**TheFundsChain; clearing model**

**Clearing transactions**. DLT Technology: Hyperledger fabric. This note results from personal brainstorming on how to solve the cross-asset transaction puzzle in the cases below.

We attempt to show how the use of an intermediate chaincode, endorsed by a trusted “clearer” could do the trick.

**Use cases for a clearing model**

In this experiment, we do not attempt to provide an oracle for an external payment system (that’s another story), but we seek to coordinate a set of PTF and FUND chaincodes.

1. Investor i exchanges z shares from fund A against c units of currency (i.e. DVP on fund order)
2. Administrator a exchanges, for investor i, x shares from fund A against z shares from fund B (i.e. fund merger)
3. Administrator a transfer z shares from fund A from investor I to investor j (transfer of ownership)
4. Issuer s pays d units of currency to investor i (franco transaction) (i.e. dividend)
5. Issuer s delivers z shares of fund A to investor I, without compensation (franco transaction) (i.e. attribution of shares)

These use case show that such a mechanism is required, even in the case of totally externalized cash payments: at least use case (2), (3) and (5) must be managed on-the-chain.

Pi is the portfolio chaincode for investor i. A, B are fund chaincodes. Z is a currency. (4) and (5) are actually the same, since currency is just a particular asset.

Let us summarize the initial state of each of these chaincodes:

Pi {(xi,A) ;(yi,B) ;(ci,Z)} Pj {(xj,A) ;(yj,B) ;(cj,Z)}

A {(xi,i) ; (xj,j) ; (cA,Z)} B { (yi,i) ; (yj,j) ; (cB,Z)}

1. **DVP (fund order)**

Begin transaction dvp

Pi <= Pi{+zi, A) ; (-di,Z)}

A<=A((+zi,i) ; (+di,Z)}

Commit dvp

Checks:

ci – di > =0

Final state

Pi {(xi+zi,A) ;(yi,B) ;(ci-di,Z)} Pj {(xj,A) ;(yj,B) ;(cj,Z)}

A {(xi+zi,i) ; (xj,j) ; (cA+di,Z)} B { (yi,i) ; (yj,j) ; (cB,Z)}

1. **Asset transfer (fund merger)**

Begin transaction transfer – Possible cash adjustments not represented

Pi <= Pi{-xi, A) ; (+zi,B)}

A<=A((-xi,i)}

B<=B((+zi,i)}

Commit transfer

Checks:

xi-xi>=0

yi+zi>=0

Final state

Pi {(0,A) ;(yi+zi,B) } Pj {(xj,A) ;(yj,B) ;(cj,Z)}

A {(0,i) ; (xj,j) ; (cA,Z)} B { (yi+zi,i) ; (yj,j) ; (cB,Z)}

1. **Transfer of ownership**

Begin transaction ownership

Pi <= Pi{-zi, A) }

Pj <= Pj{+zi, A) }

A<=A((-zi,i) ; (+zi,j)}

Commit ownership

Checks:

xi-zi >=0

xj+zi>=0

Final state

Pi {(xi-zi,A) ;(yi,B) ;(ci,Z)} Pj {(xj+zi,A) ;(yj,B) ;(cj,Z)}

A {(xi-zi,i) ; (xj+zi,j) ; (cA,Z)} B { (yi,i) ; (yj,j) ; (cB,Z)}

1. **Payment franco (dividend)**

Begin transaction dividend

Pi <= Pi{(+di,Z)}

A<=A( (-di,Z)}

Commit dividend

Checks:

ci+di>=0

cA – di > =0

Final state

Pi {(xi,A) ;(yi,B) ;(ci+di,Z)} Pj {(xj,A) ;(yj,B) ;(cj,Z)}

A {(xi,i) ; (xj,j) ; (cA-di,Z)} B { (yi,i) ; (yj,j) ; (cB,Z)}

1. **Asset transfer franco (share attribution)**

Begin transaction franco

Pi <= Pi{(+zi, A)}

A<=A((+zi,i) }

Commit franco

Checks:

xi+zi > =0

Final state

Pi {(xi+zi,A) ;(yi,B) ;(ci,Z)} Pj {(xj,A) ;(yj,B) ;(cj,Z)}

A { (xi+zi,i) ; (xj,j) ; (cA,Z)} B { (yi,i) ; (yj,j) ; (cB,Z)}

**Clearing chaincode**

Illustration on DVP: post instructions, match instructions, clear instructions



**Processing faults:**

* Clearer is assumed to never fail for a long time: failures are transitory
* We assume that actions are taken when tx are committed, not merely when they are endorsed (see improvements below to see what’s happening if we relax this hypothesis)
* Fault #1: phase 1 instructions are not all committed: 1 passed but not 2: transaction is stalled in a pending settlement status. After a timeout, PTF and FUND assume the settlement failed
* Fault #2: phase 2 query cannot complete (either PTF or FUND are down): clearer reiterates until timeout then yields a fail
* Fault #3: phase 3 (clearing status propagation) fails
  + 1 fails: clearer reiterates until timeout, after which status becomes failed
  + 1 successful, but status propagated to one of either PTF, FUND but not both: clearer reiterates until timeout then yields a SUCCESS. When PTF or FUND eventually recovers, it will ask clearer for a status

Recovery upon a fault:

* Fault #1: PTF or FUND has decided a fail unilaterally (timeout on pending status):
  + Party inquire clearer to check if an explicit SUCCESS has been decided
  + If not, party invokes clearer with its part of the instruction sets to force any matching transaction into fail (clearer may by this time also have concluded by himself to a fail)
  + if clearer meanwhile inquires for a matching status, answer is failed
* Faults #2, #3: clearer has decided a fail unilaterally:
  + Clearer keep attempting to propagate the fail status until another (longer) timeout after which any recovering party (PTF or FUND) should naturally conclude with a fail unilaterally

**Byzantine faults:**

* Clearer is trusted and cannot initiate Byzantine faults
* BZ-Fault #1: (rogue SDK) Phase 1: 2 sent without one of (1): phase 2 concludes an unmatched pair and yields a fail
* BZ-Fault #2: (rogue chaincode) Phase 2: PTF or FUND responds to Clearer’s inquiry with false claims (e.g. replayed past tx). Unique token doesn’t match and clearer concludes to a fail.
* BZ-Fault #3: (rogue chaincode) Phase 3: (either PTF or FUND) refuses to validate status propagation. Clearer may have decided a SUCCESS on its side. The success status may have been successfully propagated to the other party. Status mismatch between P, F and C.
* BZ-Fault #4: (rogue chaincode) Phase 3: (either PTF or FUND) validates status propagation but updates its state with a different value. Status mismatch between P, F and C.
* “rogue chaincode” issues may be created willingly or by a bug…
* Status mismatch handling:
  + We assume the chaincode is buggy, but that it doesn’t attempt to forge a fake state (chaincodes are trusted containers after all)
  + We may accept the chaincode reflects a stale state, which may be self-checked to resynchronize (see below)
  + State staleness is detected as follows:
    - The status update sent by the clearer is actually signed with a timestamp. The status update embeds the counterpart account/registry line (kind of “double accounting certificate” issued by the clearer)
    - Any changed PTF or FUND state variable subject to clearing (e.g. an inventory line) is accompanied with the latest change status signed by the clearer
    - It is easy to check that an inventory is invalid (state change timestamp doesn’t match clearer timestamp, and for every change, a valid counterpart entry must be found – e.g. double accounting). For instance, this is what will be performed by payment processing SDKs. Parties querying the state should be able to check. In case of doubt the ledger may be queried
    - If a chaincode detects a stale state (exhibits a bug causing a rogue behavior) the chaincode must resync by querying clearer.

