[ATMSD-2] Withdrawal function test for ATM System

Created: 23/Jun/25 11:40 AM - Updated: 24/Jun/25 8:57 AM

Status:	Backlog
Project:	ATM System Design
Component/s:	ATM Application, Bank Application

Type:	Task	**Priority:**	Medium
Reporter:	Roland Traier-Kiss [Midori]	**Assignee:**	Jack Powell
Resolution:	Unresolved		
Labels:	ATM, interactions		

Test Details

Estimated execution time (h):	6.5

Approvals

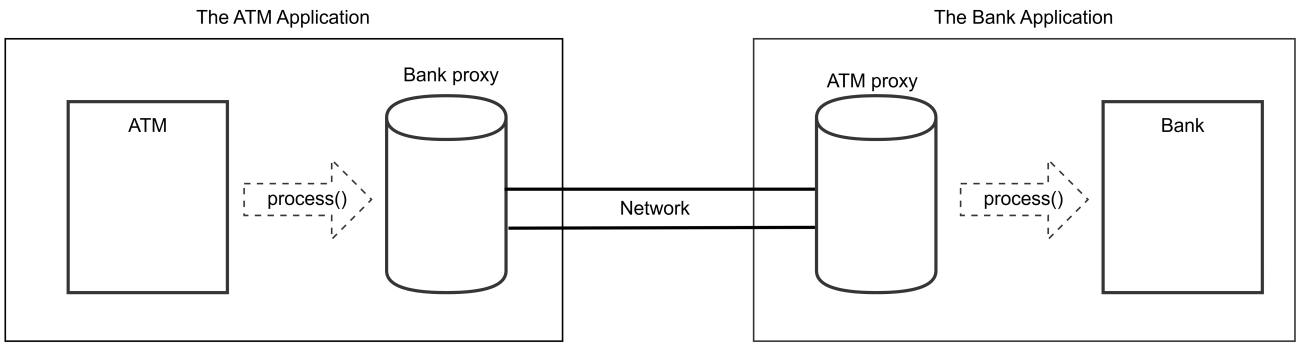
Approved by:	Casey Ford, Jack Powell, Daike Tanaka
Final approval date:	23/Jun/25

Execution Requirements

Assets required for execution:	Test-ready ATM system, ATM display, Keyboard, Receipt printer, Cash dispenser
Security clearance required for execution:	C2

Description

Test case of basic function "Withdrawal" to verify that the implementation is basically correct.



At this point, the `ATM` needs to send a message to the `Bank` object, asking it to process a transaction (passing the Withdraw transaction object as an explicit argument). The `ATM` object lives in one address space (the `ATM` application) but the `Bank` lives in a different address space (the `Bank` application). We will employ proxies to make the `ATM` and the `Bank` objects viewed in the same address space.

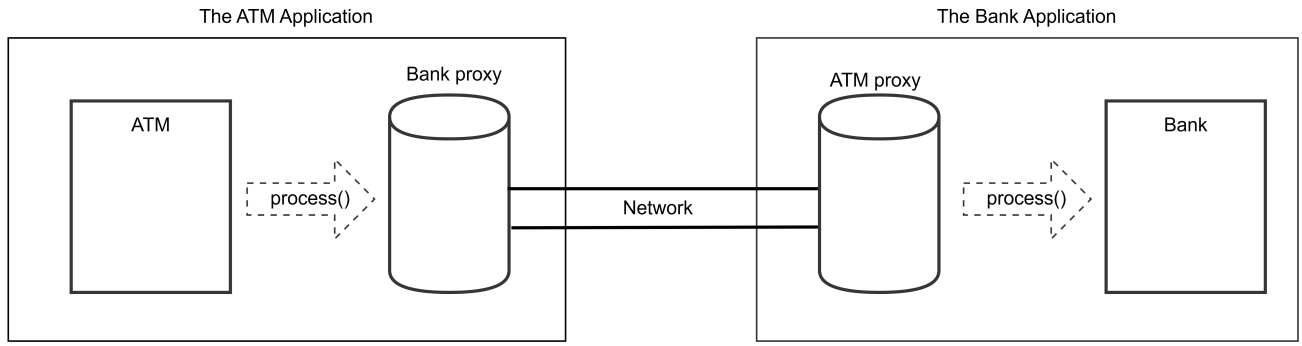
The `ATM` cannot send a direct message to a `Bank`, so it sends a message to a `Bank` proxy that lives in the `ATM`'s address space (see attachment). This proxy packs up the request and transaction object and ships it across the network to an `ATM` proxy that lives in the `Bank`'s address space. The `ATM` proxy unpacks the request, reconstitutes the transaction object, and sends the process message to the real `Bank` object.

The real `ATM` and `Bank` are completely unaware that they are really talking to proxies. This allows us to ignore the distributed facet of a distributed application during high-level design, leaving the gory details to low-level design proxy classes.

Test Case	Name	Priority	Status	Objective	Precondition
ATMSD-T1 (1.0)	ATM - Withdrawal function test	Normal	Approved	Test case of basic function "Withdrawal" to verify that the implementation is correct.	Assets: • Test-ready ATM system • ATM display • Keyboard • Receipt printer • Cash dispenser

Test Case	Test Step	Test Data	Expected Result
ATMSD-T1 (1.0)	1. Initiate a connection to **BANK**.	Use TCP/IP over Ethernet.	Connection to **Bank** has been established.
ATMSD-T1 (1.0)	2. Insert a readable card.		System asks for entry of **PIN**.
ATMSD-T1 (1.0)	3. Enter **PIN**.		System displays transaction types.
ATMSD-T1 (1.0)	4. Choose Withdrawal transaction.		System displays account types.
ATMSD-T1 (1.0)	5. Choose checking account.		System displays possible withdrawal amounts.
ATMSD-T1 (1.0)	6. Choose amount that • the ATM currently has and • is not greater than the amount available		Message is shown: *Verifying account balance*.
ATMSD-T1 (1.0)	7. Cancel transaction during verification.		Message is shown: *Withdrawal cancelled*.

Attachments



ATM-sysdesign.png (201 kB)

Links

Bugs detected

detects	~~ATMSD-10~~	Bank proxy doesn't provide available accounts	Done
detects	~~ATMSD-11~~	No transaction options when correct PIN entered	Done

Requirements verified

verifies	ATMSD-3	Connection can be initiated while in idle state	Defined
verifies	ATMSD-4	System asks for PIN when readable card is inserted	Defined
verifies	ATMSD-5	System verifies PIN number	Defined
verifies	ATMSD-6	When correct PIN is entered, transactions menu is shown	Defined
verifies	ATMSD-7	When transactions is selected, Bank sends a list of available accounts	Defined
verifies	ATMSD-8	System keeps track of money on hand	Defined
verifies	ATMSD-9	Transaction can be cancelled at any state	Defined

Comments

Casey Ford added a comment - 24/Jun/25 8:51 AM

However I think I get the idea behind this design decision, but wouldn't it be a better alternative to have the `Bank` simply tell its transaction object to process itself, handing it the whole list of accounts?

Jack Powell added a comment - 24/Jun/25 8:55 AM

Yes, I can see the point, Casey Ford. In this way, the particular process method, which runs for a given transaction type, can be responsible for determining the selection of account object(s). It also allows us to keep related data and behavior closer together by avoiding the removal of the account number from the transaction object.