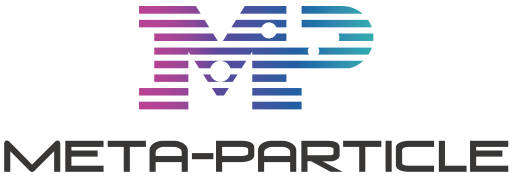
Particle Protocol White Paper V1.0: A protocol for implementing NFT meta-programming



eric
eric@particleprotocol.com
2022.1

Abstract

Particle Protocol intends to build core infrastructure for Metaverse by providing a universal interface that may let any NFTs have interactive capabilities and programming behaviour. The assets built through it support the ERC721[1] specification. In other words, the asset definition of Particle Protocol can be regarded as an NFT similar to encrypted digital artwork. To distinguish it, the NFT defined or generated by Particle Protocol is collectively called Particle. We have expanded the protocol on the basis of ERC721. NFT is not only a static asset, but also a component with dynamic interaction capabilities and program behavior. Digital drawing NFT, music NFT, game props NFT, avatar NFT, Loot NFT, and so on. These are the raw materials needed in the formation of the Metaverse, like reinforced concrete in architecture, and elements in chemistry. They need to be transformed, distorted, synthesized and split to create a real world. Only when two hydrogen elements and one oxygen element are synthesized together in a certain way can they become water and give birth to life. Life is an important aspect that the Metaverse must possess. Particle Protocol believes that interactive capabilities and program behavior can also be ERC721, making program behavior part of NFT, which can not only exert strong distortion ability on NFT, but also can be arbitrarily transferred. Users can claim or purchase the Particles they need, then choose the raw material NFT they already owned, and then they can start their own creations full of infinite imagination.

Contents

1 Particle Protocol Core

1.1 Design objective

The community, nowadays, has developed many projects and tabled plenty of EIPs relating to NFTs, including EIP721[1], EIP1155[2], EIP998[3], EIP1948[4], EIP2981[5], EIP3386[6], EIP3440[7], EIP3569[8], EIP3589[9], and EIP3664[10]. It is not difficult to find that other EIPs are basically for supplementing some customized interface functions except EIP721. For example, EIP1155 and EIP998 provide batch processing capabilities; EIP1948 expands dynamic data storage capabilities under existing standards; EIP2981 increases the interface function of NFT royalties; EIP3440 adds author's signature; EIP3569 offers an operation interface of NFT metadata sealing. These EIPs, or customized interface functions, are very fragmented. With the development of NFT, or with the development of Metaverse, there will inevitably be more EIPs for NFT. If each featured interface function tables an EIP, it may increase the cost of use for developers and users and become less comprehensible.

In fact, from a technical point of view, the purpose of adding various interface functions is to process NFTs with the EIP721 standard. From this perspective, we can consider all existing NFTs as raw materials and consider other interface functions as processing tools. These processing tools are essentially programs or scripts, and they should be diversified.

After analyzing products on the market, we realize that most NFT products are image-based and multimedia, such as music and a small animation and are relatively static and isolated. Therefore, these NFTs cannot add new elements, extend functions, synthesize and split, and of course, interact with other NFTs. This situation is like various atoms increasingly appearing in the universe, but these atoms cannot interact. They cannot combine into molecules and synthesize into objects. Thus, they are entirely isolated.

Suppose the static NFT gives people the imagination of design. In that case, creating the script can make the NFT have dynamic capabilities, and those interactive NFTs can create the NFT applet, which is the synthesis application. We believe everything can be Particle NFT. For example, the image is Particle NFT, the text is Particle NFT, the code is Particle NFT, the DeFi operation is Particle NFT, and the interface is Particle NFT. Therefore, we must make the NFT specification less fragmented and have infinitely scalable interaction capabilities through mutual synthesis. Our goal is to let NFTs become programmable so that some NFTs with black and white characters like Loot can also become colourful, thus adding a new dimension to the entire world of NFTs.

1.2 NFT Meta-Programming

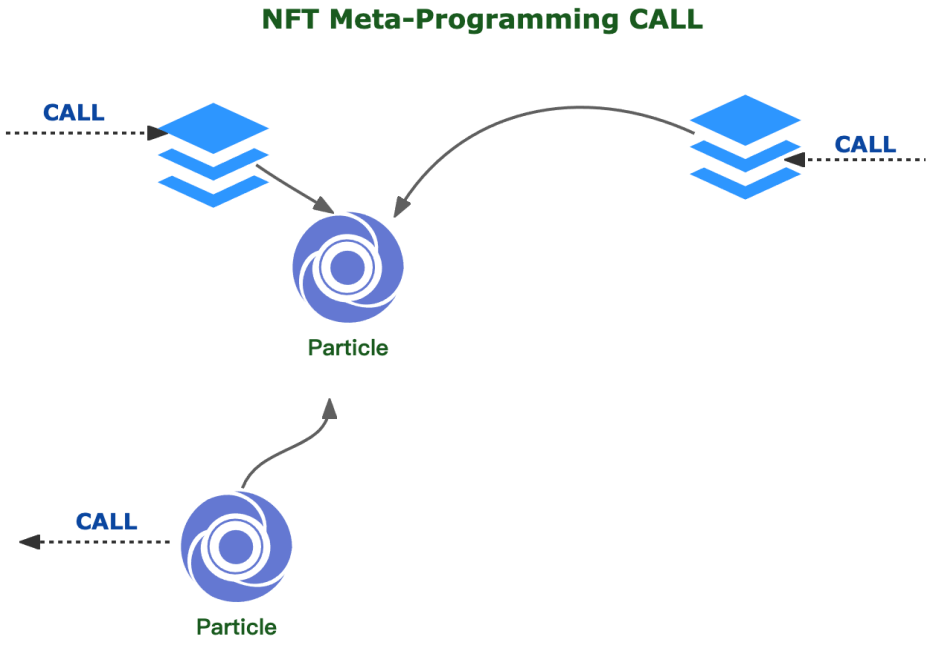
The definition of NFT, or the market's understanding of NFT, is a static resource or asset such as an image, a piece of music, a small animation, or some rights. Developing smart contracts or external programs is needed if you want to use these static resources or assets. Therefore, we cannot distribute the NFT processing program well because they are unrelated and cannot wholly perform advanced operations such as synthesizing and disaggregating with NFT resources and processing programs. Just a bit like a variable in a programming language which variables define data types and processes that perform on types. For example, integer variables represent integer data types and define operations like addition, subtraction, multiplication, and division. Those variables are a complete whole and are indivisible; otherwise, we are unable to program.

In the EIP721 standard, there are only interface functions such as transfer and approval within smart contracts and are no processing methods for Metadata which is the core of NFT. Without Metadata, NFT can be meaningless. For example, game props NFTs need to describe level, colour, shape and even 3D object structure in Metadata; music NFTs need to describe duration, author and cover in Metadata. Therefore, our processing of NFT is essentially processing Metadata, and the smart contract only records some summary information and ownership.

