Mech Marketplace description and contracts overview

Summary: This document outlines the core workflow of the Mech Marketplace, detailing the roles and interactions of key participants. It also explores key design choices and inherent vulnerabilities, and offers an in-depth overview of the smart contracts that govern and facilitate transactions within the platform

Created: Jan 16, 2025
Current Version: 1.0.0
Target Version: 1.1.0
Approved for external use by: N/A

Status: Approved
Owner: Mariapia Moscatiello Silvère Gangloff
Contributors: Aleksandr Kuperman
Other stakeholders: [tag here]
Approvers: David Minarsch

1. Overview

The Mech Marketplace allows:

- any agent registered on Olas ServiceRegistry create its corresponding Mech on the Mech Marketplace in order to deliver task-based services;
- any account to request some task executions from Mechs registered on the Mech Marketplace.

This document provides an overview of the workflow Mech Marketplace interacts with and describes the Mech Marketplace contracts located here.

2. Workflow

An agent performing task-based services, is registered and deployed in Olas ServiceRegistry and has a specified serviceId. We refer to such agents as Mech services. These Mech services can be registered in the MechMarketplace contract, which triggers the deployment of a Mech contract associated with the given serviceId [this is illustrated on **Fig. 1**]. Once deployed, these Mechs are available for task execution through the MechMarketplace.

A requester is the party[1] that submits the task execution to a Mech. The submission can be done in two different ways:

1. On-chain requests [Fig. 2 & Fig. 3]

[1] A requester can either be an EOA or a multisig. When the requester deliberately specifies a serviceID while making a request via the Mech Marketplace, the multisig associated with that serviceID will be responsible for covering the cost of task execution.

The on-chain option can be done by sending the request to the MechMarketplace, specifying the serviceId of the mech they wish to engage for the execution of the task. This Mech is referred to as a 'priority Mech'. The requester also specifies a responseTimeout[2], which is the time they expect the priority Mech to deliver the task for. In order to submit a request, a payment must be made, which covers the cost of the task execution as well as a protocol fee. Payments are managed through the Balance Tracker contracts, which track and handle payments made in either native tokens, ERC20 tokens, or via subscriptions, depending on the Mech's payment model, and check that funds of the requester are sufficient. If it is the case, the request is relayed to the priority Mech.

If the priority Mech completes the task within the designated responseTimeout, it is rewarded with a positive Karma score. If the Mech fails to meet the timeline, it is penalized with a negative Karma score. In such cases, any other Mech can step in to fulfill the task, and the first Mech doing so receives a positive Karma score for execution of the task.

2. Off-chain requests [Fig. 4]

The on-chain option can be done by sending a request with a signature directly to a Mech service off-chain. In this case, the delivery must be already pre-paid, such that the requester has a balance tracker balance covering all the signed requests. Otherwise the delivery is rejected. The Mech can then deliver on-chain with the requester signature, gets its payment via the Balance Tracker contract and gets rewarded with a positive Karma score. In this situation, there is no deadline and no other Mech can take-over if the Mech fails to deliver (in particular no Mech receives a negative Karma score).

In both cases, Mech payments for requested task executions are collected in the Balance Tracker contract, and Mechs can collect payment for delivered executions from the Balance Tracker. A portion of the payment is retained by the Balance Tracker for DAO management, which can later be drained into the DAO chosen drainer by any account [this is illustrated on **Fig. 5**]. The exact payment structure (fixed or dynamic pricing) and the tokens used for payment depend on the Mech's pre-set payment model.

