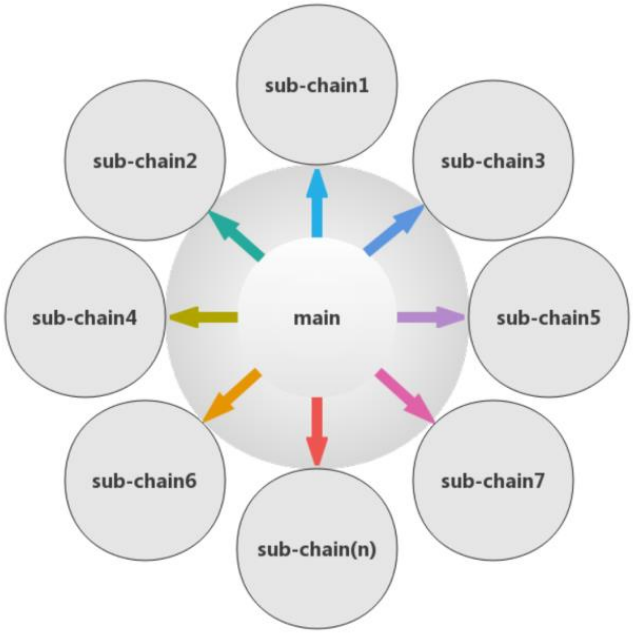
MANX Cross Chain Technology

Topic 1: Design of MANX Main Chain and Sub-Chain Structure:

MANX is built on a hierarchical subchain architecture that contains a core main chain and configurable subchains, where the subchains coordinated with the main chain. The main chain is the cornerstone of the system. It contains the basic token system, the core consensus mechanism, and the cross-chain protocol for configuring and coordinating the subchains. In the cross-chain protocol of MANX, there are two scenarios:

1. **The interaction between the main chain and the subchain**

   Subchains are configured by MANX's generation template and can have their own independent consensus and storage mechanisms. After the subchain generates a block, it will communicate with the main chain at regular intervals, so that the main chain can obtain the subchain block header and other statistical information. In addition, the main chain will generate related Merkle proof data of the subchain. The MANX main chain generates the corresponding Merkle proof data for each generated subchain.



Interaction between main chain and subchain

As shown in the figure, the main chain serves as the hub for subchain interactions, which is a distributed hub in blockchain structure.

2. **The interaction between the main chain and the outer chains**

MANX adopts the relay chain approach to support more generalized cross chain communications, which enables the exchange of data between multiple blockchain systems. The relay chain is a special subchain in MANX. In fact, the relay chain is a more general form of the Hub structure described above. We can also apply the relay chain approach to exchange data between MANX main chain and subchain. However, the Hub structure is more efficient in this case.

Topic 2: MANX Cross Chain Functionalities:

The MANX cross chain mechanism supports asset exchanges between different chains, and also the following cross chain functions:

1) Transfer of digital assets between multiple chains, including MANX main chain, subchains, and outer chains
2) Atom exchange between multiple chains
3) Monitoring and verification of event information from other chains
4) A unified account mapping model among multiple chains to facilitate multi-chain asset management

In the entire MANX cross-chain design, the main chain maintains function purity, responsible for the basic macro-chain economic model and coordination with all subchains. The Main chain also provides underlying function calls for subchain configuration. The MANX main chain maintains the justice of the distributed network.

Topic 3: MANX Application-oriented Hierarchical Subchain Design

Each subchain can focus on its own business logic while the main chain provides infrastructure services. Theoretically, this approach can provide unlimited performance extension. Subchains define their own business logic, such as the application templates provided by MANX: websites, exchanges, payments, insurance, investment advice, social networking, games, and e-commerce.

The ledger data of various applications does not need to be stored on the main chain, rather each can have their own independent storage. However, the configuration information for the subchain and block headers of daily operations needs to be sent to the main chain. The core main chain stores the data after verification and confirmation. The subchain can be

connected to all the nodes of the main chain, and it can also be connected only through the main chain block header.

The instruction set of a application, such as a script running on a subchain must come from the core main chain, namely, a subset of the core main chain. The subchain script program can directly access nodes and contracts deployed on the main chain through the protocol channel. However, the main chain does not necessarily have full access to a subchain. The subchain will have an isolation model and can control the data interface available to the main chain and the outside world.

