Free Market Protocol Architecture

December 28, 2022

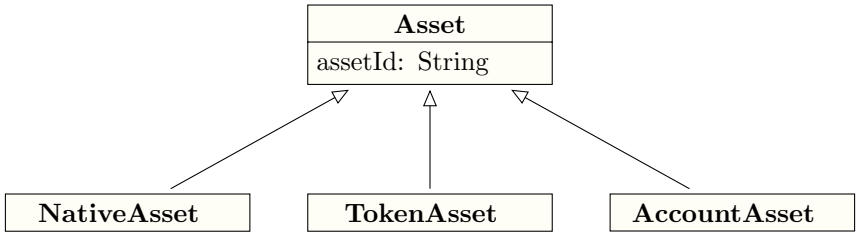
The main function of FMP is to transform assets from a 'start state' into a 'target state' by executing a workflow. Both start state and target state consist of one or more cryptographic assets. FMP strives to execute these workflows as efficiently as possible, incurring the least amount of gas fees and slippage as possible, while also requiring a minimal amount of end-user 'approvals' as possible.

Workflow Domain Model

The following section provides an overview of the FMP Workflow domain model.

Assets

FMP Workflow can operate on any of the following kinds assets:



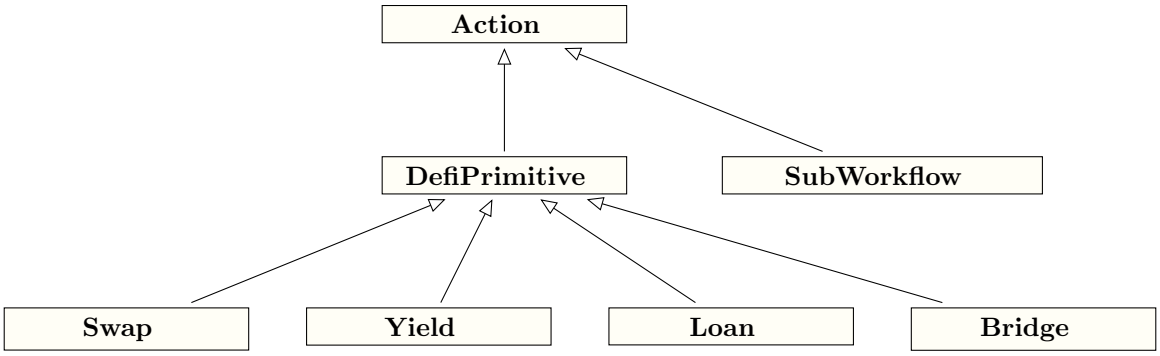
- **NativeAsset**: The asset that is native to a blockchain, for example Ether (ETH) on Ethereum blockchain, or SOL on Solana.
- **TokenAsset**: A fungible token, for example "USD Coin" (USDC) which exists as an ERC20 on Ethereum and as an SPL-Token on Solana.
- **AccountAsset**: An account with some blockchain protocol that holds a balance of some asset. For example, when a user deposits tokens into a DeX such as DyDx, the tokens are transferred from the user to the exchange, and in return the Dex maintains a balance of those tokens in an account associated with the user.

For simplicity, an account with a negative balance is also modeled as an "asset." It is not referred to in the model as a liability.

Many assets such (for example USDC) exist on many chains. In our model, assets are associated with exactly one chain, so for example USDC on Ethereum is considered a different asset than USDC on Polygon.

Actions

Actions are the individual transformation steps within a Workflow. An action has one or more input assets and one or more output steps, but typically have one input and one output.



- **Defi Primitives** are atomic operations within a workflow. They are implemented by integrating with existing third-party smart contracts.
  - **Swap** A simple swapping from one asset to another, E.g., a workflow action to exchange WBTC for USDC on Uniswap.
  - **Yield** An investment that generates returns over time. E.g., adding liquidity to an Automatic Market Maker pool such as Curve's 3Pool.
  - **Loan**: Providing collateral using one asset and borrowing a different asset. E.g., using USDC as collateral on Aave and borrowing BNB.
  - **Bridge**: Transferring assets from one blockchain to another. For example, using Wormhole to transfer USDC from Ethereum to Avalanche.
- **Sub Workflow** Since workflows have inputs and outputs (described in the next section), workflows can be nested inside of each other.