[2] In order to have responseTimeout being reasonably selected this value is bounded below and above by values specified by the Mech Marketplace

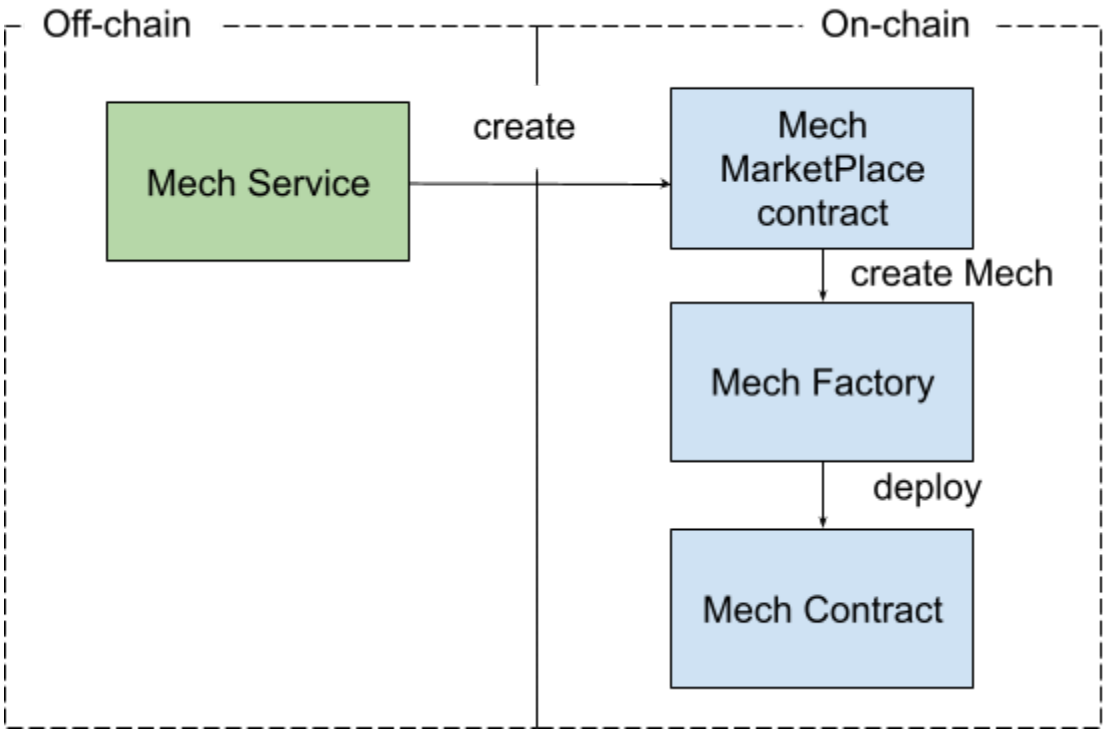


Fig. 1. Mech Service registration on the Mech Marketplace contract with an on-chain transaction. Mech Contract deployment is triggered.

