Interoperability Project for Enterprise Blockchains: YUI and Cross Framework

YUI Interoperability Project

Connecting public and private blockchains as well as traditional systems

Public Blockchains

Private Blockchains

Traditional Systems

Datachain
1. Quick Introduction about Datachain, Inc.
2. Quick Introduction of YUI and Cross Framework
3. Details of Cross Framework
What is Datachain?

We provide interoperability solutions for networks of blockchain

- Developing Cross Framework for interoperability among various blockchains
- Contributing YUI project to Hyperledger Lab
- Working on various PoC for Enterprise use
1. Quick Introduction about Datachain, Inc.

2. Quick Introduction of YUI and Cross Framework

3. Details of Cross Framework
We are developing cross-chains solutions in trustless. Part of them are contributed into Hyperledger Lab project as **YUI**.

(https://github.com/hyperledger-labs/yui-docs)

<table>
<thead>
<tr>
<th><strong>YUI</strong></th>
<th>A Project to Achieve Interoperability Between Multiple Heterogeneous Ledgers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Module</strong></td>
<td>Modules to support the development of cross-chain contracts</td>
</tr>
<tr>
<td><strong>IBC Module</strong></td>
<td>Client (on-chain) modules to support various ledgers in align with the design principles</td>
</tr>
<tr>
<td><strong>Relayer</strong></td>
<td>a middleware to relay packets between sets of various ledgers.</td>
</tr>
<tr>
<td><strong>Development Tools</strong></td>
<td>Explorer etc.</td>
</tr>
</tbody>
</table>

Available in YUI: Fabric-IBC, Besu-IBC, Corda-IBC.
We are developing cross-chains solutions in trustless. Then part of them are contributed into Hyperledger Lab project as YUI. ([https://github.com/hyperledger-labs/yui-docs](https://github.com/hyperledger-labs/yui-docs))
Interoperability Overview

**Ledger X** e.g. Hyperledger Fabric

**App Module**
Middleware for cross-chain transactions

**IBC Module**
Compliant to Cosmos ICS
Ledger-dependent

---

**Relayer**
Off-chain process to relay packets bet. ledgers

**Ledger Y** e.g. Hyperledger Besu

---

**App Module**

---

**IBC Module**

---

Cannot communicate directly without interoperability solution

Unlike trusted third party model, communication is verified on each chain
1. Quick Introduction about Datachain, Inc.

2. Quick Introduction of YUI and Cross Framework

3. Details of Cross Framework
IBC Basics: Relay scheme

IBC does not rely on trusted third party, it is categorized as Relay scheme

Trusted Third Party
Rely on specific authority

Verification of transaction is done in central authority

Relay
On-chain verification on each chain

Safety of the system is guaranteed by on-chain verification

(conf.) HTLC
Hashed timelock of tokens

Bound to simple atomic swap of token
IBC is similar to TCP/IP, IBC has abstractions like Channel, Connection and Handshake is required to realize reliable communication.
Light Client to verify the packets from Blockchain Y.

Relay the Packet. No trust is needed on Relayer.

Blockchain X
- Application
- Cross Framework
- IBC Channel
- IBC Connection
- Client (for Y)

Relayer

Blockchain Y
- Application
- Cross Framework
- IBC Channel
- IBC Connection
- Client (for X)
IBC Basics: High Level Architecture 03

Light Client to verify the packets from Blockchain Y.
Relay the Packet. No trust is needed on Relayer.

Blockchain X
Application
Cross Framework
IBC Channel
IBC Connection
Client (for Y)

Relayer

Handshake

Blockchain Y
Application
Cross Framework
IBC Channel
IBC Connection
Client (for X)

Light client protocol

Copyright © 2022 Datachain, Inc. All Rights Reserved.
IBM Basics: High Level Architecture 04

Light Client to verify the packets from Blockchain Y.
Relay the Packet. No trust is needed on Relayer.

Handshake + Packet transfer
Handshake

Light client protocol

Copyright © 2022 Datachain, Inc. All Rights Reserved.
Today's Main Topic

CROSS FRAMEWORK
Cross Framework enables Cross-chain transactions in atomic on top of IBC communication. The framework provides the state store with a locking mechanism and the coordinator of contract execution.

