DI-FCUL **MEI/MI/MSI**

Intrusion (Detection and) Tolerance 2024/2025

Project 1 – BFT-SMaRt Decentralized Token Infrastructure (DTI)

This project aims to build a decentralized token infrastructure to support an NFT market using the BFT-SMaRt replication library [1] (http://bft-smart.github.io/library/).

Project Description

DTI is a deterministic wallet-like service that manages coins based on the UTXO (Unspent Transaction Output) model introduced in Bitcoin and NFT (Non-Fungible Tokens) that can be transacted using the coins. In this model, each object (coin) represents a certain amount of currency a user possesses. This means a transaction consumes a given number of input objects to produce one or two output objects.

The system needs to keep track of two types of tokens and their ownership.[1] These tokens will contain the following information:

- Coin:
  - *ID*: an integer (int) with the coin ID (no two coins can have the same id)
  - *Owner*: an integer (int) with the id of the client owning the coin
  - *Value*: a real number (float) with the value of the coin
- NFT:
  - *ID*: an integer (int) with the NFT id (no two NFTs can have the same id)
  - *Owner*: an integer (int) with the id of the client owning the NFT
  - *Name*: the name (a String) of the NFT
  - *URI*: the URI (a String, typically a URL) of the NFT
  - *Value*: a real number (float) with the value of the NFT

The owner IDs are the numbers used for starting processes on BFT-SMaRt. We typically use 0-3 for the replicas and other positive numbers for clients.

To support these tokens, DTI needs to implement an interface for managing coins and NFTs (in a similar way the BFTMap implements put, get, etc.) Below we list the coin-related operations:

- **MY_COINS()**: get the IDs and values of the coins associated with this user.[2]

[1] This is the state of the service, and can be stored in two objects of the class *java.util.TreeMap* replicated on each server.
[2] You can obtain the id of the sender of an operation at the replica by using the *getSender()* method of the *msgCtx* object received as an argument of *appExecuteOrdered/Unordered*.

- **MINT(*value*)**: require the *value* of the coin to create for the issuer. For that, the issuer needs permission to execute this operation, as defined during system initialization.[3] The operation returns the ID of the newly created coin.

- **SPEND(*coins, receiver, value*)**: requires an array with the IDs of the *coins* that will be used as input, the ID of the user that will receive the transfer (*receiver*), and the *value* to be transferred. If the indicated coins provide enough funds to execute the transaction (*sum(coins) >= value*), the operation consumes the coins and generates two coins, one for the receiver with the value it received and another for the issuer (who invoked the operation) with the remaining value (*sum(coins)* - *value*). The operation returns either the ID of the coin created for the issuer with the remaining value, 0 in case no coin was created (due to no remaining), or -1 if the operation failed.

Besides the coin-related operations, DTI will also support the creation and exchange of NFTs (Non-Fungible Tokens) using the coins. Below, we list the operations related to NFTs:

- **MY_NFTS()**: list the ID, name, URI, and value of the NFTs the issuer possesses.

- **MINT_NFT(*name, uri, value*)**: create an NFT for the issuer with the *name* and *uri* specified. There cannot be two NFTs with the same name. Returns the id of the newly created NFT.

- **SET_NFT_PRICE(*nft, value*)**: change the price of a given NFT owned by the issuer to a new value.

- **SEARCH_NFT(text)**: list the id, name, URI, and value of the NFTs whose name contains the provided text (ignores case).

- **BUY_NFT (*nft, coins*)**: requires an array with the IDs of the *coins* that will be used as input and the ID of the NFT to be bought. If the indicated coins provide enough funds to execute the transaction (*sum(coins) >= nft.value*), the operation changes the NFT owner field to the buyer ID, deletes the provided coins, and generates two coins, one for the NFT owner with the value it received and another for the issuer (who invoked the operation) with the remaining value (*sum(coins)* - *value*). The operation returns either the ID of the coin created for the issuer with the remaining value, 0 in case no coin was created (due to no remain), or -1 if the operation failed.

To implement these operations, the state of the replicas will comprise collections of coins and NFTs. The groups are free to implement it how they see fit if the described operations are supported.

Project Delivering

The delivery of the project should be a zip file containing:

[3] For simplicity, you can assume only client 4 can run MINT() and create money.

1. The code of the project
2. A README file explaining how to build and run the replicas and a test client (the project is expected to run in Linux terminals, not in an IDE), together with potential optimizations or limitations the group implemented.

Deadline

Each group must deliver one copy of the project on the course Moodle on March 27th (13:00hs).

References

[1] A. Bessani, J. Sousa, E. Alchieri. **State Machine Replication for the Masses with BFT-SMaRt.** Int. Conference on Dependable Systems and Networks (DSN 2014).