Although MANX has a hierarchical chain structure, the operations of each chain are equal and asynchronously decoupled. Applications on a subchain do not need to frequently query, call or verify to the main chain. They only need to report block header data to its parent chain as a certificate at regular intervals (the parent chain node will store the certificate after verification).

Topic 4: Comparison of MANX with other Cross Chain Techniques:

PROJECT	MANX	Corda, Interledge	Polkdot, COSMOS	Atom Swap	WanChain Fusion, EKT
Principle	Relay chain + variable mapping + monitor/execute isolation strategy + controlled variable management contract	Notary model	Relays	Hash lock	Distributed private key control
Fairness: (Decentralization Degree)	Highest	Low	High	High	High
Cross-chain asset transfer	Support	Support	Support	Only Support Asset Exchange	Support
Cross-chain smart contract	Support	Not Support	Support	Not Support	Not Support
Multi-chain, multi-contract, multi-variable distributed service	All Support	Not Support	Part Support	Not Support	Not Support

Cross Chain Capabilities Comparison

Topic 5: MANX Distributed Service Interaction Protocol

Cross-chain distributed services means that the processing steps of a transaction are distributed across different blockchains while the entire transaction has guaranteed consistency. Cross-chain asset exchange is the initial stage of heterogeneous blockchain multi-chain integration, while distributed services are an advanced stage of multi-chain integration, which extends the specific asset exchange transaction to an arbitrary transaction. Cross-chain smart contracts make cross-chain distributed services possible. Different parts of a smart contract can execute on different blockchains, or all parts could be executed on the same blockchain. If any part of a smart contract fails to execute successfully, the entire contract is rolled back to its pre-execution state. Cross-chain smart contracts significantly expand the application of blockchain. Distributed services are no longer limited to a single chain but instead can execute with multi-chain, multi-contract, multi-variable collaborative adaptation and heterogeneous integration.

The MANX distributed service solution coordinates k users to make transactions between k public chains. We denote k public chains as C_i , $i = 1,2, \dots, k$, and denote the relay chain as R.

The cross-chain participants are divided into three categories:

1. **User nodes**: Directly participate in the relay chain and the update of each public chain state machines. The user nodes are denoted as $USER_i, i = 1,2, \dots, k$

2. **Monitoring nodes**: Monitor the digital assets in each public chain and update state variables involved in cross-chain distributed services. The updates are packaged as the input of the relay chain smart contract, driving the relay chain contract status. Denote the monitoring nodes as $MON_i, i = 1,2, \dots$

3. **Execution nodes**: Monitor the result of the relay chain smart contract state machine. The execution results are packaged as the input of each public chain smart contract and drive the execution of the contract state machine. Execution nodes are denoted as $EXE_i, i = 1,2, \dots$

Monitoring nodes and execution nodes selection: The selection principles include the node credit value, the margin adequacy and stability. The node credit value is the service rate for a monitoring node or an execution node. The margin adequacy is whether the node has enough MANX tokens as the asset collateral, and the node stability refers to the continuity of online service.

Account creation requirements: The user node has accounts in each associated public chain $USER_1(C_j), 0 \leq j \leq k$, and has relay chain account $USER_1(R)$. Similarly, the monitoring node is required to have relay chain account $MON_i(R)$, and the execution node is required to have the relay chain account $MON_i(R)$.

Margin mechanism: Each cross-chain distributed service has a corresponding value metric. When a user node invites a monitoring node and an execution node to perform cross-chain distributed services, the monitoring node and the execution node put certain MANX tokens into the relay chain smart contract. Similarly, the user node needs to deposit a certain number of MANX tokens as a margin for compliant execution of the cross-chain transaction.

Monitoring node and execution node incentive mechanism: In each cross-chain distributed service, each user node deposits a certain number of MANX tokens as a prepaid service fee. The service charge = margin * interest rate * contract execution period + fixed service fee. If the actual service fee > prepaid service fee, the relay chain contract determines the user node at fault for causing the delay according to the monitoring node input. The blamed user node will lose all prepaid service fees, and other users' prepaid service fees will be returned. Otherwise, after the execution, the relay chain contract automatically allocates work according to the actual service fee, and returns the remaining service fee (prepaid service fee - actual service fee).

The penalty mechanism for monitoring nodes and execution nodes: If certain monitoring nodes or execution nodes fail to perform their duties in time, their margin will be reduced by a certain percentage to compensate other compliant nodes.

