

Application of the Decentralized Network of Services

Table of contents

Introduction	3
Applications	4
<ul style="list-style-type: none">• Application Examples• Task• Rules	
Services	9
<ul style="list-style-type: none">• Receiving tasks from Engine• Send events to Engine• Configuration File	
Engine	11
<ul style="list-style-type: none">• Accessible to all• Developer friendly• Smart proxy	
Network	13
<ul style="list-style-type: none">• Consensus• Side Chain• Clusterization• Parallel execution• Asynchronous executions	
Glossary	15

Introduction

This document outlines the implementation of the decentralized network of services (called MESG Network), previously introduced in the document entitled “Decentralized Network of Services: Migrating traditional technologies towards decentralization”.

The use and functionality of the decentralized network is made possible via MESG Engine, a software which connects the actors in the network and manages the deployment of, and interaction between, services.

Applications

Applications are the main feature of MESH. The goal of applications is to help users and companies to create better solutions and to allow businesses to use blockchain technology without needing to manage the actual complexities.

To keep Applications lightweight with shareable components, Applications on MESH are built to connect events from a technology to tasks on any other technologies.

Applications take the form of a list of tasks which are to be executed once a specific event occurs. All tasks within an Application built with MESH are based on events. Each time a task is completed, it can trigger one or more additional tasks which depend on the result of the previous tasks. This way, the system can remain flexible and simple as an event system.

where every event that contains data will eventually trigger an action. With this in mind, users can create any Application they need with any technology they want. If the technology can send and receive data, it can be integrated and used by MESH.

Applications are triggered when a selected event is triggered. Every time this selected event is triggered, the Workflow within the Application is executed. So, the same Workflow might have multiple executions with potentially different results.

Applications can be private or public. When set to private, only the creator will be able to use it. When they're made public, any user can use it. Because applications have one point of entry with specific data and potentially multiple types of results, it might be used exactly like a Service task. This way, users can use/create reusable Applications. Public Applications are accessible via a Marketplace with the objective of offering as many reusable Applications and Services, allowing basic users to create powerful solutions without the complexities, so they can focus on building their application / business.

Common users can create small Applications that are similar to what can be done with services such as [IFTTT](#) or [Zapier](#). Small companies can use more complex Workflow systems to automate some of their tasks. Developers (possibly from companies) can create advanced Workflows to create full Applications the same way they already do using systems like [Serverless](#) or Backend-as-a-Service systems like [Graph.cool](#) or even [Firebase](#).

Application Examples

Below is a non-exhaustive list of different applications that can be implemented to solve common issues that users, companies and/or developers face:

User wallet creation

- **When** a new user registers on your Application
 - **then** an Ethereum address is created for them
 - **then** this address is added to their contract information
 - **then** some free ERC20 is given to this address
 - **then** an email is sent to notify the user of the completed actions

IoT with an electric car example

- **When** your electric car finishes charging
 - **then** the entity who delivered the electricity is paid
 - **then** the driver is notified that the charge has completed
 - **then** the car is put into standby mode

Image processing (decentralized processing)

- **When** an image is uploaded
 - **then** it is resized
 - **then** it is compressed
 - **then** it is uploaded it to different CDN

Blockchain value transfer

- **When** you receive some Bitcoin
 - **then** the equivalent amount is transferred to Ethereum
 - **then** a notification of the result of the transfer is sent

Ecommerce cart system that accepts payments

- **When** a user adds a new product to their cart
 - **then** a new Ethereum address is created for them
 - **then** wait for the completed payment in Ethereum to this address
 - **then** when the payment has successfully completed, mark the cart as processing
 - **then** send an email to the user
 - **then** send out the product for delivery
 - **then** when the delivery system receives the product, notify the user of the delivery status
 - **then** when the delivery has been made, mark the cart as paid

