Unspent Transaction Output (UTXO)-based Chaincode

In the previous tutorial, it was demonstrated that making a simultaneous transfer of blue coins may result to an error. This is due to the two transfers involve the same world state key.

This can be addressed by using a UTXO model when coding the chaincode.

In a UTXO model, an organization will use a different world state key in each transaction to prevent the error that happened in the previous version of the chaincode.

Quick Setup

1. Run the quick setup.

```
chaincode> ./quick-setup.sh blue-coin-utxo blue-coin 1.0
```

   Notice that the `folder name` used is `blue-coin-utxo`.

2. Run the quick extended setup.

```
chaincode> ./quick-extended-setup.sh blue-coin-utxo blue-coin 2.0
```

3. Generate initial coins for org1, org2, and org3.

```
chaincode> docker exec cli0.org1 peer chaincode invoke \
  -o orderer.example.com:7050 \
  -C mychannel -n blue-coin \
  -c '{"function":"generateInitialCoin","Args":["Org1MSP"]}'

chaincode> docker exec cli0.org2 peer chaincode invoke \
  -o orderer.example.com:7050 \
  -C mychannel -n blue-coin \
  -c '{"function":"generateInitialCoin","Args":["Org2MSP"]}'

chaincode> docker exec cli0.org3 peer chaincode invoke \
  -o orderer.example.com:7050 \
  -C mychannel -n blue-coin \
  -c '{"function":"generateInitialCoin","Args":["Org3MSP"]}'
```

4. Confirm that the blue coins are saved by opening a browser and checking the contents of couchdb of peer0.org1.example.com

```
browser> http://localhost:5984/_utils/
```

5. Enroll users for org1 and org3.

```
bc-utxo-client> node enrollAdmin.js 1
bc-utxo-client> node enrollUser.js 1 1
bc-utxo-client> node enrollAdmin.js 3
bc-utxo-client> node enrollUser.js 3 1
```

The `enrollAdmin.js` and `enrollUser.js` programs are the same code found in `client/blue-coin`.

Test the Chaincode

1. Transfer 50 blue coins from org1 to org2.

```
bc-utxo-client #1> node transferCoinWithBlueCoinManager.js 1 1 \
  Org1MSP Org2MSP 1111 50
```

Notice that there is an extra parameter `1111`. This parameter is the `serial no.`. You can place any value here as long as it is unique.

If this is incorporated in a program, the `serial no.` can be automatically generated through a random number generator or by using the timestamp as the `serial no.`.

2. Transfer 150 blue coins from org3 to org2.

```
bc-utxo-client #2> node transferCoinWithBlueCoinManager.js 3 1 \
  Org3MSP Org2MSP 2222 150
```

Simultaneous Transfer

1. Open a second blue coin UTXO client terminal.

2. For the first terminal, type the following command. DO NOT press enter yet.

```
bc-utxo-client #1> node transferCoinWithBlueCoinManager.js 1 1 \
  Org1MSP Org2MSP 3333 5
```

This will transfer blue coins from org1 to org2.

3. For the second terminal, type the following command. DO NOT press enter yet.

```
bc-utxo-client #2> node transferCoinWithBlueCoinManager.js 3 1 \
  Org3MSP Org2MSP 4444 5
```

This will transfer blue coins from org2 to org1.

4. Press enter on both terminals.

   IMPORTANT: Try to make the gap between the two presses as short as possible (e.g., less than 1 second).

   This will make the two function calls be part of the same block.

   Notice that unlike in the original implementation of blue coin, both transactions are successful.

   Even if the two transactions involve transferring blue coins to org2, the two transactions used two different keys related to org2: `Org2MSP-from-Org1MSP-3333` and `Org2MSP-from-Org3MSP-4444`