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
   ```

   Expected Output:

   ```
   Quick setup for chaincode blue-coin is complete.
   ```

   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
   ```

   Expected Output:

   ```
   Quick extended setup for chaincode blue-coin is complete.
   ```

1. 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"]}'
```

Expected Output:

```
... UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke
successful. result: status:200 payload:"
{\"status\":200,\"message\":\"Successfully generated blue
coins\",\"payload\":{\"mspId\":\"Org1MSP\",\"amt\":500,\"spent\":false}}"
```

```
... UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke
successful. result: status:200 payload:"
{\"status\":200,\"message\":\"Successfully generated blue
coins\",\"payload\":{\"mspId\":\"Org2MSP\",\"amt\":500,\"spent\":false}}"
```

```
... UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke
successful. result: status:200 payload:"
{\"status\":200,\"message\":\"Successfully generated blue
coins\",\"payload\":{\"mspId\":\"Org3MSP\",\"amt\":500,\"spent\":false}}"
```

2. 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/
```

Expected Output:

```
{
  "_id": "Org1MSP-initial",
  "_rev": "1-d434a0b5fa01d40a469c852a0d39699d",
  "amt": 500,
  "mspId": "Org1MSP",
  "spent": false,
  "~version": "\u0000CgMBBAA="
}
```

```
{
  "_id": "Org2MSP-initial",
  "_rev": "1-1808e96c45e2b27c4029b8a9c1351b49",
  "amt": 500,
  "mspId": "Org2MSP",
  "spent": false,
  "~version": "\u0000CgMBBQA="
}
```

```
{
  "_id": "Org3MSP-initial",
  "_rev": "1-6991f75d9d7afccd8c2954fad51cbc83",
  "amt": 500,
  "mspId": "Org3MSP",
  "spent": false,
  "~version": "\u0000CgMBBgA="
}
```

3. Install the necessary packages.

```
bc-utxo-client> npm install
```

4. 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
```

Expected Output:

```
Enrolling Administrator admin of org1...
Administrator admin of org1 enrolled successfully.
```

```
Registering User user1... of org1
User user1 of org1 registered successfully.
Enrolling User user1... of org1
User user1 of org1 enrolled successfully.
```

```
Enrolling Administrator admin of org3...
Administrator admin of org3 enrolled successfully.
```

```
Registering User user1... of org3
User user1 of org3 registered successfully.
Enrolling User user1... of org3
User user1 of org3 enrolled successfully.
```

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> node transferCoinWithBlueCoinManager.js 1 1 \
  Org1MSP Org2MSP 1111 50
```

Expected Output:

```
transferCoin invocation successful.
Invoked by user1 of org1.
txId: f28754542b8c17b4bdcc5dfbced1f02a717012c6d8045b7f665d5e3090322532
status: VALID  blockNo: 7
response: {
    "status": 200,
    "message": "Transferred successfully the amount of 50 blue coins from
Org1MSP to Org2MSP",
    "payload": ""
}
```

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> node transferCoinWithBlueCoinManager.js 3 1 \
  Org3MSP Org2MSP 2222 150
```

Expected Output:

```
transferCoin invocation successful.
Invoked by user1 of org3.
txId: 3d11c271061999b2d46d68cb5918153764d283e7a14434d5efd4abb844820ea2
status: VALID  blockNo: 8
response: {
    "status": 200,
    "message": "Transferred successfully the amount of 150 blue coins from
Org3MSP to Org2MSP",
    "payload": ""
}
```

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).

Expected Output:

```
transferCoin invocation successful.
Invoked by user1 of org1.
txId: 7eb1f7edfc7e2f78c59f682acb9fdcb75f776d2ab9e542b5b6c68e6e85ac6f79
status: VALID  blockNo: 9
response: {
    "status": 200,
    "message": "Transferred successfully the amount of 5 blue coins from
Org1MSP to Org2MSP",
    "payload": ""
}
```

```
response: {
    "status": 200,
    "message": "Transferred successfully the amount of 5 blue coins from
Org3MSP to Org2MSP",
    "payload": ""
}
transferCoin invocation successful.
Invoked by user1 of org3.
txId: 092137f34aa3d039c9ea873a680115176c32804b6c7be896dd0600004bb0917d
status: VALID  blockNo: 9
```

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`

5. Check couchdb of peer0.org1.example.com

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

6. Click `mychannel_blue-coin` and look at the list of keys.

Expected Output:

```
Org1MSP-change-from-Org2MSP-1111
Org1MSP-change-from-Org2MSP-3333
Org1MSP-initial
Org2MSP-from-Org1MSP-1111
Org2MSP-from-Org1MSP-3333
Org2MSP-from-Org3MSP-2222
Org2MSP-from-Org3MSP-4444
Org2MSP-initial
Org3MSP-change-from-Org2MSP-2222
```

```
Org3MSP-change-from-Org2MSP-4444
Org3MSP-initial
```

Notice that there are more than eleven (11) keys generated. In the previous tutorial, there are only three (3) keys: `Org1MSP`, `Org2MSP`, and `Org3MSP`.