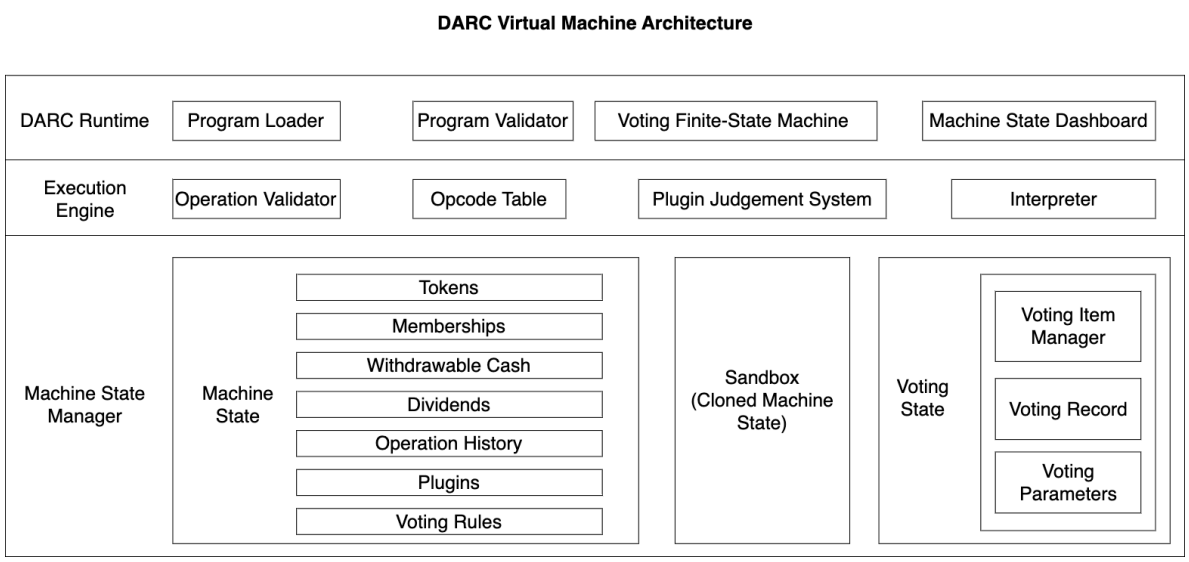

图 1: DARC 虚拟机的架构。

1 DARC 架构

DARC 是一个使用 Solidity 编程语言 [?] 构建的虚拟机，可以编译并部署到任何兼容 EVM 的本地 devnet、testnet 或 mainnet 上，而不受合约代码大小限制 (EIP-170) [?]。在 DARC 协议成功编译并部署到您的区块链之后，用户可以使用 darc.js（用于部署和与您的 DARC 交互的官方 Node.js SDK）或 ethers.js [?] 以及相应的 DARC 应用二进制接口（ABI）编写程序并发送至您的 DARC 虚拟机。用户也可以通过这种方式从部署的 DARC 中读取数据和信息。

图 1 展示了 DARC 协议的概览。

1.1 DARC 运行时

DARC 虚拟机的第一层是智能合约的应用二进制接口（ABI），允许用户编写程序并将其传递给 DARC 执行。程序加载器是所有通过使用 darc.js 或 ethers.js 从用户那里发送的程序的入口。

1.2 程序加载器

程序加载器是 DARC 运行时中接收用户程序的模块。作为 DARC 虚拟机的入口，程序加载器管理处理程序的整个过程，包括验证程序、执行程序、判断程序是否符合限制插件系统的要求、将程序发送到沙箱、启动投票、结束投票、暂停投票、将不必要的现金（原生代币）退回到账户余额。程序加载器是处理执行程序整个生命周期的基础和核心模块，以及 DARC 的投票期间的状态。

1.2.1 程序验证器

程序验证器是检查程序是否有效的模块，包括计算每个程序的原生代币总消耗、检查每个操作的基本语法和参数。当程序验证器接收到计算出的原生代币消耗量时，程序加载器将与用户发送的程序所附带的原生代币实际金额进行比较。如果发送的原生代币值大于总消耗，则程序将被执行，并且剩余的原生代币金额将被退回到用户的余额中；如果原生代币的值小于总消耗，则程序将直接被拒绝，并且所有原生代币将被退回到用户的余额中。