Cross framework provides **transaction coordination** for multiple blockchain, thus enables **atomic commit/abort** among them

- example: transaction B on ledger Y is invoked if transaction A on ledger X is successful

To make prepare/commit phases, a wrapper for state store with a **locking mechanism** is introduced

- example: lock the state in ledger X before invoking contract in ledger Y
Cross Framework: Transaction Initiator

- Initialize cross-chain transactions and call Authenticator for authentication
- After authentication done, TxRunner is called to call actual contract functions
message `InitiateTx` {
    string chain_id = 1;
    uint64 nonce = 2;
    CommitProtocol commit_protocol = 3;
    repeated initiator.ContractTransaction contract_transactions = 4;
    repeated auth.Account signers = 5;
    ibc.core.client.v1.Height timeout_height = 6;
    uint64 timeout_timestamp = 7;
}

message `ContractTransaction` {
    google.protobuf.Any cross_chain_channel = 1;
    repeated cross.core.auth.Account signers = 2;
    bytes call_info = 3;
    cross.core.tx.ReturnValue return_value = 4;
    repeated Link links = 5;
}
Cross Framework: Authenticator

- Provides a transaction authentication and manages its status.
- There are several authentication methods, SignTx, IBCSignTx, and ExtSignTx.
- Authentication of a transaction is performed by the accounts specified in signers of each contract transaction in contract_transactions. The execution is blocked until the authentication is completed.
- Provides a transaction authentication and manages its status.
- There are several authentication methods, `SignTx`, `IBCSignTx`, and `ExtSignTx`.
- Authentication of a transaction is performed by the accounts specified in signers of each contract transaction in `contract_transactions`. The execution is blocked until the authentication is completed.

```protobuf
// For `SignTx`
message MsgSignTx {
  bytes txID = 1
  repeated bytes signers = 2
  ibc.core.client.v1.Height timeout_height = 3
  uint64 timeout_timestamp = 4
}
```
Cross Framework: Tx Runner

- Executes transactions after Authenticator has successfully authenticated
- As Atomic commit protocol, Two-phase commit / Simple Commit is supported
Cross Framework: Contract Manager

- Manages Contract Module that implements a contract and State Store. It also provides API to commit and abort changes. These APIs are called by Tx Runner corresponding to the status of the commit protocol flow.
• Defines a smart contract. It is implementation for required functionalities by developer.
Cross Framework: flow from InitiateTx to SignTx

Initiator Coordinator / Participant

Chain A

- Transaction Initiator
- Authenticator
- Transaction Runner
- Contract Manager
- Contract Module

Submit InitiateTx

Chain B

- Transaction Runner
- Contract Manager
- Contract Module

Authenticator

Submit IBCSignTx

Client A

Client B

Copyright © 2022 Datachain, Inc. All Rights Reserved.
Atomic commit protocol
Atomic commit protocol

1. Two-phase commit protocol
2. Simple commit protocol
Cross Framework: 2 Phase Commit Protocol

Example of 2 Phase Commit Protocol

Ledger
- Coordinator Chain
  - Role: Coordinator, Participant, Relayer
  - Transaction: e.g. send token and change owner of ticket

Prepare Phase
- Create InitiateTx and Submit
- Initialize Tx State
- Create Packet
- Detect Packet and Relay
- Confirm Status
- Ask Commit after all participants returns OK by Ack Packet

Commit Phase
- PacketPrepare
- Verify Packet
- Prepare Commit
- Detect Packet and Relay
- Execute Commit
- Create Packet
- PacketCommit
- Detect Packet and Relay
- PacketCommitAcknowledgement

Phase
- Create Packet
- PacketPrepareAcknowledgement
- PacketCommitAcknowledgement
- Tx Done
Cross Framework: 2 Phase Commit Protocol

1. Initiate Step

Ledger:

- Coordinator Chain
- Chain A
- Chain B

<table>
<thead>
<tr>
<th>Role</th>
<th>Coordinator</th>
<th>Participant</th>
<th>Coordinator</th>
<th>Participant</th>
<th>Relayer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiate Tx and Submit</td>
<td>Create InitiateTx and Submit</td>
<td>Create Packet</td>
<td>Initialize Tx State</td>
<td>Create Packet</td>
<td>Create Packet</td>
</tr>
<tr>
<td>Prepare Phase</td>
<td>Detect Packet and Relay</td>
<td>Detect Packet and Relay</td>
<td>Detect Packet and Relay</td>
<td>Detect Packet and Relay</td>
<td>Detect Packet and Relay</td>
</tr>
<tr>
<td>Commit Phase</td>
<td>Store changed state, get locked to resources by executing contract</td>
<td>Store changed state, get locked to resources by executing contract</td>
<td>Store changed state, get locked to resources by executing contract</td>
<td>Store changed state, get locked to resources by executing contract</td>
<td>Store changed state, get locked to resources by executing contract</td>
</tr>
<tr>
<td>Ask Commit after all participants returns OK by Ack Packet</td>
<td>Ask Commit after all participants returns OK by Ack Packet</td>
<td>Ask Commit after all participants returns OK by Ack Packet</td>
<td>Ask Commit after all participants returns OK by Ack Packet</td>
<td>Ask Commit after all participants returns OK by Ack Packet</td>
<td>Ask Commit after all participants returns OK by Ack Packet</td>
</tr>
</tbody>
</table>

