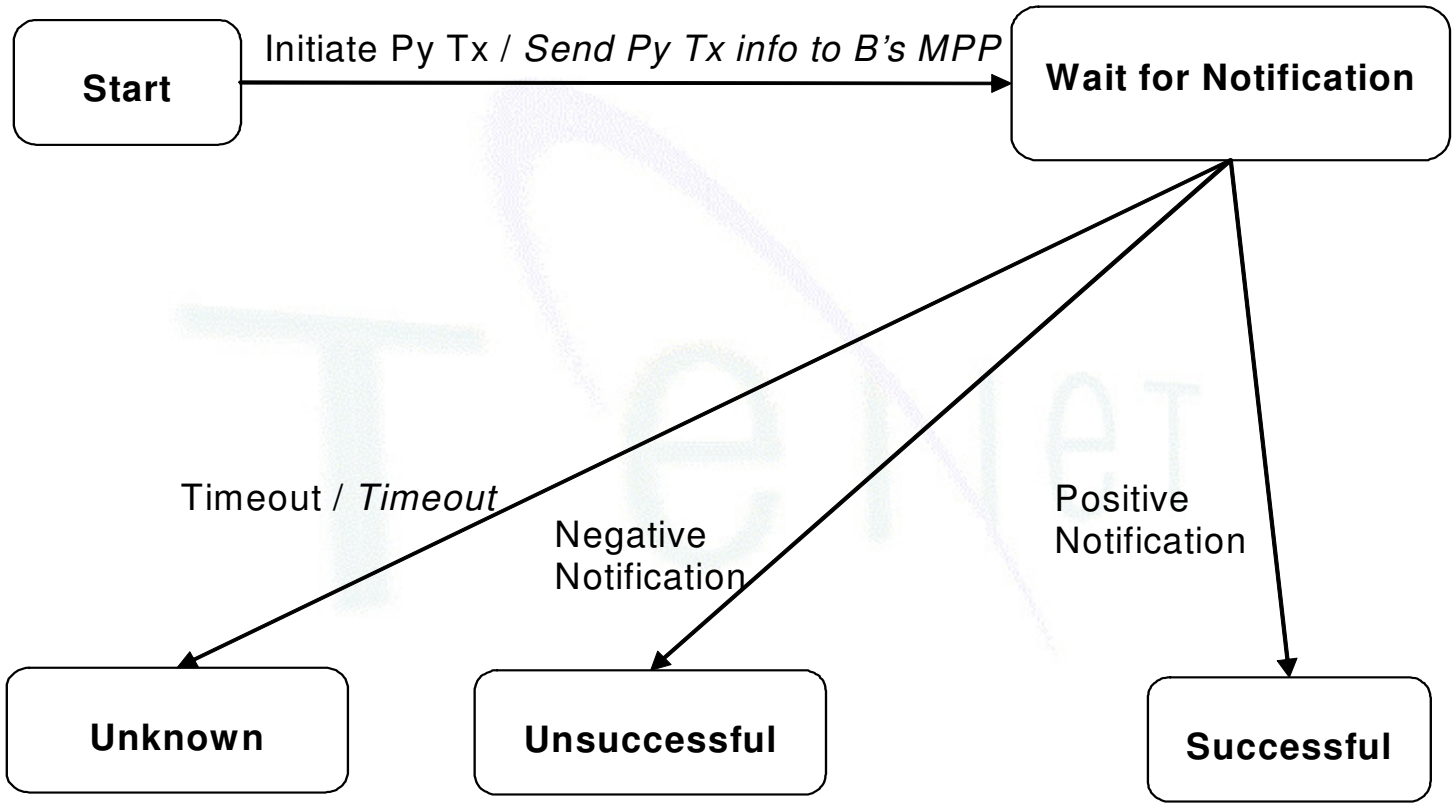
Application Manager

- Manages the transactions by issuing requests to different modules in Layer 3

FSMs

- The protocol is defined in terms of FSMs (finite state machines)
- Helps in:
  - Precise specification of the protocol
  - Test-case generation
  - Implementation

An FSM (Beneficiary Pull)



Implementing FSMs: Switch

```
Switch (state) {
    Case (Start):
        Switch (event)
            Case (Initiate Payment Transaction):
                Send Payment transaction  info to Beneficiary's MPP
                Change state to 'Wait for Notification'
    Case (Wait for Notification):
        Switch (event)
            Case (Initiate Payment Transaction):
                Ignore duplicate request
            Case (Positive notification):
                Notify Beneficiary
                Change state to Successful
            Case (Timeout):
                Notify Beneficiary
                Change state to Unknown
    Case (default):
        Handle "unexpected state" error
    }
```

Implementing FSMs: Dispatch Table

int (*dispatchTab[NUM_STATES, NUM_EVENTS])();

State / Event	Start	Wait for Notification	Successful	Unsuccessful	Unknown
Initiate Payment Transaction	SendToBenMPP	Ignore	…	…	…
Positive Notification	Ignore	NotifyBen			
Negative Notification	Ignore	NotifyBen			
Timeout	Ignore	NotifyBen			

result = (*dispatchTab[state, event])();

Golden Rule of Protocol Implementation

- Be liberal in what you accept
  Be conservative in what you send
- Eg: ISO8583 Currency field is not specified in MPFI messages
- When generating messages, set to INR
- When receiving messages:
  - if set to INR, accept
  - if set to some other currency, reject
  - if not set, assume INR and accept

Summary

- Reference MPP design is a 4-layered model:
  - Application Manager
  - Function Modules
  - Message Processor
  - Network Connections
- An FSM gives a precise specification of the protocol
  - Implemented using *Nested Switch* or a *Dispatch Table with function pointers*
- Be *conservative* in *sending* and *liberal* in *accepting* a message