The goal of Particle Protocol is to achieve the programming ability of Metadata, that is, NFT Meta-Programming. Through NFT Meta-Programming, we can process NFT resources any way we want. For example, we can flip, arrange, and stack image NFTs, synthesize music NFT with a music player NFT, aggregate DeFi operations into an NFT, and even add a UI NFT to create our unique DeFi applet.

Imagine that we need to create a singing doge NFT. At this time, we first need to prepare some raw materials, such as a doge image NFT, a piece of music NFT and a particular UI NFT. Assuming these NFTs are programmable and we can synthesize them according to the pre-defined procedure. Therefore, we can synthesize the dog image NFT with the music NFT and then synthesize it with the UI NFT. At last, we will get an NFT with a cute doge, and when we click on its belly, it begins to sing. And that is NFT Meta-Programming capability.

It is very flexible to call NFT Meta-Programming, as shown in the following figure:



After users have Meta Particle NFTs, they can write a program by calling the script in Particle NFTs they already owned. The calling process needs to pass the authentication of the owner's private key. The script written by the user can be a purely external program, or it can be an independent Particle NFT. In this way, we can realize the distribution of works of NFT meta-programming and calling Particle is just like calling a program library. Thus, we can implement sealable, capitalized, and authentication-capable program calls.

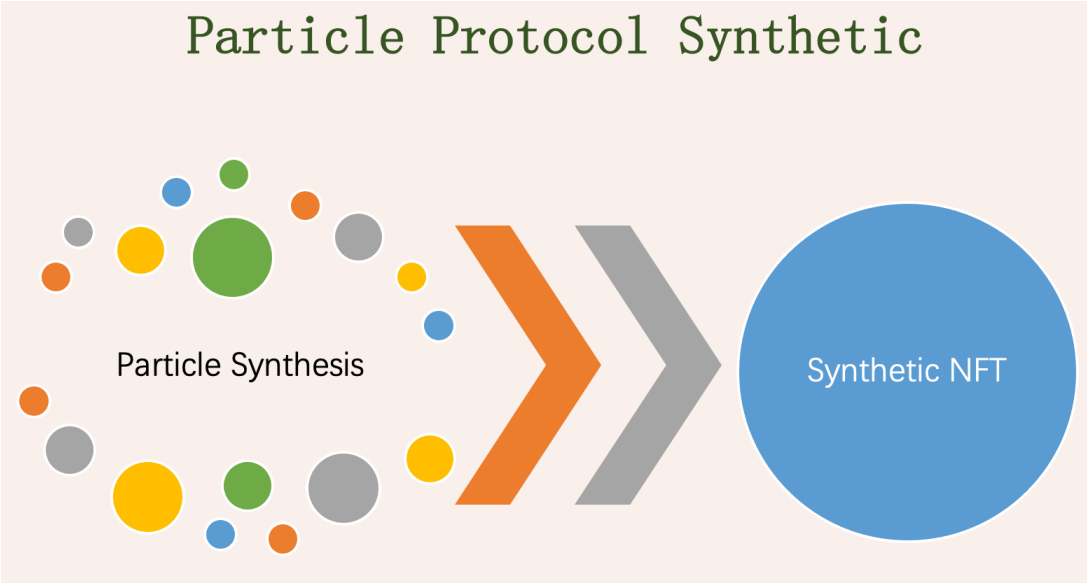
1.3 NFT As a Service

When NFT has the ability of program interaction,, a new world will be opened. We can realize almost completely third-party server free applications, and you will see that this method is still very environmentally friendly. In the current DAPP development process, smart contracts can be directly deployed to the blockchain, which is very convenient and can enjoy the decentralization of the blockchain network. However, as an app, it is often necessary to provide a web service, which is usually deployed to AWS or other cloud servers.First of all, the deployment process is troublesome. You have to buy a server, configure the environment, upload the service program, and run the application. Second, it is hard to distribute this web service efficiently. For example, when you need to transfer the Web service to others for maintenance, You have to change the key, change the password, and sometimes even change the mobile phone number. If multiple people maintain the components, it almost requires a professional team.

Furthermore, running this web service in different environments is inefficient, which requires plenty of technical support, such as running a demonstration in an intranet or migrating to another heterogeneous network. Finally, whether the web service is temporary or permanent, and whether hundreds or thousands of people use the web service, we must design a load balancing architecture to support various maintenance tools and backups. This web service keeps running 24 hours a day which is eco-unfriendly, heavy and inefficient.

When NFT is programmable, NFT itself can be a service, for example, a Web service, or any other form of service, which can run uninterruptedly or temporarily start up and run, and be distributed to anyone needed. All we have to do is to transfer it once. Don't forget that the NFT with service capabilities is an asset and can be traded and transferred arbitrarily in the market.

A schematic plot for NAAS (NFT As A Service):



Users can encapsulate various program services by synthesizing Particle NFTs, distributed as an independent service or a synthesis unit aggregating multiple program services. The user owns the right to use the service program and has ownership that truly owns this service program. This provides finer granularity encapsulation capability for rights distribution and collaborative design of various service programs and also convenient for users to perform peer-to-peer distribution.

1.4 Asset Oriented Script Distribution

Particle Protocol provides a new software component distribution method called Asset Oriented Script Distribution, referred to as AOSD. We know that every NFT is an asset, and with Particle protocol, every NFT can also carry programs. Therefore, every program held by NFT is also an asset. Through this synthetic method, we can achieve program distribution at any granularity. For example, we can distribute a player UI, a sorting algorithm program, a layer synthesis tool and, of course, complex software composed of various small programs.

This distribution process completes the transfer of ownership and right of use of a program asset. Users can claim or purchase corresponding NFTs with programs according to their needs and transfer them to each other. For instance, A can transfer one of his own Defi NFTs to B, B can transfer his own music player NFT to C, and C can continue his synthetical creation. Using this distribution method, users only need to manage assets through their private key to receive and transfer or use the Particle Protocol dedicated client to complete the asset-oriented, peer-to-peer distribution.

1.5 User Defined Metaverse

The NFT meta-programming capabilities implemented by Particle Protocol can release the massive productivity of the community. However, the Metaverse requires enormous content and huge participants to build, including artists, programmers, modellers, product designers, and innovators to participate in many roles to provide content together.

Take games as an example. Traditionally, a team or a company design and develop a game, and players are only the game users or experiencers. Even if in Gamefi, we can trade or transfer props in the form of NFT in the blockchain network instead of participating in constructing the game or changing the game's elements. For example, players cannot synthesize names or logos to props unless the game itself supports it.