Transaction e.g. send token and change owner of ticket

Copyright © 2022 Datachain, Inc. All Rights Reserved.
Cross Framework: 2 Phase Commit Protocol

2. Prepare Step

Ledger

<table>
<thead>
<tr>
<th>Role</th>
<th>Coordinator Chain</th>
<th>Chain A</th>
<th>Chain B</th>
<th>Relayer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coordinator</td>
<td>Participant</td>
<td>Participant</td>
<td>Participant</td>
</tr>
<tr>
<td></td>
<td>Participant</td>
<td></td>
<td></td>
<td>Relayer</td>
</tr>
</tbody>
</table>

**Prepare Phase**
- Create InitiateTx and Submit
- Initialize Tx State
- Create Packet
- Verify Packet
- Prepare Commit
- Verify Packet
- Prepare Commit
- Verify Packet
- Prepare Commit
- Detect Packet and Relay
- Confirm Status
- Create Packet
- Execute Commit
- Create Packet
- Execute Commit
- Create Packet
- Execute Commit
- Create Packet and Submit
- PacketPrepare
- PacketPrepare Acknowledgement
- PacketCommit
- PacketCommit Acknowledgement
- Tx Done

**Transaction e.g. send token and change owner of ticket**
- Store changed state, get locked to resources by executing contract
- Ask Commit after all participants returns OK by Ack Packet

Copyright © 2022 Datachain, Inc. All Rights Reserved.
3. Confirm Step

Transaction e.g. send token and change owner of ticket

Prepare Phase

- Create InitiateTx and Submit
- Initialize Tx State
- Create Packet
- Verify Packet
- Prepare Commit
- Verify Packet
- Prepare Commit
- Verify Packet
- Prepare Commit
- Detect Packet and Relay
- Store changed state, get locked to resources by executing contract
- Ask Commit offer all participants returns OK by Ack Packet
- Confirm Status
- Create Packet
- Execute Commit
- Create Packet
- Execute Commit
- Create Packet
- Execute Commit
- Detect Packet and Relay

Commit Phase

- PacketPrepare
- PacketPrepareAcknowledgement
- PacketCommit
- PacketCommitAcknowledgement
- Tx Done
Cross Framework: 2 Phase Commit Protocol

4. Commit Step

**Ledger**
- Coordinator Chain
- Chain A
- Chain B

**Role**
- Coordinator
- Participant
- Relayer

**Prepare Phase**
- Transaction: e.g., send token and change owner of ticket
- Create InitiateTx and Submit
- Initialize Tx State
- Create Packet
- Detect Packet and Relay
- Verify Packet
- Prepare Commit
- Verify Packet
- Prepare Commit
- Verify Packet
- Prepare Commit
- Detect Packet and Relay
- Confirm Status
- Create Packet
- Detect Packet and Relay
- PacketPrepareAcknowledgement

**Commit Phase**
- Store changed state, get locked to resources by executing contract
- Ask Commit after all participants returns OK by Ack Packet
- Create Packet
- PacketCommit
- Detect Packet and Relay
- Execute Commit
- Create Packet
- Execute Commit
- Create Packet
- Execute Commit
- Create Packet and Submit
- PacketCommitAcknowledgement