General idea:

* Our clearing protocol relies on the idea that the peer initiating the transaction has an incentive to properly instruct its order (phase 1). If he doesn’t, orders remain unsettled.

In case of doubt, we may replace the investor by the investor custodian

Consequences of a rogue SDK posting many unclearable orders to a fund: (i) The fund manager sees unsettled (yet valid) orders and moves its assets accordingly. (ii) Therefore, the rogue SDK fooled the fund manager. Example: to force an asset manager to underperform, flood him with redemption orders, etc. **How do we prevent this?**

**Clearing sequence for funds is actually initiated by the transfer agent, not the investor**

**If frequent fails occur with a given PTF (or FUND for redemptions…), the transfer agent may decide to switch to prepaid mode by requesting orders to be backed by earmarked cash**

* For performance reasons, the clearing chaincode may deployed by every peer of a FUND or PTF chaincode: essentially, this is a public ledger – only peers with a clearer role endorse transactions.
* Contents are encrypted so that only the endorsing peers of the chain codes involved between a BEGIN and a COMMIT may read the payload of a transaction (beware, this makes it more than bilateral!)
  + With functional encryption, one may make the clearing check a zero-knowledge proof
* The clearer has the responsibility of final state integrity: since the Hyperledger system does not guarantee such a cross-chaincode integrity, the final clearing phase completion status is persisted by the clearer (the state of the clearing chaincode).
* Any transaction subject to clearing that is validated by a fund (FUND) or a portfolio (PTF) returns to the submitting client (and possibly posting an event to other endorsing peers):
  + A globally unique clearing ID deliver by the Clearing chaincode
  + An encrypted transaction summary (like the begin/end and Pi, A, B transactions models above, with initial state in the relevant assets)
  + Encrypted data may be read by:
    - Clearer party
    - Any chaincode involved in the transaction
  + Encrypted data is signed by the Fund
  + The client that posted the transaction must reuse the same clearing ID to inform all subsequent transactions in the same BEGIN/COMMIT boundaries
  + If it doesn’t (with a limited time frame), the transaction is not cleared.
* Further checking: optionally, the concerned PTF and FUND chaincodes may query the other involved objects for cross-checking…
* Clearing may be net the proceeds of several transactions before checking

**Protocol improvements**

1. More than 2 parties

Our mechanism is essentially a 2-phase commit transaction manager living on the DLT.

It may be used as a notary to validate transactions involving more than 2 chaincodes.

For instance, one might decide that the custodian is not peer for the FUND, but has a FUNDPTF chaincode instead (manage assets only, and in particular, cash). Clearing would thus involve 3 chaincodes.

Clearing could be applied to other transactions, like payments on a contract (e.g. trailer fees).

1. Netting

Settlement instructions may be batched, e.g. all transactions between PTF pi and FUND A of the day may be processed as a single clearing instruction (invoking the clearer with a batch of “globalTransactions” objects)

1. Parallelizing settlement

All parties may clear in parallel. There may be several clearers. There may also be several clearinghouse chaincodes (e.g. several large trustable institutions may deploy their own clearinghouse)

1. Anticipate outcome by using only endorsements

We would prefer not… but performances rule! What would happen?

This makes a number of new faults to be analyzed.

1. Anchoring

Parallel processing with different parties might cause double spending.

Example:

Ptfi purchases some FUND A

Ptfi sells some FUND B

Ptfi purchases some FUND C

(orders are instructed then confirmed… the price is revealed and orders are executed…)

Now settlement instructions are posted (by transfer agent) to parties and clearer.

Transfer agent for FUND A send debit instruction to PTFi

Transfer agent for FUND B send transfer instruction to PTFi

Transfer agent for FUND C send debit instruction to PTFi

(orderers ensure that all these transactions are sent orderly to the PTFi chaincode)

If all transactions are passed in this order, serializability ensures that no double spending occurs.

Anchoring should not be necessary provided PTF and FUND take into accout “pending settlement” transactions in their double spending check. Failed settlement transactions may be safely excluded from double spending checks of further transaction.