1.2.2 投票有限状态机

投票有限状态机（FSM）是处理投票程序生命周期的有限状态机，包括空闲状态（所有程序都可以提交和执行）、投票状态（只有包含“投票”操作的程序可以执行）和执行待定状态（允许用户执行之前通过投票过程批准的待定程序的一段时间）。

1.2.3 机器状态仪表板

机器状态仪表板是一组允许用户从 DARC 实体读取所有必要数据和信息的接口，包括 DARC 设置的基本信息和参数、所有代币持有者的多级代币系统余额、可提取的现金和分红余额、限制插件、投票规则和操作历史记录。机器状态仪表板中的所有函数都是只读的，并且用“view”关键字声明，用户可以在不需要额外燃料费的情况下读取信息。

1.3 执行引擎

执行引擎是核心模块，通过插件系统检查验证并在 DARC 虚拟机的机器状态和沙箱中执行每个操作。它类似于其他解释型编程语言的解释器，如 Java、Python、JavaScript、Perl 等。

在程序通过运行时程序加载器验证并发送后，它将直接传递给执行引擎。操作验证器是检查每个操作是否具有正确参数语法的模块，例如参数的长度和类型。在程序中的每个操作码和相应参数被操作验证器检查并验证后，它将准备好被插件判断系统检查并在操作码表和解释器内执行。

操作码表是分析每个操作的模块，带有操作码和参数，并在解释器中执行。当操作被发送到操作码表时，它将与参数一起被比较并发送到解释器执行在机器状态或沙箱中。

插件判断系统是核心模块，检查每个操作并为程序做出最终判断，决定程序应被接受并在机器状态中执行、被拒绝、在沙箱中执行并做出决定、挂起并开始投票以做出最终决定，或在沙箱中执行后被拒绝。插件判断系统从机器状态中读取两个限制插件数组：操作前插件数组和操作后插件数组。

当程序通过操作验证器时，它将首先被所有操作前插件检查：如果所有操作都被所有操作前插件批准并允许在机器状态中执行，则程序将被传递到操作码表并在解释器中执行；如果任何操作被任何操作前插件拒绝，则整个程序将被拒绝并且事务将被回滚；否则，如果没有操作被拒绝，但至少一个操作被任何操作前插件标记为“需要沙箱”，则程序的合法性尚未确定，整个程序需要在沙箱中执行。

在沙箱中执行后，操作和沙箱状态将由操作后插件检查：如果所有操作和沙箱状态都被所有操作后插件批准，则程序将被传回操作码表并在解释器中执行；如果任何操作或沙箱状态被任何操作后插件拒绝，则程序将被拒绝并且事务将被回滚；否则，如果没有操作被拒绝但至少一个操作被任何操作后插件标记为“需要投票”，则程序将被挂起并等待最终投票结果，并且只有在通过投票过程批准并在“执行待定时间”内执行后才被允许执行。

由于沙箱还包含所有操作前插件和操作后插件的副本，操作后沙箱检查仅使用当前状态中的操作后插件。这是因为程序的合法性应仅由当前机器状态决定，包括当前插件判断系统，而机器状态只允许由当前机器状态、当前插件判断系统和必要时的投票结果批准的程序更新。

执行引擎的工作流程如图 2 所示。

1.4 机器状态管理器

机器状态管理器是存储和管理 DARC 所有状态的模块。它包含三部分：机器状态、沙箱（克隆的机器状态）和投票状态。机器状态包含 DARC 虚拟机的所有必要状态和资产，包括代币系统、成员资格、可提取的现金和分红余额、操作历史、插件和投票规则。

1.4.1 代币

代币系统是 DARC 协议中最基本和核心的设计。机器状态包含一个代币数组，每个代币包含不同的投票权重、分红权重、代币余额、代币符号（代币信息）和总供应量。

每个级别代币的信息结构存储在机器状态管理器中，如下 Solidity 结构所示。

```
struct Token {
  uint256 tokenClassIndex;
  uint256 votingWeight;
  uint256 dividendWeight;
  mapping (address => uint256) tokenBalance;
  address[] ownerList;
  string tokenInfo;
  uint256 totalSupply;
  bool bIsInitialized;
}
```