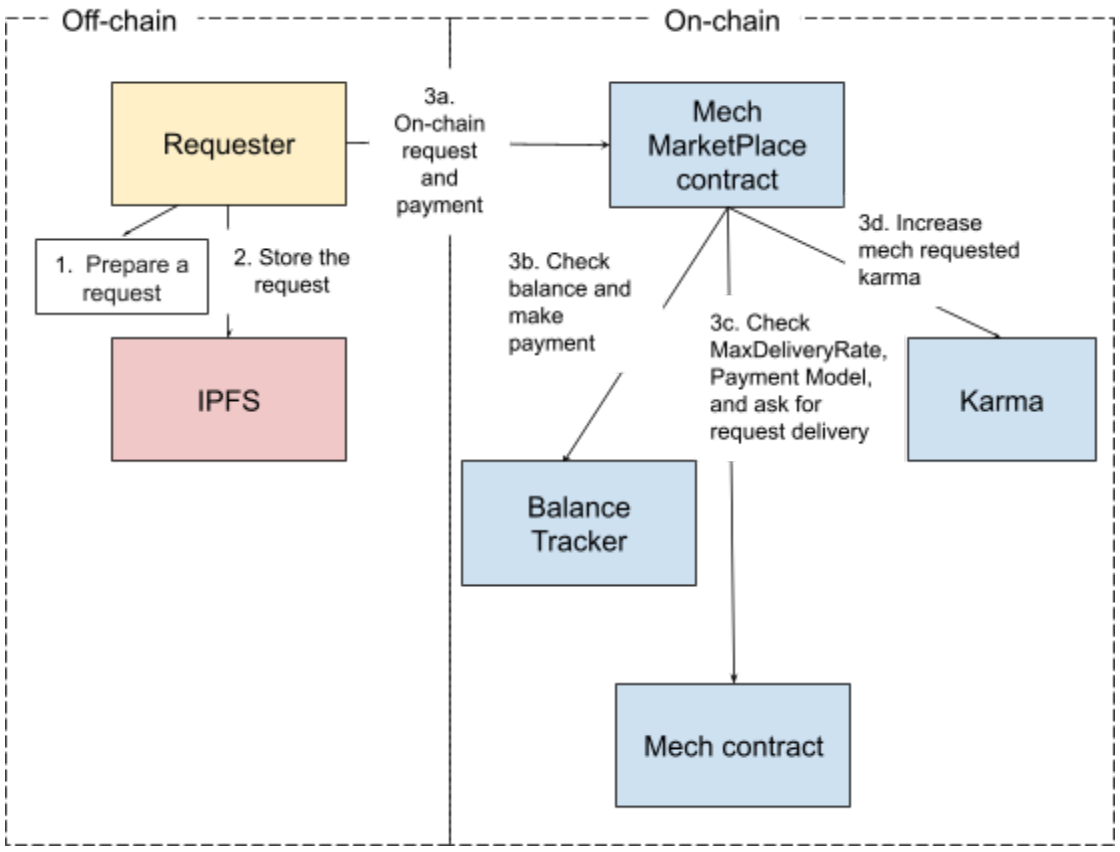


Fig. 2. Requester sending on-chain requests to the Mech via the Marketplace. Note that the only on-chain request transaction (3a), also triggers 3b, 3c, and 3d.