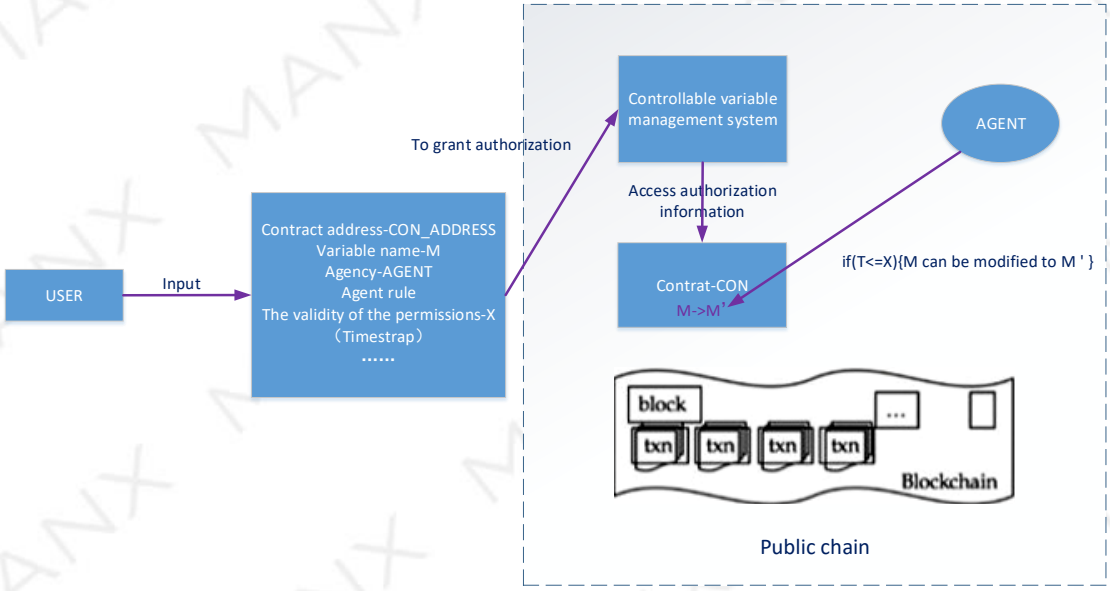
Relay chain contract: The relay chain smart contract includes mechanisms for event processing and saving, as well as a complete state machine for accepting and processing smart contracts. Data status processing is completed in the contract. The smart contract will run state machine judgment on the input event information. If the trigger condition of one or more actions is satisfied, the state machine will run the pre-set contracts automatically.

The atomicity of the relay chain contract state machine: The smart contract follows the atomic principle. The state variables are either all updated to a new state, or are all returned to the original state.

Arbitration mechanism: MANX Relay's arbitration mechanism is implemented through a built-in Simple Payment Verification (SPV) mechanism. An external third party,

called Relayer, sends a transaction to MANX Relay's smart contract, which contains the block headers of latest *k* public chains. MANX Relay verifies the transmitted block header based on the existing block header information. If the check passes, the transaction is added to the block headers maintained by MANX Relay. A built-in SPV node is implemented in the arbitration smart contract within MANX Relay. If a user node doubts the correctness of the information sent by the monitoring node, it can demand SPV verification. If the result shows that the monitoring node is cheating, the monitoring node will be punished by the built-in logic.