Action Notes

- The possible set input asset(s) for a given action are restricted to assets that that particular Action supports. For example, a Curve 3Pool Swap action can only accept USDT, USDC, or DAI as an input asset.
- Some Actions require additional configuration parameters, which are provided as an Action is included in a workflow.
- The output asset(s) is not configured explicitly, but is inferred by the Action's input asset(s) as well as any additional configuration parameters required by the Action.

Branches

A common pattern in DeFi workflows is to repeat a sequence of steps until some desired outcome is achieved. To support this, Branches allow flow of execution to be determined by evaluating a boolean expression against data contained in the workflow at runtime.

Workflows

Workflows represent configuration of reusable asset transformation logic using the elements described above. The definition of a workflow consists of:

- One *Input Assets* that represent the starting point of the workflow. The amount of that asset is set at execution time, it is not part of the workflow definition.
- A directed graph of nodes, where the nodes consist of *Actions* and *Branches*.

System Components

Free Market Protocol consists of the following components:

FMP-SDK: A high level programmatic interface enabling developers to easily integrate with the FMP system. FMP-SDK is written in Typescript and runs in the browser and NodeJS, allowing integration of FMP into web based UIs and stand-alone backend systems.

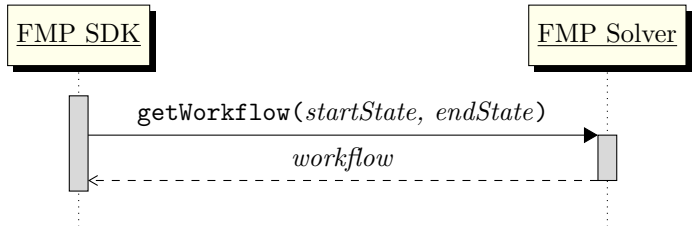
Workflow Solver: Given a start-state and a target-state, Workflow Solver will generate a sequence of low level workflow steps to transform the user's starting portfolio to the target portfolio. Workflow Solver is aware of approximate gas costs and slippage, and produces a workflow that minimizes these costs.

Workflow Engine: Using the workflow steps generated by Workflow Solver, Workflows Engine executes the steps on-chain. There is a separate implementation of Workflow Engine deployed on each supported blockchain. Workflow Engine carries out the low level workflow steps by invoking with 3rd party contracts. All steps in a workflow are executed within a single transaction.