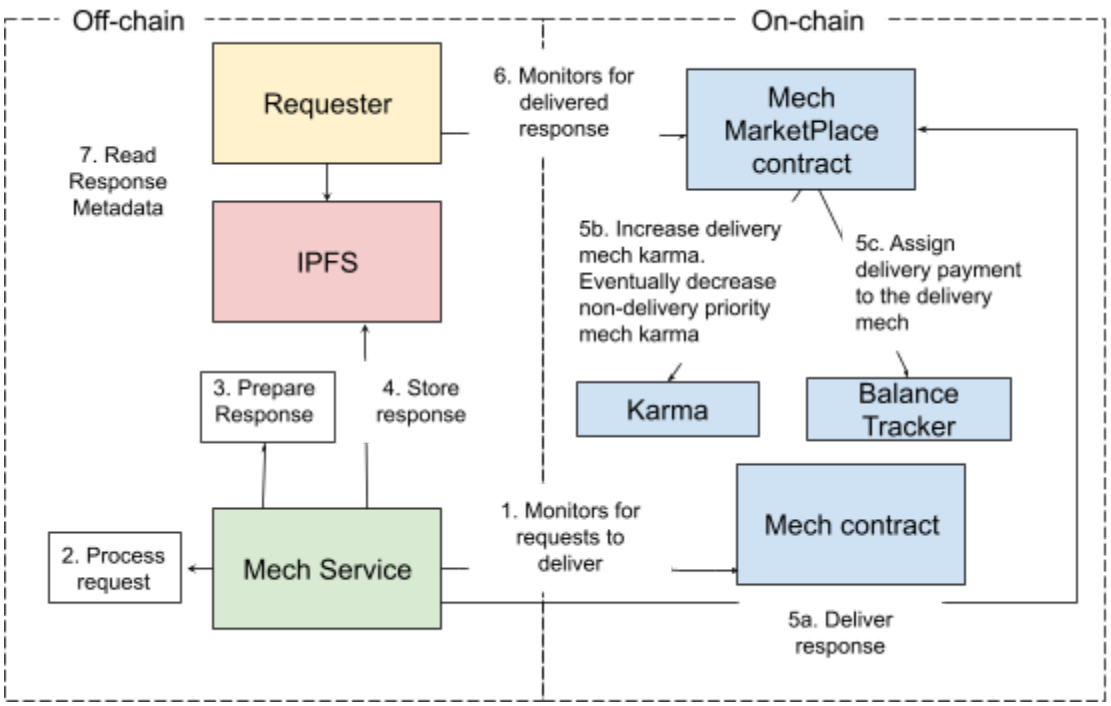


Fig. 3. Mech delivery response to an on-chain request via the Marketplace. Note that only on-chain request transactions (5a) also triggers 5b and 5c.

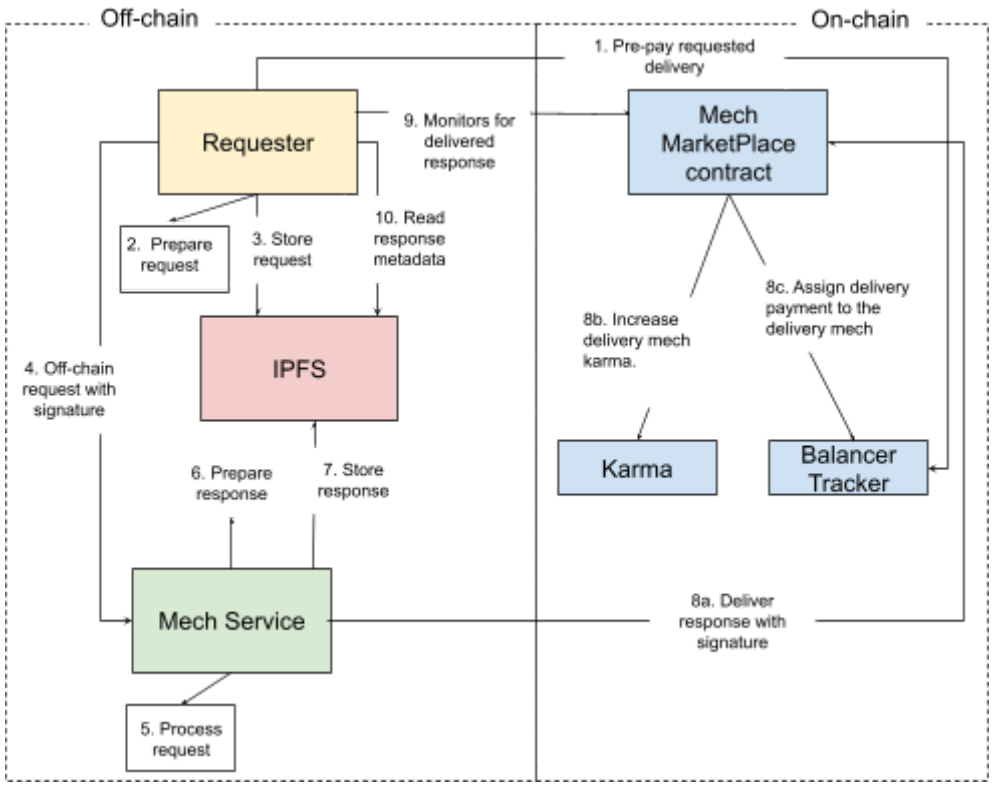


Fig. 4. Off-chain request with signature and mech delivery with signature workflow. Note that, despite pre-payment being marked as step 1, it only needs to happen before the on-chain mech delivers with a signature.