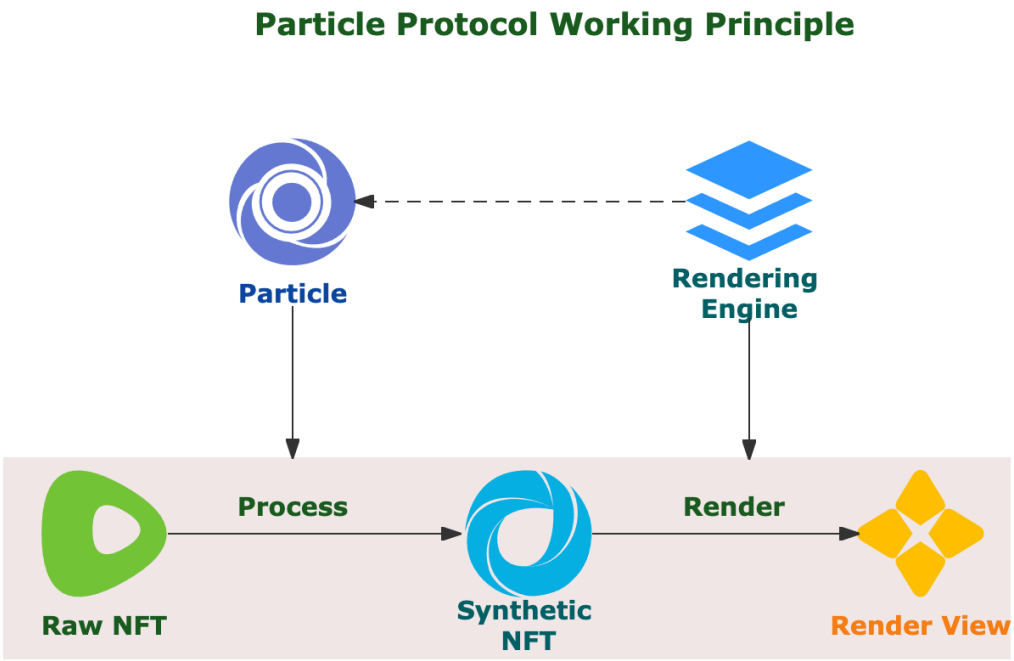
At the initial stage of GameFi design, props can be minted as NFT in the blockchain network, and we can trade and collateralize those NFTs through DeFi facilities. However, these are independent operations on props, and players cannot use those props back in the game after recreating them regardless of any forms of processing. For example, there is a horse prop, can a user continue using it after changing its fur colour from black to white? It is usually impossible because the horse NFT is just an image or a 3D object which is not programmable, and the game cannot recognize it after synthesizing the NFT.

Particle Protocol allows users to participate in constructing games. For example, a Loot holder can synthesize their Loot NFT with a 3D-prop-UI Particle NFT and join a game. Thus, this game is no longer designed and developed by a single fixed team but a decentralized and collaborative community. While developing a game, each user can create their script components through Particle Protocol, and multiple users can also collaborate. From music to props, from scenes to plots, each part is contributed by the user. We synthesize them piece by piece, layer by layer and finally complete a user-defined game.

In this way, we call it UGM, which is stands for User-Generated Metaverse.

1.6 Working Principle

Particle Protocol is a protocol specification that defines the interface standards and data formats that allow NFTs to become programmable. Particle itself complies with EIP721, which makes every Particle a standard NFT. If we want to extend the interface functions, we do not need to define a standard immediately. Instead, we can implement it by horizontally synthesizing the EIP721 standard. In Particle Protocol, we consider image, music and other NFTs as raw materials. We can generate new NFTs by processing with NFTs that carry scripts according to Particle specifications. The following figure shows the basic workflows:



As shown in the figure, after the Particle completes the processing of raw NFT and generates a synthetic NFT, the rendering engine then processes that NFT to present a rendered view with features. The rendering engine deals with the presentation layer through the rendering script in the Particle from the above chart, and it runs in a browser or a mobile App to render and display the synthesized NFT. Furthermore, the script of the rendering engine is synthesized from Particle. Moreover, Particle is divided into multiple types to deal with processing different resource types. For example, image flipping, colouring and layout; Defi trading, lending and mining; music, short video, advertising and media. All of them provide a variety of direct-oriented raw materials NFT programmable processing. In addition, some provide configuration and service capabilities, such as commission, subscription and authorization.

2 Core Principles

2.1 Metadata Specification

The Particle Protocol standard includes ERC721, and the standard interface methods are all consistent. The Meta Particle standard does not define a unique interface standard but implements new script functions through the extended definition of metadata. The metadata format is defined as follows:

NO.	Field name	Description
1	name	Name of the item
2	image	Raw image data，supports png、svg、jpg
3	external_url	External link to the original website for the item
4	description	A human readable description of the item. Markdown is supported
5	attributes	These are the attributes for the item
6	animation_url	A URL to a multi-media attachment for the item
7	youTube_url	A URL to a YouTube video
8	scripts	Particle Scripts

The first 7 fields are compatible with OpenSea, and the last field is defined to support the implementation of script functions.In the script field, contract scripts and rendering scripts can be defined to realize the program capability and rendering behavior of particle respectively. Different types of particles have different definitions of scripts.

2.2 Definition of Script

The script in Particle is implemented using JavaScript so that the execution environment of the script can be as universal as possible. The definition of the script is the definition of a set of methods described in JSON format, and the content of the function method is base64 encoding to avoid unknown encoding escapes in script programs. In the definition of Particle, there are two main script definitions. One is the rendering script, and the other is the contract script. Both types of scripts are defined in Metadata. The rendering script is used to describe how the synthesized NFT will be parsed and called, and the contract script defines the program behaviour in the Particle. The parsing of these two scripts requires an externally defined SDK, which we call Particle.js.

2.2.1 Rendering Script

The role of the rendering script is mainly to provide the calling specification of the external presentation layer for the NFT synthesized with Particle Protocol. When Particle Protocol synthesizes an ordinary raw NFT, new action behaviours will be bound, and the displayed content may also change. Therefore, to inform the external program how to express the synthesized NFT, a set of scripts needs to be specified as a parsing reference. The format of the rendering script is defined as follows:

```
{
    "script_type":"",
    "max_limit":,
    "limit_address":[],
    "inputs":{
        "type":,
        "name":""
    },
    "outputs":[
        ""
    ],
    "function":""
}
```

The structure is composed of three sections: `inputs,` `outputs` and `function,` and the definition are as follows

1. `inputs`
    - **Definition:** Represents the passed in rendering parameters. The rendering script passes the Metadata of the raw material NFT or the field data in the Metadata. By default, Particle.js uses the format of an array to process. Therefore, rendering parameters are divided into `type` and `name.`
    - `type` is divided into 4 encoding types, as follows:
        - Type number：`0`
            - Definition：It means to merge the Metadata of the NFT, which is converted into a JSON string by default and passed into the `function.`
            - Example：

```
{
    "inputs":{
        "type":0,
        "name":"metaData"
    }
}
```

        - Type number：`1`
            - Definition：It means to merge the field data in the Metadata of NFT, convert it into Base64 encoding format and pass it into the `function` by default. The field name contains: **image**, **animation_url**, **youtube_url**, generally used for image processing.