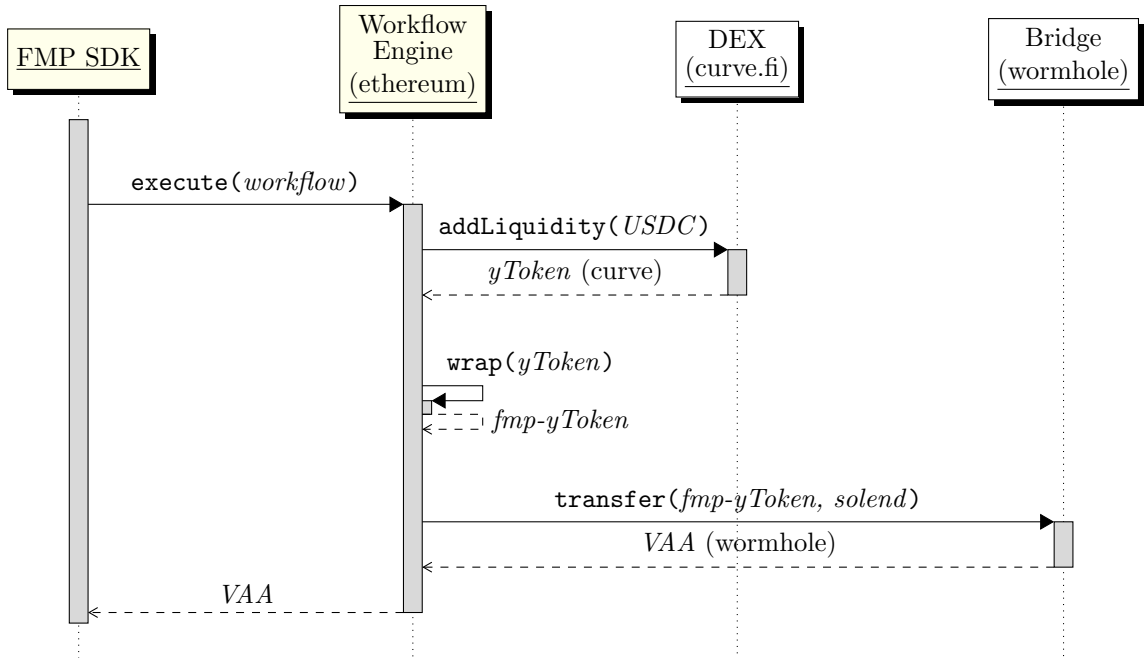
Component Interactions

The following illustrates a simple scenario and describes the interactions between the various components. In this scenario, a user starts with USDC in their Ethereum wallet, and they would like compose a new portfolio earning yield from Curve.fi on the Ethereum blockchain, and Solend on Solana.

First, FMP-SDK passes *startState* and *targetState* to Workflow Solver, representing the user's starting target crypto portfolios respectively. Workflow Solver then returns a complete set of low level workflow steps. Note that the workflow contains the required steps accross all blockchains involved in the portfolio transformation:

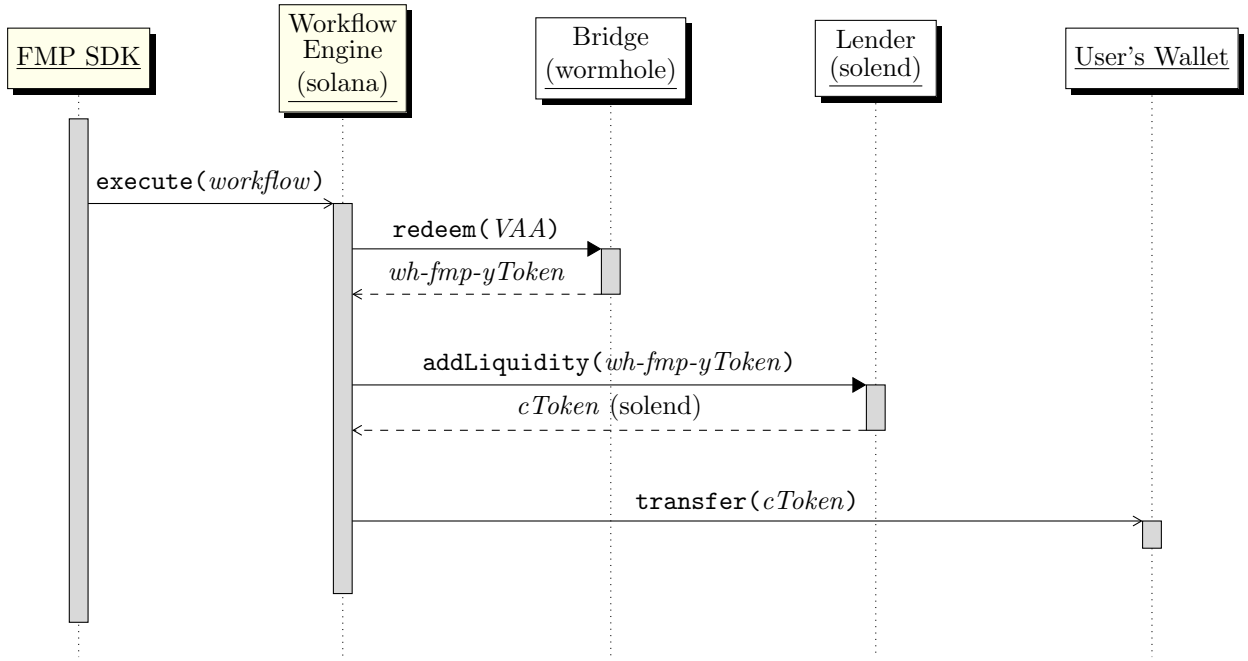


Next, FMP-SDK passes *workflow* to the Workflow Engine to execute the Ethereum part of the workflow:



In the scenario above, Workflow Engine first invokes Curve's addLiquidity method, providing USDC. Curve returns some quantity of *yToken*. In the next step, Workflow Engine wraps the *yToken*, creating a *fmp-yToken*. In the last step, the *fmp-yToken* is sent to Wormhole to be transferred to Solana. The *VAA* returned by wormhole a token needed to redeem the fmp-yToken on Solana. Note that this entire portion of the workflow happens on-chain and within a single transaction. As such, only one approval is needed from the user to execute all the steps in this workflow. Also, the workflow steps execute as an atomic unit; either all steps succeed or the entire transaction is discarded.

FMP-SDK continues the workflow by invoking the Workflow Engine on Solana:

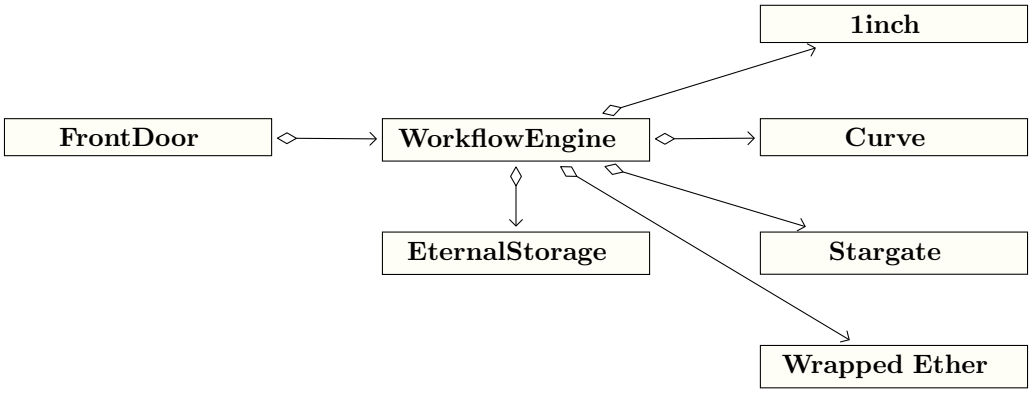


In the Solana portion of the workflow, Workflow Engine redeems the *fmp-yToken* which is now also

wrapped by Wormhole (*wh-fmp-yToken*). In the next step this is deposited into Solend in exchange for Solend's *cToken*. In the final step the *cToken* is transferred to the user's Solana wallet.

Deployment Architecture

The following diagram provides an example of how FMP smart contracts are deployed on-chain. Given that our target for or initial MVP release is EVM compatible blockchains, this diagram is specific to EVM.



Each box in the diagram represents a separatly deployed contract.

FrontDoor: This contract is the entry point for external callers. It provides a stable address and is not upgradable. It contains the address of the current version of the WorkflowEngine and simply delegatecalls all transactions to it.

WorkflowEngine: Contains the logic for parsing workflow definitions and carrying out the on-chain invocations.

EternalStorage: Provides all data persistance for WorkflowEngine via the Eternal Storage pattern described here: `https://fravoll.github.io/solidity-patterns/eternal_storage.html`

Step Implementations: (1inch, Curve, Stargate, etc) Workflow steps are deployed as separate contracts. This provides a great deal of modularity among the integrations and enables them to be upgraded independently.
IActionManager