Notes on Hyperledger (25/11/16 – 16/11/16)

Review: 30/06/17

**Hyperledger basics**

1. We have the possibility to setup an *orthogonal* set of endorsers (database peers) and consenters (consensus participants). With V1, consenters are called “orderers” and they participate a (BFT) ordering service, which is a consensus delivering serialized (ordered) transactions.

Endorsers are the only participant which execute the chaincode. Other peers (“committers”) merely replicate the status obtained by ordering transactions.

“Consensus” must not be thought as a vote to agree or disagree on a chaincode execution: consensus is getting the proper order of transactions. Only endorsers actually check the execution. The “consensus of endorsers” is modelled by an endorsement policy.

1. How this could benefit our model?
   1. Chaincode replication is limited to peer nodes
   2. Transaction validation is split in two steps: a functional step (endorsement) and a BFT step (consensus), which is essentially technical
2. Strong limitation (for the moment): a chaincode is isolated
   1. Cross-chain codes operations are not yet supported (it is possible to query another chaincode, though)

This is really annoying…

* 1. In essence, atomic pipelined transactions involving several chaincodes is not supported yet.

It is a post-V1 feature, but there is no announcement on the roadmap.

* 1. This significantly challenges our {object directory}/{directory} design pattern, and more generally our design choices balancing privacy, flexibility, performances and consistency.

1. We have a possibility of a persistent state (e.g. balance of accounts) vs track record (i.e. ledger)
2. There is no sharding beyond the level of peers: all peers duplicate the whole ledger for any given chaincode they are in charge of (note: channels may be used to partition ledgers between unrelated participants). This advocates for some designs in which a “chaincode” is much more granular than “a class of business objects” (see options below). Granularity must neatly balance data privacy constraints, flexibility and data efficiency.

Note: in V1, data sharding may be developed using CouchDB. The granularity is finer than the chaincode (each chaincode gets its own autonomous CouchDB instance, which may be split in shards.

Basically, we liken a chaincode to autonomous entity in charge of the full life-cycle of a business object.

**TheFundsChain: building blocks**

We essentially have 5 classes of chaincode:

* Fund
  + An autonomous smart contract taking care of all aspects of a fund’s life cycle
  + In particular, it contains the investors’ registry, and could as well handle an inventory of assets, in effect implementing a full-fledged fund’s balance sheet and ledger
* Funds directory
  + A list of Fund objects
  + Directories come in many variations as any participant may find useful to list its own perimeter of interest in a directory
  + A directory of particular interest is the regulator’s public directory of funds under its jurisdiction
* Investors Store / KYC Store
  + A private structure describing all the investors of any given distributor
  + A zero-knowledge proof query operation to allow external participants to check KYC/suitability
* Identity directory
  + A common directory of MSP’s connected to the platform, with their role under various jurisdictions
* Portfolio
  + A structure owned by an investor custodian for an investor to keep the books on its investments. Most robo-advising features start from here.

A 6th object class may also be defined: distribution agreements. Other agreements may well be stored (e.g. custody agreement, …) but the system is not able to *implement* them (as it should manage the assets of a fund to do so). However, a funds master repository is a good place to store such contracts and manage the administrative workflow.

We count NAV’s as a subpart of the funds master model (but possibly technically set in a different chaincode).

Maintaining the Fund registry requires a read-only access to the current validated state of both Funds Master and Investors Store. Investors Store & Funds Master are orthogonal (i.e. KYC check are performed independently from transactions).

The semantics of transactions stored on the ledger vary depending on the object:

* Funds: history of changes
  + Special case worth mentioning: NAVs – time series of NAVs
* Fund Registry: record of past transactions (i.e. financial transactions semantics)
* Investors: history of changes (i.e. master data semantics)

Involved parties heavily depends on the Funds master configuration (e.g. registration country).

Assuming the central roles of issuer, distributer, investor, accountant and TA are peers, we must manage several scenarios regarding custodians:

1. Funds custodian is not a member: messages / oracle through TA
2. One of the investor’s custodian (cash or securities) is not a member: messages / oracle through TA
3. If orders are settled through a centrally managed settlement system, the latter may provide required messages / oracles (acts as a TA: this means that the pure TA in CSD mode is disrupted – “centralisateur” in French model). More generally, TA’s act as trusted agents on behalf of non-member parties (distributors, custodians, other CSD…).