- Example：

```
{
    "inputs":{
        "type":1,
        "name":"image"
    }
}
```

- Type number：`2`
  - Definition：It means to merge the field string in the Metadata of NFT, and the field string is passed into the `function.`
  - Example：

```
{
    "inputs":[{
        "type":2,
        "name":"attributes"
    }]
}
```

- Type number：`3`
  - Definition： It means custom parameters. If there are multiple custom parameters, they will be rendered in the default display form on the front end.
  - Example：

```
{
    "inputs":[
        {
            "type":3,
            "name":"type"
        }
    ]
}
```

2. `outputs`， represents the output value, displayed in the form of an array containing the object name, returned the data object by default, and the output is based on the object name.
3. `function`， represents rendering execution code, using Base64 encoding format.

The above is the format definition of the rendering script. In the actual processes, the passing Metadata parameters will be rendered into the function using the template engine, and different Particle types will use various processing methods. Take a left and right flip of the image as an example. The sample code is as follows:

```
let content = `
    let images= params[0][0];
    function flip_left_right(image) {
            let that = this;
            var canvas = document.createElement("canvas"), context =
canvas.getContext("2d");
            canvas.style = "position: absolute; left:-200%;top:-200%";
            canvas.id = "flip_left_right"
            var obj_image = new Image();
            obj_image.src = image;
            let index =  image.indexOf('base64')
            let file_type_text = image.slice(0,index)
            document.body.appendChild(canvas)
            obj_image.setAttribute("crossOrigin", "");
            return new Promise ((resolve,reject) => {
                obj_image.onload = function () {
                obj_image.width = file_type_text.indexOf('svg')==-1?
obj_image.width:obj_image.width*8;
                obj_image.height = file_type_text.indexOf('svg')==-1?
obj_image.height:obj_image.height*8
                canvas.width =  obj_image.width;
                canvas.height =  obj_image.height;
                context.scale(-1, 1);
                context.translate(-canvas.width, 0);
                context.drawImage(obj_image, 0, 0);
                let ImageData = context.getImageData(0, 0, canvas.width,
canvas.height);
                let img_type =
file_type_text.indexOf('jpeg')>1?'image/jpeg':'image/png';
                let image_URl = canvas.toDataURL(img_type);
                if(document.getElementById('flip_left_right')){

 document.body.removeChild(document.getElementById('flip_left_right'))
                }
                    resolve(image_URl);
                },
                obj_image.onerror=() =>{
                reject(new Error('urlToBase64 error'));
                }
            })
        }
    async function getURL(){
       let data={};
       data.url=[];
       for(var i = 0; i<images.length;i++){
         await flip_left_right(images[i]).then(res=>{
            data.url.push(res)
          })
       }
```

```
        return data
    }
    return getURL();
`;

var image = '……';
let fun_value = [[image]];
var AsyncFunction = Object.getPrototypeOf(async function () {}).constructor;
let fun = new AsyncFunction('params', content);
fun(fun_value).then(console.log);
```

As shown in the above code, we define a javascript function method for the left and right flip of the image, then render the image as a parameter to the processing function, and return a new image object after processing. There are some differences in the function processing methods of different Particle types. For example, the return value will be different. It is not necessarily the value type that returns an image, but the overall structure definition and calling method are the same.

2.2.2 Contract Script

The reason why Particle can process other NFTs is achieved through its scripting ability. This script is called a contract script. Like the rendering script, the contract script will also include input, output and function. The format is defined as follows:

```
{
    "script_type":"",
    "max_limit":,
    "limit_address":[],
    "inputs":[
        {
            "type":"uint8",
            "name":"type"
        },
        {
            "type":"address",
            "name":"erc20Address"
        },
        {
            "type":"address",
            "name":"from"
        },
        {
            "type":"address",
            "name":"to"
        },
        {
            "type":"uint256",
            "name":"amount"
```

```
        }
    ],
    "outputs":[
        "value1",
        "value2"
    ],
    "function":""
}
```

The structure is divided into `inputs,` `outputs,` and `function.`

1. `inputs`

   Represents the passed in contract parameters, divided into `type` and `name.` The `type` is the Solidity language type, and the `name` is the parameter name. The passing data will be combined into an array according to the sequence. For example, Particle.js will pass the `ether` parameter by default, put it in the first array, and pass it to the `function`.

2. `outputs`
   Outputs represent the output data displayed in an array containing the object name, returns the data object by default, and the output is based on the object name.

3. `function`

   Represents the contract execution source code using the Base64 encoding format. When executing the contract script, it needs to call the client plug-in for managing private keys. Thus, `function` needs to inject a plug-in for managing private keys by default. When operating Particle on the front-end, the contract script can be executed unless there is data. The page defaults to display the contract data in `inputs,` performed using JavaScript's `Function` constructor. If `outputs` has a value, it will display on the page according to parameters after execution by default.

Let's look at an example of a contract script, taking the balance query as an example, as follows:

```
async function func1() {
    let scriptString = `
        let ethers = params[0];
        let type = params[1];
        let erc20Address = params[2];
        let address = params[3];

        if (type == 0) {
            async function getBalance() {
                var provider = new ethers.providers.Web3Provider(window.ethereum);
                let balance = await provider.getBalance(address)
                let data = {'amount': balance._hex};
                return data;
            }
            return getBalance();
        } else {
            async function getBalance() {
```

```
                let erc20Abi = [{
                    "inputs": [
                        {
                            "internalType": "address",
                            "name": "",
                            "type": "address"
                        }
                    ],
                    "name": "balanceOf",
                    "outputs": [
                        {
                            "internalType": "uint256",
                            "name": "",
                            "type": "uint256"
                        }
                    ],
                    "stateMutability": "view",
                    "type": "function"
                }];

                var provider = new ethers.providers.Web3Provider(window.ethereum);
                var constract = new ethers.Contract(erc20Address, erc20Abi,
provider.getSigner());
                let res = await constract.balanceOf(address);

                let data = {'amount': res._hex};
                return data;
            }
            return getBalance();
        }`;

    let en = base64.encode(scriptString);
    let de = base64.decode(en);

    var AsyncFunction = Object.getPrototypeOf(async function () {}).constructor;
    const fun = new AsyncFunction('params', bbbb);
    let arrPa = [ethers, 0, "", ""];
    let val = await fun(arrPa);
    console.log("--------", val);
}
```

As shown in the above code, the calling method is the same as the rendering script. After the getBalance script is defined in a Particle, it has the ability to query the balance of the address. Of course, the UI part is not defined in the above script definition, apart from the getBalance function method. If you want this balance query to have a user-friendly interface, you can continue the synthesis with a UI-type Particle NFT.