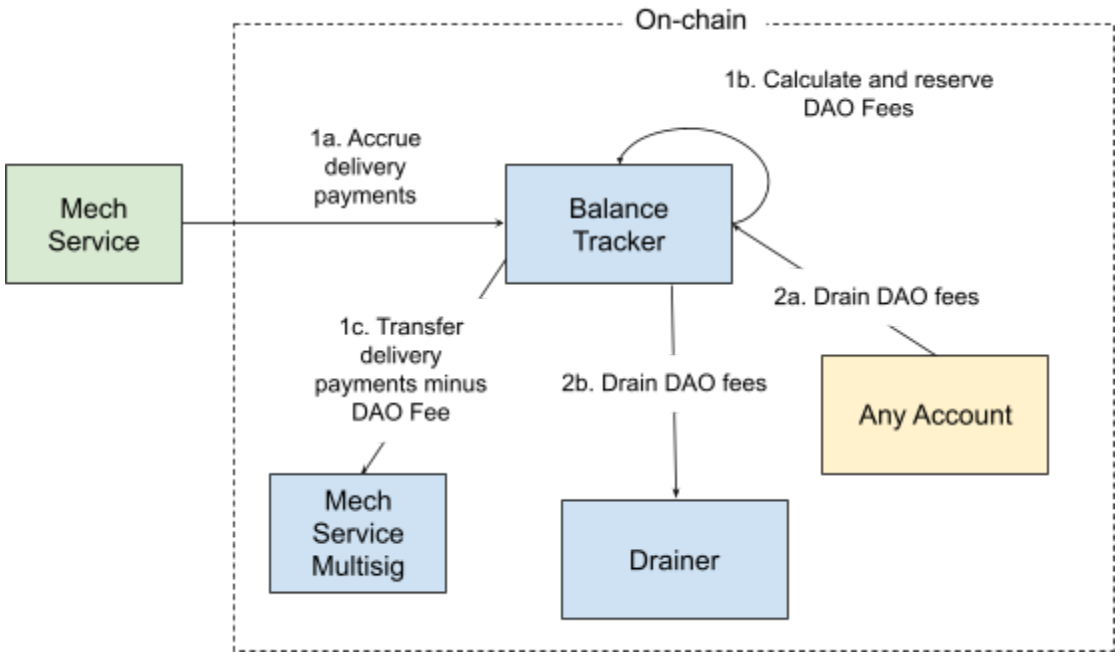


Fig. 5: On-chain flow for Mech's recovery of payments and DAO's Mech Marketplace fees. Note that the on-chain payment accrual transaction (1a) also triggers 1b and 1c, similarly, the on-chain drain call 2a also triggers 2b.

3. Design Choices and Inherent Vulnerabilities in the Mech Marketplace System

Mech Due Diligence: Balancing Delivery Efficiency and Risk

In the fast-paced Mech Marketplace, Mechs compete to complete tasks quickly and efficiently to secure payment and boost their Karma scores. Speed is crucial, as delivering tasks faster increases the chances of receiving payment before competitors. However, before accepting a task with a signed request, a Mech must perform thorough due diligence to ensure it can meet delivery requirements and deadlines. Mechs must strike a balance between thorough verification and efficiency—over-verification can slow down delivery, while skipping checks could

lead to costly mistakes. The key is to perform the necessary checks without compromising delivery time. In view of this, Mechs should consider the following:

- **Signature Validation:** For off-chain signed requests, the Mech should validate the authenticity of the provided signatures. Requests with unverified or improperly signed transactions should not be processed on-chain.
- **Balance Check:** Verify the requester's on-chain balance, particularly for off-chain requests, to ensure they have sufficient funds for the task. Specifically, the mech should verify that the delivery will be finalized on-chain if submitted against the current state. One way to do it is for delivery mechs to perform the gas estimate, and if failed - not proceed with requested delivery. In order to save on compute, the delivery gas estimation can be done with mocks of delivery data first to make sure the requester has correct balances to cover. Note that, while balance verification is a useful safeguard, it doesn't guarantee payment for off-chain tasks (see the "Failure of Delivery Payments" section in the *Karma System: Vulnerabilities, Exploits, and Risks* discussion)
- **Reputation Tracking of Requesters:** To mitigate the risk of payment failures, unsuccessful on-chain deliveries, excessive time spent on off-chain computations, or Karma system manipulations, Mechs could implement a reputation tracking system for requesters. By monitoring a requester's history of successful and failed payments, as well as signature-based deliveries, Mechs can assess whether the requester is likely to fulfill payment obligations or pass necessary verifications before proceeding with task delivery. Requesters with a poor reputation could be blacklisted or deprioritized. This reputation system would encourage requesters to maintain a reliable payment record and provide correct signatures, thereby saving Mechs time and resources on off-chain tasks. It would also reduce the risk of malicious exploitation by requesters who seek to leave Mechs unpaid after task completion or attempt to manipulate the Karma system.