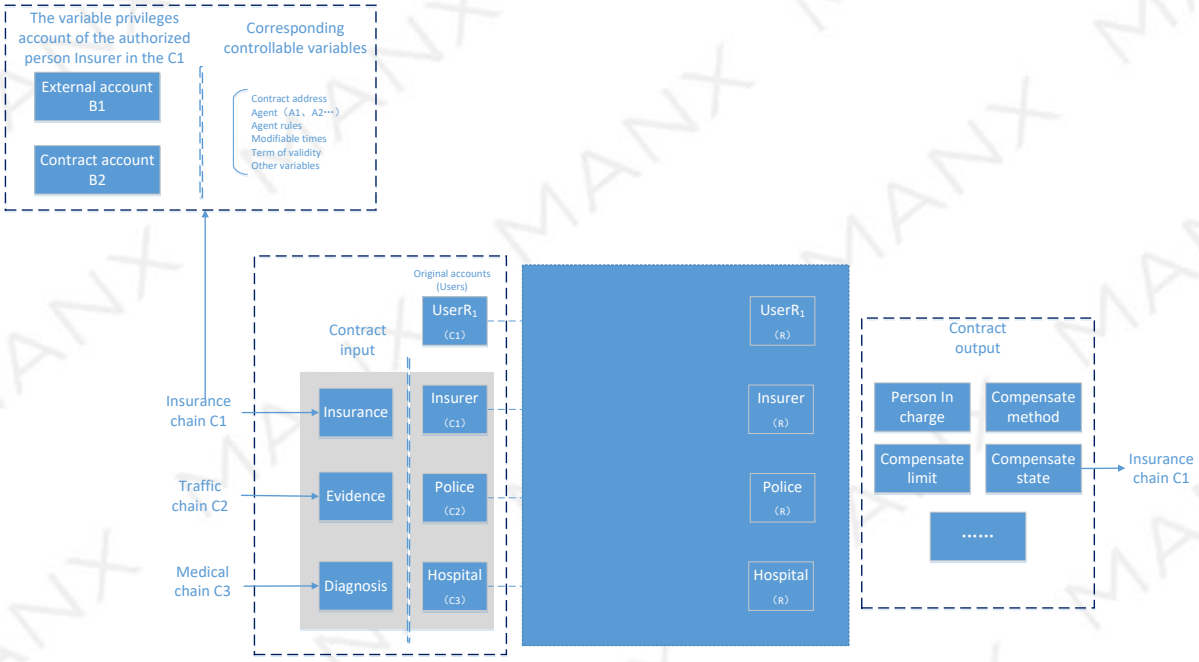
Controllable variable management system: Any USER can establish a management system on each public chain. Through this system, USER can authorize other agent nodes to modify the variable in the contract he created or the variable in other contracts that he himself has the modification permission. The right to use this system is restricted to USER himself. Every time USER creates a contract CON on public chain C, the address of CON is included in the controllable variable management system. The user node enters the CON_ADDRESS, the variable name M, the agent node AGENT, the agent rule, and the validity period X of the modification authority into the controllable variable management system, and then performs authorization. As long as time T is within the valid period X (T≤X), the AGENT can modify the variable M in the CON. If time T exceeds the valid period X (T>X), the AGENT will lose the modification rights.



Controllable variable management system

Topic 6: Distributed Business User Case:

Assume that the $USER_1$ unfortunately had a traffic accident. He had previously purchased a traffic accident insurance. The insurance contract information was stored by the insurance company INSURE on the insurance chain C1, and the claims conditions include the EVIDENCE information issued by POLICE and DIAGNOSIS information issued by HOSPITAL. The EVIDENCE information is stored on the traffic chain C2, and the DIAGNOSIS information is stored on the medical chain C3. If $USER_1$ wants to make a successful claim, he needs to combine the CONTRACT information on C1, the EVIDENCE information on C2 and the DIAGNOSIS information on C3 together to produce the claim result. Finally, the claim result needs to be updated to insurance chain C1. This is called distributed service which involves multi participants and is based on cross-chain smart contracts.



The overall distributed service architecture of insurance, traffic, medic

The detailed process is as follows:

1. Account application. $USER_1$ already has C_1 account $USER_1(C_1)$ and apply relay chain account $USER_1(R)$； Police office already has C_2 account $POLICE(C_2)$ and apply relay chain account POLICE(R)； Hospital already has C_3 account $HOSPITAL(C_3)$ and apply relay chain account $HOSPITAL(R)$。

2. Relay chain contract establishment. $USER_1$, $POLICE$, $INSURER$, $HOSPITAL$ agree to use the relay chain R as the cross-chain operating environment to establish cross-chain smart contract which includes the following information:

2.1. Mapping of distributed information or states. Existing public chain variables are called original images, and relay chain variables are called mirror images. They must be consistent. Original image assures the validity of the relay chain input information.

$$C_1.\text{contract\_policy.insurer.policy} \leftrightarrow \text{R.this.insurer.policy}$$
$$C_2.\text{contract\_evidence.police.evidence} \leftrightarrow \text{R.this.police.evidence}$$
$$C_3.\text{contract\_diagnosis.hospital.diagnosis} \leftrightarrow \text{R.this.hospital.diagnosis}$$