2.3 Metadata Template

The above content describes the field specifications of metadata and the definition of rendering scripts and contract scripts. Combine these to get a complete metadata template format, as shown below:

```
{
    "name":"",
    "image":"",
    "description":"",
    "external_url":"",
    "animation_url":"",
    "attributes":[

    ],
    "youtube_url":"",
    "scripts":[
        {
            "script_type":"",
            "max_limit":,
            "limit_address":[
                ""
            ],
            "inputs":{
                "type":,
                "name":""
            },
            "outputs":[
                ""
            ],
            "function":""
        }
    ]
}
```

As you can see, the metadata template format of Particle is compatible with ERC721, and it is also compatible with OpenSea extended fields. Furthermore, we choose extension fields to implement new functions instead of redefining all formats, which can reduce the complexity of user learning, improve the compatibility of product specifications and reduce the fragmentation of the specifications as much as possible.The fields in metadata are described below:

Field name	Description
script_type	Script type
max_limit	Maximum synthesis quantity. The default synthesis quantity must be greater than 0 and less than or equal to max_ Limit, - 1 means there is no limit on the quantity.
limit_address	Limit synthetic NFT address. This is an array type. Check whether it is the NFT address in the array during synthesis. It can be empty. If it is empty, it means there is no limit.
inputs	Input parameters in script
outputs	Output parameters in script
function	Script program

2.4 Contract Interface

Particle Protocol defines a composite interface and demonstrates a standard implementation.As shown below:

1. **IParticle**

```
// SPDX-License-Identifier: agpl-3.0

pragma solidity 0.8.0;

import {IERC721} from './IERC721.sol';

/**
 * @dev IParticle Interface
 */
interface IParticle is IERC721 {

    struct Signature {
        address creator;
        bytes sigMessage;
    }

    /// @notice Creator address and signature cannot be changed
    /// @param _tokenId The identifier for a Particle
    /// @return The creater address and signature
    function getSigned(uint256 _tokenId) external view returns (address, bytes
memory);

    /// @notice Verify that the script is signed
    /// @param _tokenId The identifier for a Particle
    /// @param _message Hash of scripts in metadata
```

```
    /// @return True if `_message` is signed by creator, false otherwise
    function isSigned(uint256 _tokenId, bytes32 _message) external view returns
(bool);
}
```

All particle contracts need to inherit IParticle. This interface contract also inherits the IERC721 contract. Therefore, it can be considered that the particle contract is actually the ERC721 contract. Through the code, we can see that IParticle Contract add one structure definition and two methods. It is mainly used to record the creator's signature verification of scripts in metadata.

2. **IPeripheral**

```
// SPDX-License-Identifier: agpl-3.0

pragma solidity 0.8.0;

import {IParticle} from './IParticle.sol';

interface IPeripheral is IParticle {

    // Raw material NFT for merging
    struct Childs {
        // ERC721 address array
        address[] ntfs;
        // TokenId array of ERC721 address
        uint256[] tokenIds;
    }

    /// @dev Emitted when `merge` and `split`.
    /// @param _owner owner of contract
    /// @param _tokenId tokenid of merged NFT
    /// @param _uri URI of merged NFT
    /// @param _nfts ERC721 contract address array to be synthesized
    /// @param _tokenIds TokenId array of ERC721 contract to be synthesized
    /// @param _operation_type Action type, 0 means synthesis, 1 means split
    event Operation(address indexed _owner,
      uint256 indexed _tokenId,
      string _uri,
      address[] _nfts,
      uint256[] _tokenIds,
      uint256 indexed _operation_type
    );

    /// @dev  Emitted when `seal`
    /// @param _owner owner of contract
    /// @param _tokenId TokenId of NFT
    event Seal(address indexed _owner, uint256 indexed _tokenId);

    /// @dev Emitted when `withdraw`
```

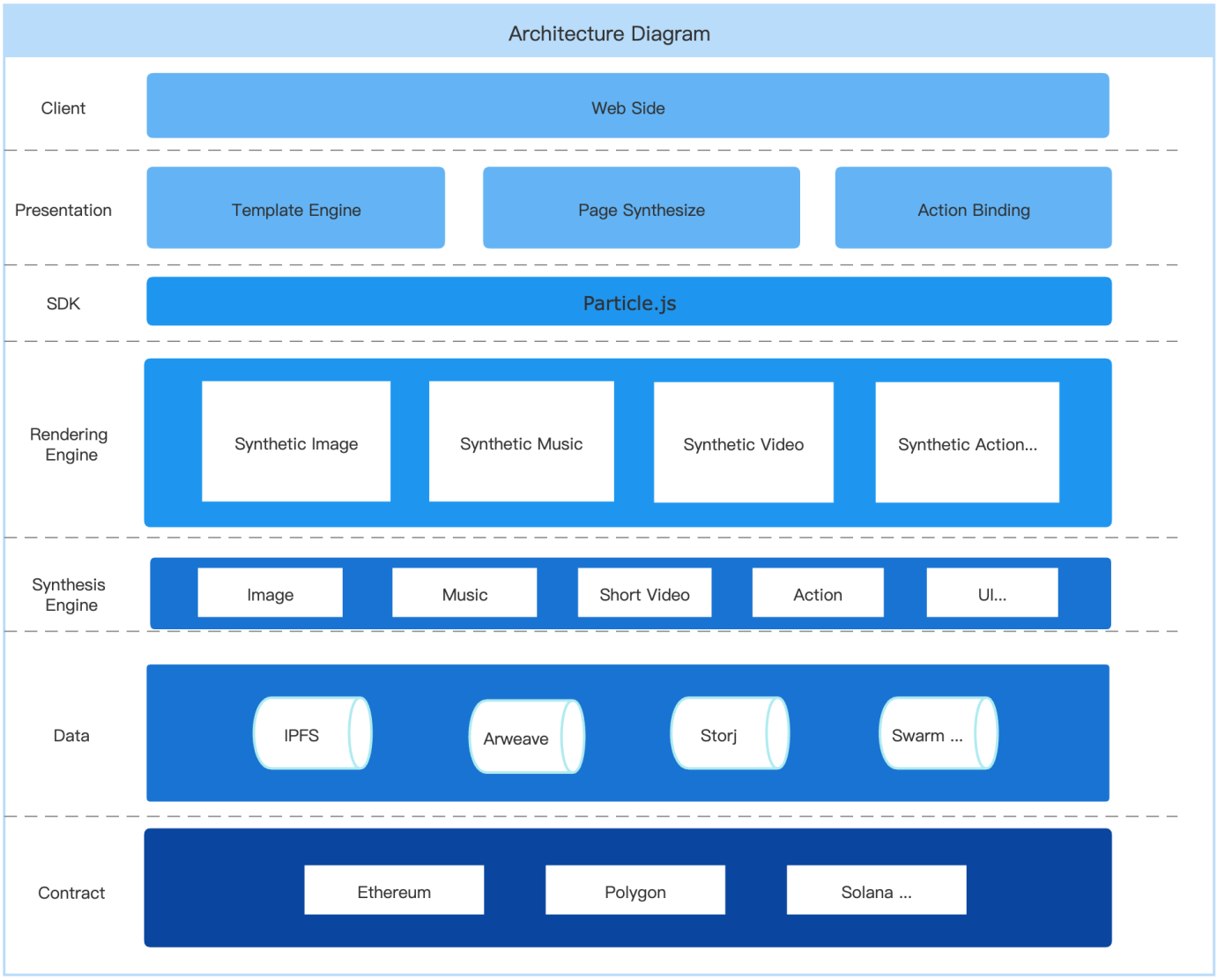
```
    /// @param _to Target address withdrawn to
    event Withdraw(address _to);

    /// @dev Return the sealed state of the NFT
    /// @param _tokenId TokenId of NFT
    /// @return True if `_tokenId` is sealed, false otherwise
    function isSealed(uint256 _tokenId) external view returns (bool);

    /// @dev Return the contract address array and tokenid array of child NFT
    /// @param _tokenId TokenId of synthetic NFT
    /// @return Contract address and tokenId of child NFT
    function getChild(
        uint256 _tokenId
    ) external view returns (
        address[] memory nfts,
        uint256[] memory tokenIds
    );

    /// @dev Return the reuse flag through 'tokenid'
    /// @param _tokenId TokenId of NFT
    /// @return True if `_tokenId` is used, false otherwise
    function isUsed(uint256 _tokenId) external view returns (bool);

    /// @dev Seal `tokenId`,Emits a {Seal} event
    /// If the NFT has been sealed, it can no longer be split
    /// @param _tokenId TokenId of NFT
    function seal(uint256 _tokenId) external;

    /// @dev Merge entry method of particle
    /// Emits a {Operation} event
    /// @param _tokenId TokenId of synthetic NFT
    /// @param _uri URI of synthetic NFT
    /// @param _nfts Contract address array of raw material NFT
    /// @param _tokenIds TokenId array of raw material NFT
    function merge(
        uint256 _tokenId,
        string calldata _uri,
        address[] calldata _nfts,
        uint256[] calldata _tokenIds
    ) external payable;

    /// @dev Split synthesized NFT
    /// Emits a {Operation} event.
    /// @param _tokenId TokenId of NFT to be split
    /// @param _to Target address transferred in when splitting
    function split(
        uint256 _tokenId,
        address _to
    ) external;
```

```
    /// @dev Return royalties
    function royaltyInfo() external pure returns (uint256);

    /// @dev Withdraw contract eth balance to address ` to '
    /// Emits a {Withdraw} event.
    function withdraw(address _to) external;

}
```