**Transaction**
- e.g., send token and change owner of ticket
- Create Packet
- Create Packet
- Create Packet

**Commit**
- Tx Done
Cross Framework: Simple Commit Protocol

Example of Simple Commit Protocol

Chain A

Role:
- Coordinator & Participant
- Relayer

InitiateTx

Initialize Tx State

Prepare

Create Packet

PacketDataCall

Relaying the Packet

PacketDataCall

Verify Packet

PacketCallAcknowledgement

Commit

PacketCallAcknowledgement

Relaying the Ack

PacketCallAcknowledgement

Create Ack

Chain B

Role:
- Participant

Ledger

Role:
- Coordinator & Participant
- Relayer

Save change operations by executing contracts
Acquiring locks on each resource

Copyright © 2022 Datachain, Inc. All Rights Reserved.
1. Initiate Step

**Cross Framework: Simple Commit Protocol**

**Ledger**

**Chain A**
- Coordinator & Participant
  - InitiateTx
  - Initialize Tx State
  - Prepare
  - Create Packet
  - Commit

**Chain B**
- Participant
  - Verify Packet
  - Commit
  - Create Ack

**Role**

- Coordinator & Participant
- Relayer
- Participant

**Diagram Notes**
- Save change operations by executing contracts
- Acquiring locks on each resource

Copyright © 2022 Datachain, Inc. All Rights Reserved.
2. Prepare Step

Ledger

Chain A

Coordinator & Participant

Relayer

Chain B

Participant

InitiateTx

Initialize Tx State

Prepare

Create Packet

Relaying Packet

PacketDataCall

PacketCallAcknowledgement

Commit

Relaying Ack

PacketDataCall

PacketCallAcknowledgement

Verify Packet

Commit

Create Ack

Save change operations by executing contracts
Acquiring locks on each resource
3. Commit Step on Chain B

Chain A:
- **Coordinator & Participant**
  - **InitiateTx**
  - **Initialize Tx State**
  - **Prepare**
  - **Create Packet**
  - **Commit**

Chain B:
- **Relayer**
  - **Relaying the Packet**
  - **Verify Packet**
  - **Commit**
- **Participant**
  - **Relaying the Ack**
  - **Create Ack**

**PacketDataCall**
- Save change operations by executing contracts
- Acquiring locks on each resource

**PacketCallAcknowledgement**
4. Commit Step on Chain A

Cross Framework: Simple Commit Protocol

Ledger

<table>
<thead>
<tr>
<th>Role</th>
<th>Chain A</th>
<th>Chain B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinator &amp; Participant</td>
<td></td>
<td>Participant</td>
</tr>
<tr>
<td>Relayer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **InitiateTx**
  - **Initialize Tx State**
  - **Prepare**
    - **Create Packet**
      - **Relaying the Packet**
        - **Verify Packet**
          - **Commit**
            - **Create Ack**
      - **PacketDataCall**
      - **PacketCallAcknowledgement**
        - **Relaying the Ack**
          - **Relayer**
    - **PacketDataCall**
      - **PacketCallAcknowledgement**
  - **PacketDataCall**
    - **PacketCallAcknowledgement**
      - **Commit**

Save change operations by executing contracts
Acquiring locks on each resource
Cross-chain Calls
Support “cross-chain calls” for general cross chain transaction

Not just asset transfer, Cross Framework supports general cross chain transaction.

Developers can build applications that invoke smart contracts on one chain from the other by implementing contract functions on the Contract Module.

This is called `cross-chain call` in Cross Framework
Support "cross-chain calls" for general cross chain transaction

Example Scenario

1. Registration
2. Query for KYC information
3. If Eligible for aid
4. Proceed for payment

KYC chain
Government chain
Payment/Coin chain

Subsidy payment
**Example: Token Transfer among different chains**

**Pseudo Code**

```
// ChainA
func Deposit(store Store, from Account, amount uint64) bool {
    balance := store.GetInt(from)
    if (balance < amount) {
        return false
    }
    store.SetInt(fromID, balance-amount)
    store.AddInt(contractAddress, amount)
    return true
}

// ChainB
func Peg(store Store, to Account, amount uint64) {
    ret := Call{
        ChannelInfo{
            Port: "transfer",
            Channel: "chainA",
        },
        ContractCallInfo{
            Func: ChainA::Deposit,
            Args: {ctx.SignerAccount, amount},
        }
    }
    assert(ret == true)
    store.AddInt(to, amount)
}
```
Cross-chain function calls are resolved in InitiateTx process

Link is functionality to associate contract transactions that make cross-chain calls. The initiator chain resolves each link to the corresponding result of the call using Linker when the MsgInitiateTx is submitted.
Cross-chain function calls are resolved in InitiateTx process

Link is functionality to associate contract transactions that make cross-chain calls.