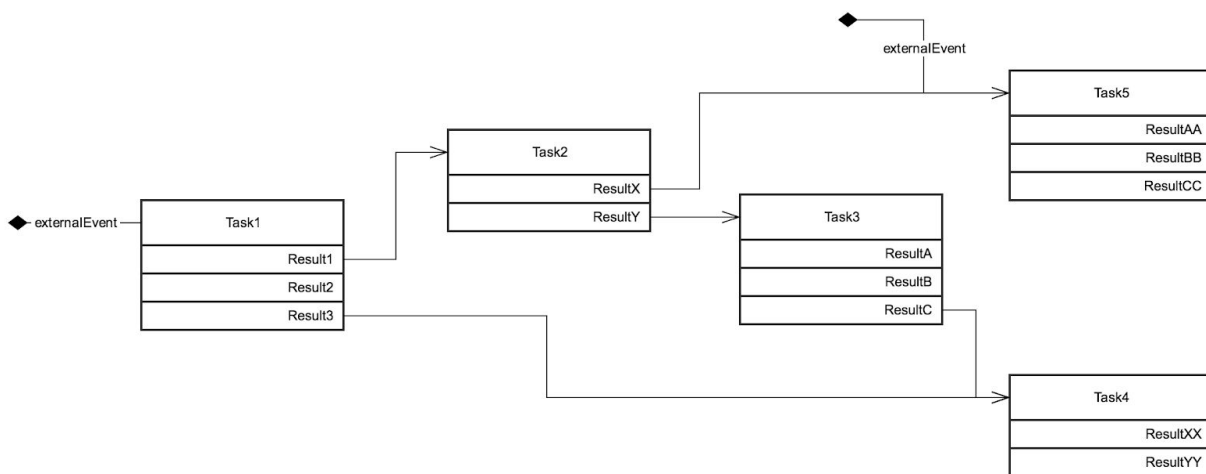
These are merely examples. It's important to understand that any system or technology that has events and actions is compatible with MESG and can be used in applications and workflows.

Task

A task is an action that accepts data as an input, and returns a status with data as an output. A task defines multiple statuses, but only one can be returned during the execution.

Rules

- The first task must be only connected to a single external event
- All subsequent tasks must be connected to a specific status of other task(s)
- Each subsequent task can be connected to one external event



Workflow example

In this example, we have a list of tasks that are connected to each other according to dependencies on the statuses of other tasks. The initial task will only have one external event, but the other tasks have both a dependency on a task, as well as a dependency on up to one external event.

The application first starts with the Task1, then Task1 returns the status Result1, and Task2 is executed. If Task2 returns the status ResultX, Task5 waits for another external event. This is considered one branch of the Workflow, but the Workflow can have multiple branches (there are three in the above example). Every branch will have a status to show whether or not the processing on the branch has completed. If it hasn't yet completed, the workflow has some remaining final tasks to execute.

Services

MESG is based on events and actions on many technologies. In order for MESG to listen to events from Technologies and execute tasks on technologies, a Service must first be created. This Service will act as a bridge between Engine and the technology.

Services have three main functionalities:

- Emit events from the technology it is connected to
- Execute a task related to the technology
- Verify the execution of a task

Services can implement all or any subset of those functionalities.

The more Services exist and the more nodes run them, the better and safer the MESG Network is. In order to reach this goal, different solutions will be created.

Once the Network is deployed, a Service Marketplace will be available within the software, Engine. It will allow Users to easily find and select Services they need for their Workflows and Applications. The Marketplace displays the Service's price, rating, users comments, number of Applications using it, number of nodes running it and the reputation of the developer.

An incentive system for the developer and actors in the network will be implemented. Each time a Service is executed, the developer and actors who participate in the network will have an opportunity to receive revenue for the work they've completed.

An important aspect to the system of Services is to have as many Services as possible running at all times. This is essential to maintain the reliability of the Network and for the security of Services. The more nodes that run a Service, the more reliable and secure it becomes. To encourage this, all users running a Service will receive revenue for every event or task they process.

The goal is to make it very simple for a user to register as a participating node in the Network. This will be accomplished by creating clear UIs and documentation to guide users and provide statistics to help them choose the best services for them. In the near future, the system will be able to automatically optimize the revenue of users and improve the performance and security of the Network.

Receiving tasks from Engine

Services need to be able to receive tasks sent by Engine. Each time a task is received, services will make sure that the sender is Engine, then they will check to ensure that it can handle the task, and if so, it will execute it. Once executed, the service will emit an event to Engine with the result of the task.

Send events to Engine

Services can send data to Engine. The data is divided into two categories:

- The result of a task: When Engine asks to start a task and expects a result from this task
- Events from the listener: When the Service emits a new event from its listener function. (e.g. a running web server receiving a request from a blockchain technology that received a new transaction)

Configuration File

In order for Engine to know all tasks, events and configurations that the Service implements, the entire configuration of the Service is defined within a configuration file.

It's necessary to add metadata like the name, description and constraints on the environment the Service needs to run in. This is useful in helping to choose the right service for your needs.

Engine

To allow people to use the MESH Network, a way must be provided for them to access the different features of MESH. For example, when users want to create an application, visualize the logs of execution, manage services, or create an account / wallet.

To allow these interactions, we created an application called Engine, which is a program that manages complexities for all users. Engine allows different levels of complexity for the different kinds of users.