代币余额是从地址到 uint256 的映射。键是当前级别代币的拥有者地址，值是该拥有者地址所拥有的代币数量。DARC 还维护一个地址数组 ownerList，作为拥有至少一个当前级别代币的所有地址的完整列表。

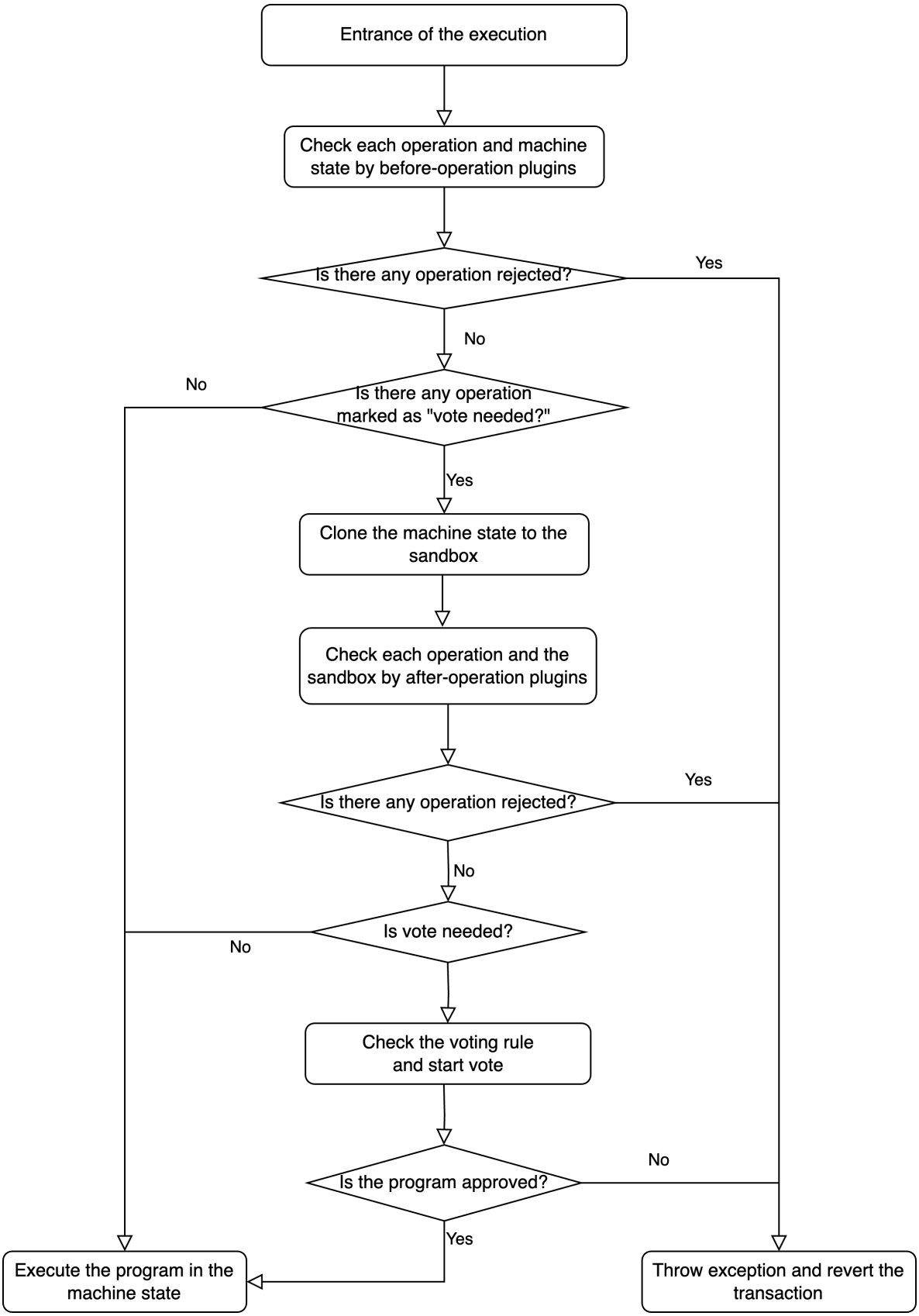

图 2: 执行引擎的工作流程。

1.4.2 成员资格

成员资格是一个从地址到级别的映射，使 DARC 能够层次化地管理不同地址。DARC 的治理可以确保不同级别的人被分配到不同的级别。

我们在 DARC 协议中引入成员资格的原因是，用户可以开发不同的插件来定义与角色和地址相关的不同规则。

在 DARC 协议的实现中，成员资格被定义为以下数据结构：

```
/**
* 每个代币拥有者的成员列表和内部角色索引号，
* 可用于代表股东、联合创始人、雇员、
* 董事会成员、特殊代理等。
*/
mapping (address => MemberInfo) memberInfoMap;

/**
 * DARC协议的股票代币拥有者信息
 * 此结构用于存储每个代币拥有者的角色索引号
 */
struct MemberInfo {
  bool bIsInitialized;
  bool bIsSuspended;
  string name;
  uint256 role;
}
```

1.4.3 可提取现金和分红

可提取现金和分红是两个独立的余额哈希映射。操作者可以从这两个余额中提取原生代币（如以太坊），如果余额不为零。对于可提取现金，向地址余额添加可提取代币的唯一方式是执行 BATCH_ADD_WITHDRAWABLE_BALANCE 操作，带有目标地址和代币金额。如果操作被批准，相应金额将被添加到余额中。对于分红，向余额添加可提取代币的唯一方式是执行 OFFER_DIVIDENDS 操作，不带任何参数。如果操作被批准，所有拥有分红权重大于 0 的代币持有者将收到相应的分红金额。

要将原生代币从 DARC 提取到地址，操作者可以执行 WITHDRAW_CASH_TO 操作，将可提取现金从余额转移到目标地址，或在操作被批准的情况下，执行 WITHDRAW_DIVIDENDS_TO 操作，将分红从余额转移到目标地址。

1.4.4 操作历史