We assume all investors are identified. Legally this means that shares are issued:

* Either in registered form (French: nominatif pur): investor’s sec. custodian is the issuer
* Either in registered form (bis) (French: nominatif administré): investor’s sec. custodian remains the issuer and any valid financial institution (e.g. a bank) is in the charge of the administration of the account (in effect: the account maintained by the issuer will mirror the account managed by the financial institution which is the “master” account to the extent that the investor communicates his instructions/orders to the financial institution.
* Either in bearer’s form: in this case, more precisely, it is an Identified Bearer’s Form (TPI: Titres au Porteur Identifiables), meaning that we assume that every single investor is properly identifiable to the platform (not necessarily to all participants). This constraint may be relaxed, provided all compliance checks may still be carried on

Funds master

* We assume that most changes to Funds Master are carried on by the Issuer
* Limited changes are carried on by the Regulator (e.g. agreement status)
* Consent / Changes may be carried on by other actors. Typically:
  + Custodian: a custody account for the funds (cash & securities)
  + TA: a registry analytical structure
  + Accountant: a fund management fees structure

NAV’s are (logically / perhaps not physically) part of the funds master model.

Funds master is essentially composed of public data.

Some data may be private / restricted. Examples:

* Intermediary NAV’s (not published) (fund accountant vs issuer)
* Restricted NAV’s (e.g. FCPE)
* Reserved / Dedicated funds: cross-references to issuer’s CRM
* Mandates & managed accounts
* Custody accounts (custodian / TA / issuer)

A significant part of funds master data is made of raw documents (PDF’s…), which are validated and published in several languages.

On the other hand, Investors store is essentially private.

It may be upgraded to restricted (partnership between issuer and distributor).

**Chaincode to chaincode queries**

There are 2 options:

* Chaincode directly queries the state of another chaincode at the same peer
* Chaincode uses GO SDK to issue a remote query on the network (see <https://github.com/hyperledger/fabric-sdk-go/blob/master/>)

Unfortunately, the switch between these two options is not seamless: the developer must know beforehand.

**Modelling options on the table (in no specific order):**

Option #1 considered: each fund is an almost self-sustained chaincode (missing: references to investors and regulator’s agreement). There should be an additional service to search funds.

Option #2 considered: each business object class is a chaincode with many occurrences (funds may possibly be grouped by issuer), similar to relational tables. A chaincode for funds (with NAVs?), one for investors and one for investor registry & orders.

Option #1 is perhaps easier to implement as the relevant partners (defined by the funds master data) are defined with the chaincode. Question: what if the fund changes a partner (example: fund is sold to another fund manager, the accountant changes, or more casually: new Prime TA partners or Distributors come in).

In other words, and in order to cope with business, the list of partners should not be static. It seems that this is not possible, since parties and roles are defined at chaincode creation time.

V1 update: it is now possible to dynamically deploy new peers. Since endorsement policy may be expressed by organization, new peers from an “endorsing organization” may join. If the endorsing responsibilities change, we may now deploy a new version of the chaincode…

~~I see two possibilities:~~

1. ~~Such (major, infrequent) changes result in a new chaincode to be forked from the original one (then: how about keeping a track record? It just looks like a “chain reboot’” of sort).~~
2. ~~Chaincode setup defines very broad policies regarding members, and the actual consensus (customized from PBFT) picks the correct parties dynamically. Sounds promising but how costly would be such a development (i.e vs POC constraints).~~

Option #2 exhibits a more classical data design. Someday perhaps, it will be possible to design things in such a natural way.

If it proves technically possible, then such a design is probably closer to a final design than the previous one (how to deal practically with 100 000s of such chaincodes?). After all, the only documented constraint on endorsement policies is that they must be deterministic and local to the peer (i.e. not calling other peers…). Implementation issues are probably untractable: scalability issues when many transactions occurs on different funds of this unique chaincode, replication of the whole data set across all possible peers… etc.

[note: in my own views, there is no real difference between endorsement and consensus, I don’t really see the point in separating these 2 roles. Update: now I got the point about limiting the number of concurrent chaincode executions (e.g. contrary to Ethereum design) and now I think I get the point of endorsement in light of the Paxos consensus algorithm]. In the end in the config I’ve in mind, there is no real difference.

The actors that count should be endorsing peers anyhow. Now that the documentation clearly states that other peers are just “committers” and the ones running PBFT are just “orderers”, things are clearer (at least for me).