Accessible to all

In time, most users will not be developers, so they'll need a proper user interface to manage everything from workflow deployment to participation in the network with a low barrier to entry. The idea is to have an easy-to-use graphical user interface where users can deploy an application or start earning tokens by activating services with just a few clicks.

It's important that Engine is easy to set up. A standard user should need to do very little to have Engine be fully functional for them. Also, users that want to install this application on another system, such as a remote server, will be able to do so easily, assisted by documentation that explains how to make your node run on most popular hosting companies.

Developer friendly

The second type of user will be developers or companies that want to have access to both the basic features and the more advanced ones. We will provide these users with a Software Development Kit (SDK) that can be used to access all functionalities. The SDK will be free and accessible to everyone without prerequisites. The SDK will contain two different interfaces: a Command Line Interface (CLI) and an Application Programmable Interface (API). The CLI will be used to access all functionalities of Engine by developers who want to use Engine manually. The API will also provide access to all functionalities so that developers will be able to use it from other software, automate actions on MESH or even create their own products atop MESH.

All of the different parts of Engine are completely open source, meaning that anyone can participate in improving them. Also, anyone can reuse any part of Engine to integrate it in their own technology and/or business.

Smart proxy

The MESG Network is composed entirely of machines running MESG Engine. All communication is done directly between the various copies of Engine, with no direct communication between individual Services and Applications on the Network. Engine connects and manages all Services and Applications in order to relieve Applications of unnecessary complexities.

If an Application requires an execution of a Service that isn't already being ran by Engine, Engine will communicate with other deployed copies of Engine on the Network which are running the desired Service. Engine can ask other machines running Engine to forward an event from a Service which it is not running locally.

All computers running Engine have equal rank in the Network and autonomously make their own decisions. All communication between Engine nodes are first cryptographically verified prior to being processed.

Network

The Network is the communication mechanism which allows Nodes running Engine to freely exchange data. The Network is decentralized without any single points of failure. It's responsible for syncing each Node's data with all of the other Nodes. The Network coordinates all Service events and actions from all Workflows and Applications with the Nodes. The security and trustability of the Network should increase as each new Node is added.

Consensus

The system of trust in the decentralized network will be achieved via a consensus mechanism.

Proof of work is inherently inefficient because resolution is based on high computational usage, and difficulty of resolution increases over time. Moreover, MESG Nodes are already responsible for executing Services and Applications, so we need a lightweight consensus system which is still secure.

MESG will implement a Byzantine Fault Tolerance (BFT) consensus mechanism coupled with a Proof of Stake system. Each staked token will afford one vote in the consensus mechanism. It will be expensive to have a majority of the voting power, resulting in increased security.

Side Chain

Scalability is a big issue for any decentralized network. A large number of Nodes executing or verifying the same task provides security to the network, however it comes at the cost of being rather inefficient and drastically limits the throughput.

Two methods can be implemented within the Network.

Clusterization

Multiple sub-networks can be run so that each can be responsible for doing one specific job. In the case of MESG, these individual sub-networks consist of nodes running the same service.

One root-network will provide security and the most basic functions. The sole purpose could be to keep track of the MESG Token, such as balances and stakes.

Similar systems to this can be seen in products such as Cosmos (hub and zones) or Ardor.

Parallel execution

Workflow executions can run in parallel. This, as opposed to sequential execution, which Ethereum currently uses.

Asynchronous executions

Executions which take a long time to execute shouldn't interfere with executions which can execute quickly, so each execution needs to be treated as asynchronous. In order to do this, a state machine will be implemented for every execution and the changing of states will be the transactions that the network needs to handle (eg: execution created, execution started, execution failed, execution succeed...).

Glossary

Technology

A Technology is anything connected to the Internet that sends or receives data (e.g. Blockchains, IOT, HTTP, API, etc.).

Network

The MESG Network is a decentralized network made up of all nodes running MESG Engine.

Engine

Software that manages all actors in the MESG network, in addition to connections and communications between all applications and services, regardless of the technology or programming language.

Application

A list of Workflows supported by business logic.

Workflow

A list of Services' Tasks executed in a specific order. Workflows are triggered by a predetermined Service's Event.

Service

A list of Tasks and Events from one Technology. Tasks are triggered by Engine to initiate interaction with the technology, and Events happening on a Technology are submitted to Engine. Services are essentially bridges between Engine and Technologies.

Task

A specific action to interact with the Technology. Tasks have inputs and outputs like functions in computer programming. Tasks belong to a single Service.

Event

A message sent by a Technology at a specific time, e.g. a blockchain transaction, a light bulb is turned on, or a user signs up to a website. Events belong to a single Service.