操作历史是一个日志系统，包含每个地址及其所有相应操作的记录，每个操作都有其最新的时间戳。当程序成功执行后，程序中包含的操作的时间戳将在操作历史中更新。通过操作历史，开发者可以为单个地址或 DARC 中的所有成员设置每个操作之间的最小时间间隔。

规则 1 是一个使用操作历史限制操作者在一段时间内执行相同操作的示例：

规则 1：我们每月提供 10 ETH 作为雇员级别和经理级别地址（4 级和 5 级成员）的工资。

对于规则 1，我们需要确保如果操作者被分配的角色级别等于 4 或 5，操作是添加可提取余额，操作的总金额小于或等于 10 ETH（或 10000000000000000000 wei），并且操作者在超过 30 天（或 2592000 秒）的时间内进行了相同的操作，则该操作将被批准，且该操作无需进行沙箱检查。

这里是一个为规则 1 设计的 By-law 脚本中的插件示例：

```
const plugin_Rule_1 =
{
    condition: operation_equals(BATCH_ADD_WITHDRAWABLE_BALANCE)
    & total_add_withdrawable_balance_LE(10000000000000000000)
    & last_operation_by_operation_period_for_operator_GE(
        BATCH_ADD_WITHDRAWABLE_BALANCE, 2592000
      )
    & ( (operator_membership_level_equals(4))
         | (operator_membership_level_equals(5))
```

```
    ) ,
    returnType: YES_AND_SKIP_SANDBOX,
    bIsBeforeOperation: true,
    level: 100,
    votingRuleIndex: 0,
    note: ""
}
```

规则 1 没有限制 BATCH_ADD_WITHDRAWABLE_BALANCE 操作的目标地址。这是因为允许操作者向自己的可提取现金中添加 10 ETH，也允许他们向其他地址添加，只要他们在 30 天内添加的可提取现金总额不超过 10 ETH。