IPeripheral is the peripheral contract of IParticle, which is used to realize the synthesis and splitting of NFT. It also supports the sealing operation. The sealed synthetic NFT can no longer be split.If the creator sets royalties in the particle contract, the creator can receive royalties in the future transactions of the particle.

2.5 Architecture Composition

The implementation of the Particle protocol includes multiple layers of components, from the achievement of the contract layer code to the synthesis engine and rendering engine, which completes from creating the Particle to call and display. The synthesis engine is used explicitly for NFT synthesis operations, which defines a set of interface functions, and calls the contract script defined in Particle. The synthesis engine supports synthesizing and disaggregating between different NFTs and the setting of action parameters. The synthesis engine is a significant and fundamental entry in the Particle Protocol product. No matter how complex the synthesis works and the script design, the synthesis engine is ultimately used to complete each entry operation. Let's look at the structure of the entire protocol:



As shown in the figure above, we will briefly explain:

- Contract layer
  Particle Protocol can be deployed in various blockchain networks, such as Ethereum, Polygon, and Solana. Furthermore, as an application protocol, it can be implemented in different underlying networks. Thus, users may comprehensively consider various factors in actual use, such as gas fees and the abundance of raw NFT materials. At present, a batch of Metaverse public chains, such as flow, chr, and wxap, provide more platform foundations for the ecological construction of the Particle protocol.
- Data layer
  From the data layer, we can see that the Particle protocol supports various peer-to-peer storage networks. To support the implementation of large-scale complex scripts, reduce the gas fee and storage cost, we make full use of the more affordable decentralized storage networks, such as IPFS and Storj. Particle Protocol will continue to support other storage networks in the future if needed.
- The Synthesis engine
  For different raw NFTs, such as images, music, short videos and UI, the synthesis engine completes the synthesis of various functions and the processing of raw materials. The synthesis engine has subtle differences in accomplishing different types of raw materials, such as parameter passing methods and returning values.