We shall develop our solution along the lines of Option#1. This means:

* Deploy FundsDirectory as a single chaincode
  + FundsDirectory is the chaincode in charge of the REGISTRATION process (issuer vs regulator)
  + FundsDirectory holds basic data about funds
  + FundsDirectory publishes the chaincodeID of each fund
  + Every peer replicates the FundsDirectory
* Deploy one chaincode for each fund
  + Such chaincodes may be specialized depending on the type of fund
  + Such chaincodes follow templates, designed to implement the specifics of each type of fund
  + Only the peers concerned replicate a fund chaincode
  + This chaincode keeps the whole set of informations for ONE FUND

Interesting extension: a fund manager may deploy a private FundsDirectory for its mandates and deploy mandate chaincodes without regulator authorization.

Question: is it relevant to deploy one instance of FundDirectory by country/regulatory zone? … And then have a “meta-FundsDirectory”?

**Thought of the day: secure execution of a chain code**

The chaincode itself may well be signed and certified, this doesn’t mean its execution is safe. Consider for instance using shared external resources (e.g. multi-lingual error messages…). If such resources are themselves not consensually validated, they constitute a security breach. Further, how do we use external libraries and tools?

Answer: the signed component is actually the docker container,

All resources are pre-compiled and statically linked: the compiled object is checksumed: basically, this is the doing of docker. “Chaincode sealing” is performed by sealing a Docker Container. A chaincode is thus instantiated thanks to deploying a docker image. Portability is managed by deploying virtual machines with an OS (e.g. Linux Ubuntu 64 bits).

Moving further on option #2:

* A fund is described by a JSON document which is stored with a single key in the chaincode state structure (V1 with CouchDB option enabled: a document in CouchDB).
* HL does:
  + Keep track of changes [perhaps one should figure out a DELTA before recording transactions]
  + Validates changes
  + Store the successive versions for each key
* A few caveats:
  + it is not possible to “reboot the chain” and prune oldest changes.

§

Let us recall the main processes interacting with a fund in the fund master (very much French-biased, but this is a good start):

1. Filing: an issuer INITIALIZES a description of a fund
2. Registration: this application is considered by one domestic or foreign regulator for REGISTRATION
3. Agreement: the regulator delivers an AGREEMENT for REGISTRATION and domestic distribution
4. Agreement notification (b): other agreements may be obtained from the same regulator or foreign regulators to distribute the fund in other countries. A special “passport” (UCITS) is available to allow EU distribution in one single agreement

Note: this seems mandatory but for information (the regulator cannot refuse it). Local regulator acts as single point of contact with other regulators to be informed. Alternate wording: Notification: To take advantage of the passport, Fund manager has to transmit notification letter + documents to its regulator who will transmit information to host authorities where the fund is intended to be distributed.

1. Infrastructure actors: the fund signs a CUSTODY AGREEMENT (or a series of specialized subparts of such a document) with authorized servicing companies. This agreement defines:
   1. Who maintains the record of the assets of the fund (“fund custodian”)
   2. Who is in charge of collecting and executing subscriptions and redemptions (“TA”)
   3. Who maintains the registry of issued shares (“registrar”)
   4. Who keeps the books for the fund, and particularly, calculates Net Asset Values (“fund accountant”)

Note: possibility for the custodian to delegate some of the services (depending on jurisdiction / FR: CoMoFi L214-10-6)

1. Identification: the fund MAY issue a request to a domestic CSD to get a newly registered security code (e.g. ISIN). A similar mechanism is used to acquire a LEI (Legal Entity Identifier)
2. Circulation: the fund MAY apply to Euroclear to get access to its settlement system
3. The fund MAY sign distribution agreements with distributors, to delegates, wholly or partially, its sales activity
4. Delegation: the fund MAY delegate, wholly or partially, its fund management activity. Therefore, the “issuer” (or “promoter”) is not necessarily the fund management company.
5. Corporate Actions
   1. Merger
   2. NAV rebasing (look for a more adapted term)
   3. Closure (look for a more adapted term)
   4. Change parties: issuer, …
   5. Dividend
   6. …

We see with this long list that the ISSUER is not the only stakeholder, even though it INITIATES most changes.

Use-cases: an issuer may easily retrieve all its funds / an infrastructure actor may easily retrieve all funds under its care / a distributor may easily retrieve all funds.

If we go for a model with 1 fund = 1 chaincode, we must find out a way to query and index them globally for search…