1. Multiple clearing SDK (e.g. everybody may act as a clearer)

This relaxes the assumption that the clearer cannot initiate byzantine faults.

1. Cross channel clearing

To be considered to clear cross-jurisdiction transactions (e.g. a PTF belonging to Lux channel interacts with a fund belonging to FR channel).

**Confidentiality**

The clearing house chaincode is a shared ledger (more or less like the “business network” model of the Fabric Composer).

Transaction details are required in order to properly clear the transaction.

However, all peers should not be able to retrieve transaction details.

Ideally, the clearer himself does not even know the transaction details.

Background describing what to be cleared:

Begin

i invokes CC.A with invoke set i{A} and result set r{A}, validated by X

i invokes CC.B with invoke set i{A} result set r{B}, validated by Y

…

Commit

The result set includes a transaction ID local to the CC. In other words the global transaction is described by:

{ globalTransaction:

initiator: "i", signature: "sign(i,whole message}"

invokes: [

{

chaincode: {

channel: "thefundschain.fr",

cc: "CC.A",

endorsers: ["a","b","c"],

validatedBy: ["X"]

},

transaction: {

invokeContent: "i{A}",

responseContent: "r{A}",

respondantSignature: "sign(X,r{A}"

}

},

{

chaincode: {

channel: "thefundschain.uk",

cc: "CC.B",

endorsers: ["u","v"],

validatedBy: ["Y"]

},

transaction: {

invokeContent: "i{B}",

responseContent: "r{B}",

respondantSignature: "sign(Y,r{B}"

}

}

]

}

Each invoke part may be produced independently by a CC when queried by a clearer.

To do so: transactions pending settlement will be indexed by their hash of invoke content by each chaincode.

This leads to a simplified global transaction description provided by i to the clearer:

{ globalHashedTransaction:

initiator: "i", signature: "sign(i,whole message}"

invokes: [

{

chaincode: {

channel: "thefundschain.fr",

cc: "CC.A",

endorsers: ["a","b","c"],

validatedBy: ["X"]

},

transaction: {

invokeContent: "hash(i{A})",

responseContent: "hash(r{A})",

respondantSignature: "sign(X,r{A}"

}

},

{

chaincode: {

channel: "thefundschain.uk",

cc: "CC.B",

endorsers: ["u","v"],

validatedBy: ["Y"]

},

transaction: {

invokeContent: "hash(i{B})",

responseContent: "hash(r{B})",

respondantSignature: "sign(Y,r{B}"

}

}

]

}

When querying chaincodes CC.A and CC.B, the clearer asks for hash(r{B}) as answer to:

checkPendingSettlementStatus(hash(i{B})

Clearer then verifies that both CC.A and CC.B did provide a correct hash(r{.}) then proceed to updating the status.

**Confidentiality improvement**

The clearinghouse is a public ledger. The track (I, CC.A, CC.B) constitutes too much of an information already.

* instead of i (identity of initiator i), we may use a T-cert
* chaincode object data may been encrypted with the PKI of clearer, so other peers cannot read it

{ globalHashedTransaction:

initiator: "tcert(i)", signature: "sign(i,whole message}"

invokes: [

{

chaincode: {

recipient: {clearer}

encryptedData: {

channel: "thefundschain.fr",

cc: "CC.A",

endorsers: ["u","v"],

validatedBy: ["Y"]

}

},

transaction: {

invokeContent: "hash(i{A})",

responseContent: "hash(r{A})",

respondantSignature: "sign(X,r{A}"

}

},

{

chaincode: {

recipient: {clearer}

encryptedData: {

channel: "thefundschain.uk",

cc: "CC.B",

endorsers: ["a","b","c"],

validatedBy: ["X"]

}

},

transaction: {

invokeContent: "hash(i{B})",

responseContent: "hash(r{B})",

respondantSignature: "sign(Y,r{B}"

}

}

]

}