- The Rendering engine

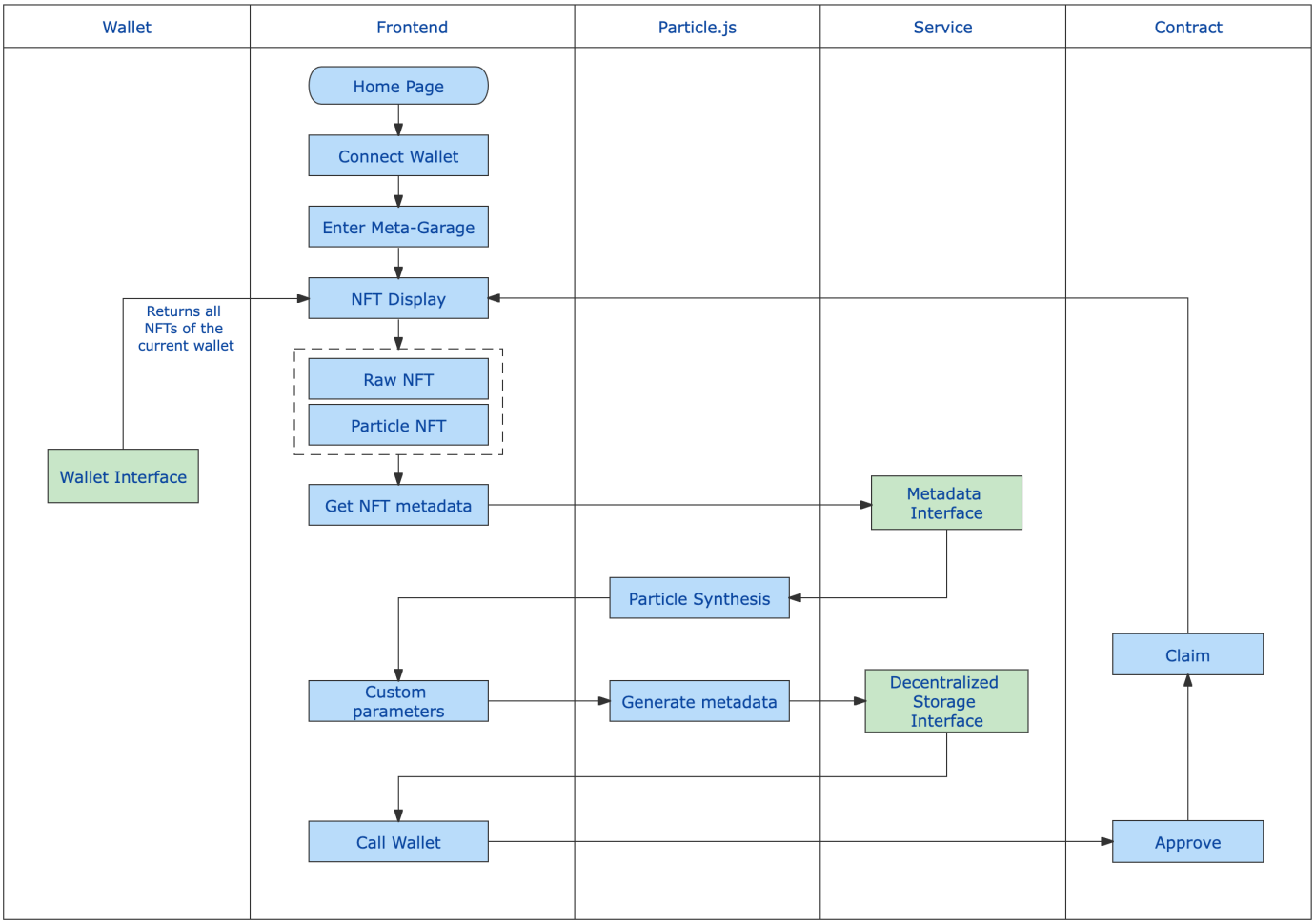
  The synthesized NFT redefines the display method and data, and each Synthetic type has different methods and data sources. Thus, the synthesized NFT can be displayed well through the rendering engine.Like the synthesis engine, the rendering engine has different rendering methods according to synthetic types. Therefore, the rendering engine is mainly used to render and display Particle NFTs and synthesized NFTs. When Particle Protocol synthesizes various NFTs through scripts, there are no technical restrictions on the number and resources, such as images with music and DeFi with UI. Thus, users can use their imagination to create various creations. Furthermore, the rendering engine is not just a straightforward display. For example, with complex synthesized NFT, the rendering engine will perform metadata routing and positioning and loads and renders according to the established order rules.

- SDK
  Particle.js is a development kit provided to users as external calls, used for calling synthesis, rendering, and method defined in Particle. It is an interactive tool for external programs and Particle protocol.

- Display Layer
  The display layer defines the specific display implementation of the synthesized NFT, mainly using the JS template engine, which merges the page and the action to final display in the WEB container. Note that the Particle protocol definition does not limit the container type of the display layer. The Web container is used as the basis of the display to be compatible with the operating environment of most current systems. After all, no system does not support Web containers.

- Client
  This is where users are directly used. The final product mainly includes website applications and mobile applications, among which the mobile applications are primarily realized in the form of Web App.

2.6 Working Process



As shown in the figure above, it shows the production workflow of Particle Protocol and demonstrates the working process of Particle Protocol. In addition, there is a `Claim` method in the figure, which is the synthesis entry method of Particle Protocol. Users connect their wallets and select their own particle NFT and other NFTs. The synthesis engine will perform authentication verification, obtain the metadata of particle NFT, call the script in particle NFT to synthesize the raw NFT and generate a new NFT.

3 Particle Type

Particle can be divided into multiple types according to different resource processing objects and the functions provided. Different types have their unique parameters and return values, as well as different script processing logic.

3.1 Type List

The primary type of Particle is mainly to realize various operations on NFT resources, which are roughly divided into the following sub-types:

1. Transfer
    - Create the transfer function of the wallet as particles.
    - Any type of asset transfer function can be created as a particle independently or in aggregate.
2. DeFi
    - Split the operation function of the defi application and create it as an independent particle.
    - Aggregate the operation functions of the defi application and create it as an independent particle.
    - Particle supports calling wallets such as metamask.
    - Particle supports wallet function calls that have been created as a particle type.
3. Advertisement
    - Support static NFT synthetic advertising through images, music, text, etc.
    - Support 3D synthetic advertising NFT.
    - Support synthetic interactive operation and realize user graffiti and pet advertising NFT.
4. Chat
    - Create P2P based chat as particle.
    - The particle chat component with different functions can be synthesized into functions similar to twitter / Facebook.
5. Magic
    - Support image NFT flipping, timeline animation synthesis and particle effect synthesis.
    - Support four grid cartoon NFT synthesis.
    - Support background color change of text and image NFT.
    - It supports mirror reflection of any text and image NFT.
6. Music
    - Support the synthesis of music and UI into player.
    - Support synthetic music station service.
7. Short Video
    - Support multiple short video synthesis.
    - Support video and text synthesis.
    - Support user operation interactive synthesis of video.
8. Wallet
    - Create a view of the asset as a particle.
    - Create the transfer of assets as a particle.
    - Create the UI of the wallet as particle.
9. Cross Chain

    - Support cross chain asset transfer without third-party services.
    - Support synthesis of cross chain applications.
10. Permission
    - Create DAO governance permissions as particles.
    - Create token permissions as particles.
11. Distribution
    - Support the synthesis of asset NFTs into particles that support batch distribution.
12. Topic
    - Supports the creation of content topic particles.
    - Supports creating articles for newsgroup topics.
    - A topic type particle can be received by subscription.
13. Subscription
    - You can subscribe to any topic using a subscription type particle.
    - Each type of subscription particle corresponds to a topic.
14. Media
    - Supports the creation of media particles with collage content.
    - Media particle can be synthesized into Topic Particle.
15. Game
    - Supports the creation of game particles with instant start services.
    - Support particle creation of user-defined games.
16. UI
    - Support to create UI as particle.
    - Support synthesis of UI.
    - Support the synthesis of UI to any other particle.
17. Service
    - Support for creating web services as particle.
    - Support synthesis of certain type of particle into service patterns.
18. Commission
    - Support royalty particle synthesis.
    - Support synthesis of any custom commission model.
    - Support commission particle synthesized into any NFT.
19. Packing
    - See 3.2 below.

Note that the above types are not immutable. The corresponding types will be merged or expanded with the iteration of the product.Particle provides the processing of conventional image NFTs and comprehensively supports various action functions required in the NFT interaction process. It can even support multiple users to create NFT workflows collaboratively.

3.2 Packing

Among all these types, the packing type needs to be specified.

The packaging class is to pack a variety of NFTs, whether it is an ordinary NFT or a Particle NFT with a script, in batches. Then, the user can transfer it all at once.