The initiator chain resolves each link to the corresponding result of the call using Linker when the MsgInitiateTx is submitted.

message InitiateTx {
  string chain_id = 1;
  uint64 nonce = 2;
  CommitProtocol commit_protocol = 3;
  repeated initiator.ContractTransaction contract_transactions = 4;
  repeated auth.Account signers = 5;
  ibc.core.client.v1.Height timeout_height = 6;
  uint64 timeout_timestamp = 7;
}

message ContractTransaction {
  google.protobuf.Any cross_chain_channel = 1;
  repeated cross.core.auth.Account signers = 2;
  bytes call_info = 3;
  cross.core.tx.ReturnValue return_value = 4;
  repeated Link links = 5;
}

...
Cross-chain function calls are resolved in InitiateTx process

Then, a ResolvedContractTx is generated from the caller's ContractTx and the call result.

```protobuf
message ResolvedContractTransaction {
  google.protobuf.Any cross_chain_channel = 1;
  repeated cross.core.auth.Account signers = 2;
  bytes call_info = 3;
  ReturnValue return_value = 4;
  repeated google.protobuf.Any call_results = 5; // List of CallResult
}
```

When a Contract Transaction is verified that the arguments ChannelInfo and ContractCallInfo of the Call function match the values of CallResult of the corresponding ResolvedContractTransaction
Contract Module: Linking Cross-chain calls

Each expected return value is given when the ContractTransaction is created. (Values are calculated off-chain).

- No dynamic on-chain execution of external contract calls, allowing for the fast cross-chain calls
- Currently, it is not suitable to apply to frequently changing values (e.g., price Oracle) because the values at the time when InitiateTx created are used.
  - We will improve this to allow specifying within acceptable limits (not implemented yet)
Conflict-free Data Types (CDT)
CDT provides a data structure that can be read and written the same state in concurrency in distributed transactions.
Cross Framework: Conflict-free Data Types (CDT) Overview

CDT has the following properties and operations:

1. Transaction can update the state without acquiring an exclusive lock
2. Subsequent transactions can update the state and perform the comparison operations under certain conditions
3. If the above operation 2 is failed, an "indefinite" error is raised and the transaction is aborted

Support Data Types and Operations

- Integer
  - operations
    - Add, Subtract
    - Compares: LT(E), GT(E), EQ
- Grow-only Set
  - operations
    - Add
    - Lookup

ex) GTE

\[
\text{GTE}(\text{Target}, \text{Value}) : \\
\begin{cases} 
\text{true} & \text{if Target} \geq \text{Value} \\
\text{indefinite} & \text{if Target} = \text{Value} \\
\text{false} & \text{if Target} < \text{Value}
\end{cases}
\]
Cross Framework: Conflict-free Data Types (CDT) Example

**Example:** One of three tx fails to assert with GTE

- Each balance is managed using an Integer store provided by CDT
- In each action that needs pre-commit, assert that sufficient balance exists using GTE operation of the Integer store
- If failed, return an indefinite error

---

e.g., balances in ERC-20

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Balance</th>
<th>GTE(Balance, 30)</th>
<th>Assertion</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx1: Transfer(Bob, 30)</td>
<td>100</td>
<td>True</td>
<td>Min &gt;= 30</td>
<td>pre-commit</td>
</tr>
<tr>
<td>Tx2: Transfer(Carol, 40)</td>
<td>100</td>
<td>True</td>
<td>Min &gt;= 40</td>
<td>pre-commit</td>
</tr>
<tr>
<td>Tx3: Transfer(Dave, 40)</td>
<td>100</td>
<td>Indefinite</td>
<td>Min &lt; 40 ∧ 40 &lt;= Max</td>
<td>pre-commit</td>
</tr>
</tbody>
</table>

**Balance:**
- **Min:** Value when all subtraction operations succeed
- **Max:** Value obtained when all addition operations succeed

**Commit:** (Tx1 and Tx2)

Tx3 fails, however, Tx1 and Tx2 still can be committed
Several development topics for further improving Cross Framework

Support for other ledgers
- Solidity (Support for coordinator, lock mechanism, etc.)
- Corda

More data structures for CDT
- Other data types like boolean, (variant) set etc.
- More research to apply other operations

Devs. support (docs, samples and tools)
- The document website is now available: https://datachainlab.github.io/cross-docs/
- Cross-chain-swap tutorial (and its source code) is coming soon
- Explorer
References:

Cross Framework
- Cross Framework
  - [https://github.com/datachainlab/cross](https://github.com/datachainlab/cross)
  - [https://github.com/datachainlab/cross-solidity](https://github.com/datachainlab/cross-solidity)
- CDT [https://github.com/datachainlab/cross-cdt](https://github.com/datachainlab/cross-cdt)
- Docs [https://datachainlab.github.io/cross-docs/](https://datachainlab.github.io/cross-docs/)

YUI
- [https://labs.hyperledger.org/labs/yui.html](https://labs.hyperledger.org/labs/yui.html)
- Docs [https://github.com/hyperledger-labs/yui-docs](https://github.com/hyperledger-labs/yui-docs)
Thank you!

Contact us:
- https://datachain.jp/
- pr@datachain.jp
- @datachain_en