2.2. Set the user nodes deposit dep(user1)，dep(police)，dep(hospital), dep(insurer) and the prepaid service fee p(user1)，p(police)，p(hospital), p(insurer)

2.3. Monitoring node and execution node matching requirements. node credit value, margin adequacy, and node stability.

2.4. Margin requirements for monitoring nodes and execution nodes. dep(MON), dep(EXE).

2.5. Contract Main Logic. Specifies contract inputs and outputs, state machine jump conditions, and target state. The input is the condition variable involved in the jump condition, and the output is the result variable of each state. The target state is the final goal of the users in the cross-chain distributed transaction.

```
if (R.this.police.evidence == TRUE
    R.this.hospital.diagnosis == TRUE)
  {R.this.insurer.compensate(R.this.user1)}
else
  throw;
```

2.6. Contract Incentive and Punishment Logic. User node, monitoring node and execution node are rewarded and punished according to the contract state machine's status and the specific execution result.

2.7. Contract arbitration Logic. Involved in incentives and penalties after the arbitration mechanism takes effect.

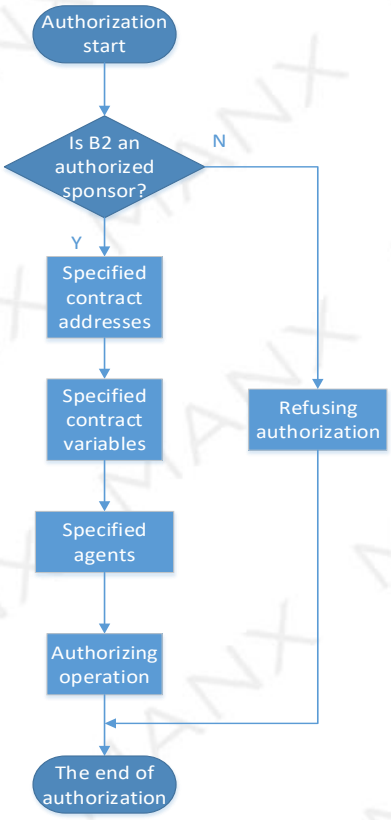
2.8. Contract validity period. If the relay chain contract state machine fails to jump to the target state within the validity period, the margin and assets will be returned. The delaying node will be penalized.

2.9. Validity of arbitration. Within the validation period, the user node can submit arbitration request to the arbitration contract.

3. Monitoring nodes and execution nodes join. Monitoring nodes and execution nodes that meet the requirements join the contract. The execution nodes jointly apply for execution accounts based on the (n,k) threshold in the related public chains, that is k execution nodes need to agree before initiating the transaction when there are n execution nodes. Here, the output related variable is R. this. insurer. compensate, and the execution node will apply execution account C_1. contract_policy. EXE_n(n $\geq$ 1)

4. Authorize output agent. In C_1, $INSURER$ has two accounts that can make changes to the contract_policy. One is the external account B1 and the other is the contract account B2. B1 is $INSURER$'s account on public chain C1, which can be understood as the public key address. B1 is the ID card of $INSURER$ on C1. B2 is the address of the contract which manages variable modification authority. $INSURER$ can authorize other execution code through B2. The authorization can be narrowed to a specific variable in a specific contract. During the claims process, $INSURER$ can always execute claim operations. $INSURER$ can also authorize some execution node C1. contract_policy. EXE as its agent through B2. The authorization operation is as follows:

```
    if (msg. sender == B2)
    {    addressOfcontract = contract_policy;
variable=>[C1.contract_policy.insurer.compensate]
        Agent=>[C1. contract_policy. EXE1,……, C1. contract_policy. EXEn]
        Rule => {……}
    }
```



Authorization procedure

5. Mirror Image disposal. The monitoring node monitors the original images C1, C2, C3(C1.contract_policy.insurer.policy, C2.contract_evidence.police.evidence, C3.contract_diagnosis.hospital.diagnosis). If the status of the original image changes, monitoring node sends the synchronization demand the relay chain contract (R.this.police.evidence, R.this.insurer.policy, R.this.hospital.diagnosis). If the number of monitoring nodes reaches a certain proportion, the contract will operate the mirror image according to the original image. C1. contract_policy. insurer. policy →
R. this. insurer. policy , C2. contract_evidence. police. evidence →
R. this. police. evidence, C3. contract_diagnosis. hospital. diagnosis →
R. this. hospital. diagnosis