4 Synthetic Application

4.1 Definition

We can consider the NFT with program behaviour built by the Particle Protocol as an application or an applet more accurately. Therefore, we use a more professional name, called synthetic application.Synthetic applications can be implemented as various functions, such as GameFi, SocialFi, DeFi, encrypted digital artwork, music and video. The synthesis is implemented on the chain. Unlike DApps, the operation of synthetic applications does not require the additional deployment of the front and back ends of the Web but is aggregated in the NFT. The synthetic application is the basic unit of the Metaverse, and through the synthetic application, we can construct almost all the materials we need.

4.2 Particle Oriented Programming

Particles are not all created by a single organization but are open to the entire community. Developers can program and create different types of Particles to create a wide variety of synthetic applications. Therefore, this method has created a new programming paradigm in the blockchain field, called Particle-oriented programming. This programming paradigm does not always involve programmers. Ordinary users are also one of the programming participants. For example, after an artist creates a picture or music, he can obtain a UI and other Particles that he needs through the market. Then create a music player or picture browser, and the creation does not even need to write a line of code. This method is also called codeless programming.

Moreover, the script function in Particle NFT does not have to be called during NFT synthesis. It can also be called externally, but it requires the user to own the Particle NFT or be authorized by the owner.

4.3 Work Environment

Particle runs on the chain, so there is no compatibility problem with heterogeneous environments, and the synthesis application based on Particle is also completely cross-platform. Note that a similar Java virtual machine does not implement this cross-platform. Instead, it relies on the blockchain environment, which directly accomplishes arbitrary deployment and transfer.

4.4 Synthetic Application Market

By providing a synthetic application market, users can put their application in, which can be publicly released and traded just like the App Store. At the same time, At the same time, quality and safety audits can also be conducted for synthetic applications.

5 MetaRune: Synthetic App IDE

5.1 Definition

MetaRune is an integrated tool for developing synthetic applications. It supports the creation and programming of Particles through a visual interface. The tool supports all types of Particle code templates and provides automated compilation, testing and deployment. MetaRune will support more and more blockchain network environments with iterations, including L2 networks.

5.2 Asset Library

MetaRune supports asset libraries, which can contain various user-defined Particle NFTs. After users claim or purchase multiple Particle NFTs created by the community, they can continue to use them to make their applications. Through asset library management, users can easily view and call the various Particles they owned. The use of this asset library may change the existing open-source software collaboration method, not through the direct import of code, nor through the upgrade and update of dependency management, but through NFT distribution to realize the identification and version management of program assets.

5.3 Collaborative Creation

Through MetaRune, we can create a collaborative NFT project. The synthesis of a large Particle can be divided into several small tasks. Each small task corresponds to its own Particle. Thus, each user can create and synthesize independently. Imagine that in this way, we can invite artists and engineers from all over the world to complete an extensive work full of infinite imagination jointly. We can see the creation process of the work and see different innovations every day. We don't know what notable works will be created in the end, and all of this can be done through Particle Protocol.

6 Economic Model

Particle protocol has its own unique economic model, which is used to obtain benefits and drive community development. At present, it supports four types of economic models.

1. Gas fee
   When the particle is claimed by the user, you can obtain additional fees. Different types of particles have different fees.
2. Royalties
   The value-added income is obtained in the process of particle transaction through royalty. If the receiver continues to synthesize and create on the basis of the obtained particle, the receiver can also become a part of the royalty income, which can be understood as the superposition of new royalties on the work.
3. Market Appreciation
   Particle in the market, it will increase in value due to users' holding and trading. Once the entire NFT creation market is activated, the community will have more and more demand for particle. With the improvement of user recognition and the increase of utilization rate, new value will be generated in the existing market.
4. Token Incentive
   In the future version iteration of particle protocol, the design consideration of the token economic model will be carried out to stimulate community creation and technology development.

7 Conclusion

Particle protocol is a set of protocols supporting metaprogramming of NFT. The NFT synthesized through particle complies with ERC721. We can call it particle NFT, or particle for short. If the synthesized NFT has the ability to provide some functional services to the outside, it can be called a synthetic application. The whole product system includes the particle contract, the contract script and rendering script in metadata, as well as the synthesis engine and rendering engine. At the same time, it also provides the web app and mobile app as the particle container. For user development, the product also includes the MetaRune IDE, which provides developers with creative convenience.

Particle is not only the engine of synthetic applications, but also the productivity component of the Metaverse.

8 Glossary

名词	含义
NFT	Non-Fungible Token Standard
metadata	Defined in NFT to describe specific attributes
Particle Protocol	An NFT metaprogramming protocol
Particle	NFTs defined or generated through the Particle Protocol are collectively referred to as Particle
Particle NFT	The meaning is the same as that of Particle
rendering script	A script that implements the particle NFT presentation layer,defined in the metadata of the particle
contract script	A Script that implements the behavior of particle NFT,defined in the metadata of the particle
rendering engine	The program engine used to execute rendering scripts
synthesis engine	The Program engine used to execute contract scripts
synthetic application	Applications created by particle synthesis

Reference

[1] William Entriken, Dieter Shirley, Jacob Evans, Nastassia Sachs. 2018. URL: https://eips.ethereum.org/EIPS/eip-721.

[2] Witek Radomski, Andrew Cooke, Philippe Castonguay, James Therien, Eric Binet, Ronan Sandford. 2018. URL：https://eips.ethereum.org/EIPS/eip-1155

[3] Matt Lockyer, Nick Mudge, Jordan Schalm. 2018. URL：https://eips.ethereum.org/EIPS/eip-998

[4] Johann Barbie, Ben Bollen, pinkiebell. 2019. URL：https://eips.ethereum.org/EIPS/eip-1948

[5] Zach Burks, James Morgan, Blaine Malone, James Seibel. 2020. URL：https://eips.ethereum.org/EIPS/eip-2981

[6] Calvin Koder. 2021. URL：https://eips.ethereum.org/EIPS/eip-3386

[7] Nathan Ginnever. 2021. URL：https://eips.ethereum.org/EIPS/eip-3440

[8] Sean Papanikolas. 2021. URL：https://eips.ethereum.org/EIPS/eip-3569

[9] Zhenyu Sun, Xinqi Yang. 2021. URL：https://eips.ethereum.org/EIPS/eip-3589

[10] TylerZ, TY. 2021. URL：https://github.com/DRepublic-io/EIPs/blob/master/EIPS/eip-3664.md

[11] Remco Bloemen, Leonid Logvinov, Jacob Evans. 2017. URL：https://eips.ethereum.org/EIPS/eip-712

Disclaimer

This document is for reference only. It does not constitute an investment proposal or an offer or solicitation to buy or sell any investment, nor should it be used to assess the value of making any investment decision. It should not be used for accounting, legal or tax advice or investment advice. This article reflects the author's current views and does not represent particle protocol or its subsidiaries, nor does it necessarily reflect the views of particle protocol, its subsidiaries or individuals related to particle protocol. The comments reflected here may change and will not be updated.