1.4.5 插件

插件是 DARC 协议中的基础机制，因为所有规则和规章都需要在 DARC 插件系统中定义、转译并保存。DARC 协议中有两个插件数组：操作前插件和操作后插件。

在图 2 中，提交给 DARC 的每个程序都需要通过所有操作前插件的检查，如果程序的任何操作违反了任何操作前插件，则程序将被拒绝。

如果所有操作都被批准且不需要沙箱检查，DARC 将直接执行程序；如果一个或某些操作需要在沙箱中检查，DARC 将首先将机器状态克隆到沙箱，然后在沙箱内执行整个程序，并根据执行结果和沙箱状态做出决定。

每个插件由以下项目组成：

- 返回类型：当条件被触发时插件的决策，包括 YES、VOTING_NEEDED、NO、YES_AND_SKIP_SANDBOX 和 SANDBOX_NEEDED。
- 等级：当条件被触发时当前插件的优先级。当操作触发多个插件时，插件系统将使用具有最高等级的插件的返回类型作为最终决策。由于每个等级只允许具有相同返回类型的插件，这将确保每个操作都从插件系统获得确定的判断结果。
- 条件节点：conditionNodes 是表示此插件表达式二叉树的条件节点数组。每个节点可以是一个带参数的条件表达式，用于验证某个标准，或一个逻辑运算符，将两个或多个子节点（条件表达式或逻辑操作）合并为单个条件表达式。
- 投票规则索引：如果条件表达式标准被触发并且插件的返回类型是 VOTING_NEEDED，则指向某个特定投票规则的索引号。如果需要投票过程，判断系统将遍历所有 VOTING_NEEDED 插件并创建一个带有选定投票规则的多项投票过程。
- 注释：插件设计者为将来目的留下的注释。
- 布尔标志 bIsEnabled：指示插件是否启用的布尔标志。由于插件在整个生命周期中是不可变的，操作者在成功部署后不能移除或修改插件。禁用某个插件的唯一方式是将布尔标志 bIsEnabled 从 True 设置为 False。如果操作被允许，也可以通过将其设置回 True 来启用插件。
- 布尔标志 bIsInitialized：指示插件是否成功初始化的布尔标志。如果 False，判断系统将跳过此插件。
- 布尔标志 bIsBeforeOperation：指示插件是操作前还是操作后插件数组的布尔标志。虽然操作前插件和操作后插件存储和管理在两个单独的数组中，但在判断系统遍历每个数组时将对此布尔值进行二次检查。这个字段是必需的，以确保在通过 By-law 脚本构建插件并通过 DARC 程序入口发送时，每个插件对象结构保持一致。

插件的结构在 Solidity 中以以下结构设计：

```
struct Plugin {
  /**
   * 当前条件节点的返回类型
   */
  EnumReturnType returnType;

  /**
```

```
     * 限制的等级，从0到uint256的最大值
     */
    uint256 level;

    /**
     * 条件二进制表达式树向量
     */
    ConditionNode[] conditionNodes;

    /**
     * 如果返回类型为VOTING_NEEDED，则当前插件的投票规则id
     */
    uint256 votingRuleIndex;

    /**
     * 插件注释
     */
    string note;

    /**
     * 指示插件是否启用的布尔值
     */
    bool bIsEnabled;

    /**
     * 指示插件是否被删除的布尔值
     */
    bool bIsInitialized;

    /**
     * 指示插件是操作前插件还是操作后插件的布尔值
     */
    bool bIsBeforeOperation;

}
```

1.4.6 投票规则数组

投票规则数组是存储 DARC 协议中所有投票规则的数组。当程序在沙箱中执行并被插件判断系统检查时，一个或多个操作可能处于待定状态，等待最终投票结果。在投票过程中，所有成员都被允许在有效的投票期间投票。每个投票规则包含启动投票项所需的所有必要要求。

1.5 沙箱

DARC 的沙箱与机器状态管理器的存储设计完全相同，包括代币系统、成员资格、可提取现金、分红、操作历史、插件和投票规则。对于不能被操作前插件批准的每个程序，DARC 机器状态管理器将首先将机器状态克隆到沙箱，然后在沙箱内执行程序，并根据执行结果和沙箱状态做出决定。

1.6 投票状态

在机器状态管理器中，投票状态是管理所有与投票相关属性的独立模块。投票项管理器包含历史上所有的投票项，每个投票项包含等待最终投票结果的程序、累积的投票权力和每个地址的投票记录。