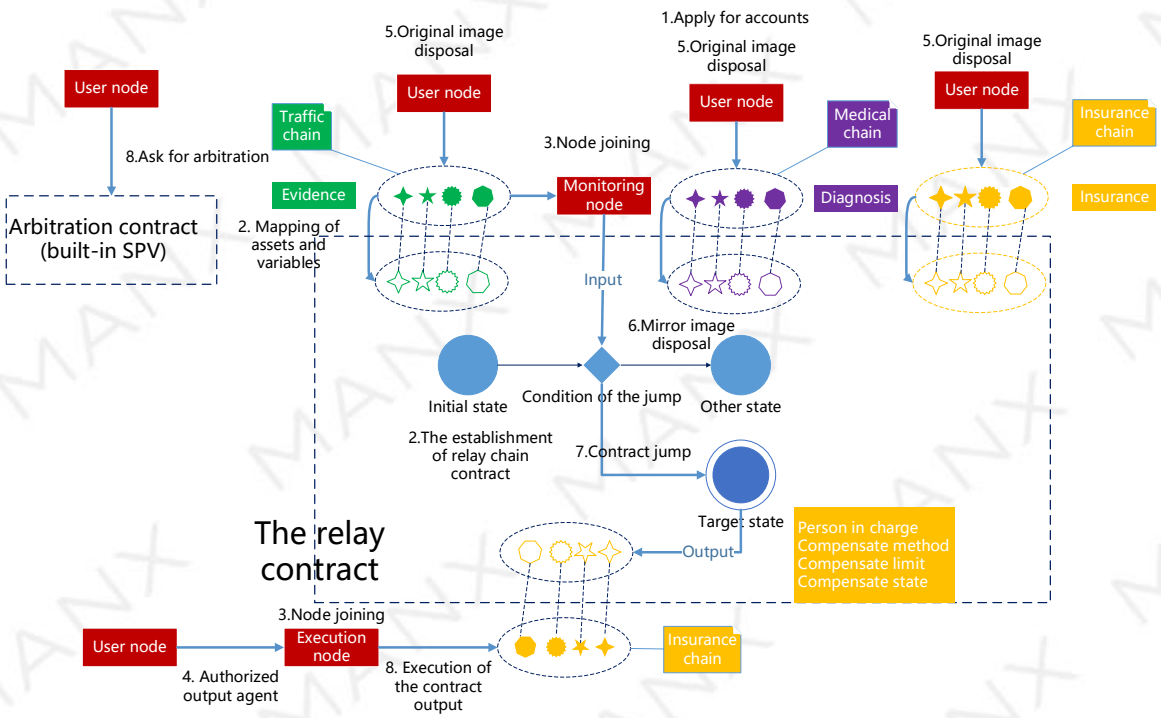
6. Relay chain contract state jump. The contract jumps according to the main logic execution status and outputs the result. Here, the relay chain state jumps and obtains the target variable
R. this. insurer. compensate(Owner: USER1, Claim method: Pay C1 token,
Amount: 5, Status: N)

7. Original image disposal. The user node or the authorized execution node operates the preimage according to the relay chain contract state machine. If $INSURER$ chooses to make the claim on his own, then $INSURER$
executes C1. contract_policy. insurer. compensate. After confirmation, $C1.contract\_policy$ automatically excutes $C1.insurer.coin \rightarrow$ C1. user1. coin. If the agent node operates, (e.g. there are n agent nodes), then $C1.contract_{policy}.EXEm.(0 < m < n)$. execute C1. contract_policy. insurer. compensate. If $(m > n/2)$, $C1.contract\_policy$ excutes $C1.insurer.coin \rightarrow$ C1. user1. coin

8. Apply for arbitration. During the validation period of the arbitration, if the user node does not agree with the information sent by the monitoring node to the contract, it can submit the relevant SPV certificate information to the arbitration contract on the relay chain. All nodes of the relay chain execute the arbitration contract and give the final arbitration result based on the public chain block header.

9. Execute reward and punishment: After the arbitration period expires, the contract settles transaction costs, rewards and penalties for the user node, monitoring node and execution node. Then, the relay chain contract halts.

10. The contract halts automatically if its validity period expires. If the contract expires before the contract state machine jumps to the target state, the relevant user nodes return to the initial state. The overdue responsibility will be investigated and the execution penalty will be made to the delaying node.



Detailed procedure