Fulfillment of Requests by Mechs

The Mech Marketplace is designed with the assumption that, eventually, a request will be fulfilled by a Mech. As a result, requesters cannot withdraw their balance from the balance tracker. This assumption is grounded in the competitive nature of the marketplace, where Mechs are incentivized to deliver tasks quickly and efficiently to earn payment and improve their Karma scores. However, in the rare event that all Mechs fail to fulfill a task, the requester can create new requests, which may eventually help deplete the locked balance over time.

Karma System: Vulnerabilities, Exploits, and Risks

The Karma system was intentionally built on the assumption that Mechs operate in a competitive environment, striving to complete tasks quickly to gain positive Karma and avoid negative Karma. This competitive foundation drives Mechs to perform at their best and may ensure that the Karma system accurately reflects their performance. However, like any system, the competitive nature also opens the door to manipulation. Even a small number of malicious actors can exploit the system and undermine its fairness. Below are some of the key

vulnerabilities and exploits that could weaken the integrity of the Karma system. Please note that this list is not exhaustive:

Sybil Attacks and Manipulation: A Sybil attack occurs when an attacker creates multiple fake Mechs to manipulate the Karma score or exploit the reputation system. For instance, an attacker could register several Mechs and intentionally fail to meet task requirements with some of them, while ensuring that one Mech performs well to artificially inflate its Karma. This manipulation skews the reputation of the Mechs involved, making it difficult for the system to assess their true performance.

Collusion Between Malicious Requester and Mech: Another vulnerability arises when there is collusion between a malicious requester and a malicious Mech or a Sybil attack where a single user controls both the requester and the Mech. In this scenario, the malicious requester sends off-chain multiple task requests to a competitor Mech, each containing issues such as incorrect balances or improper signatures. These problematic requests are designed to cause delays, forcing the competitor Mech to spend excessive time on off-chain verifications. As a result, the competitor's tasks often time out before delivery, enabling the malicious Mech (controlled by the attacker) to step in and complete the task. This allows the malicious Mech to inflate its Karma score unfairly, while the honest Mech is penalized for failing to deliver on time.

Failure of Delivery Payments: Another vulnerability arises from the failure of delivery payments. In this case, a malicious requester may initiate multiple off-chain task requests to different Mechs, but only have enough balance to pay for a few of them. Even if a Mech conducts proper due diligence by verifying that the available balance can cover the delivery of the requested tasks, it may still incur a loss if another Mech completes and delivers the task first. Specifically, if the other Mech signs the delivery transaction before the initial Mech, the first Mech may be left unpaid despite having fulfilled its obligations.

Consideration about Mech Contract Creation

With the current implementation[3], anyone can create mechs using the same service ID, but only the mech multisig can ensure that the contracts function correctly. Specifically, critical mech functions, such as delivery and signature-based delivery, are protected by the onlyOperator modifier. This approach is based on an optimistic model, assuming mechs are created with the correct service ID configuration, which enables multiple mech contracts under the same service ID to coexist. This flexibility is important, for example, when updating the payment model of a mech linked to a service ID. However, since anyone can create mechs with the same service ID, it is essential for requesters to carefully verify the configuration of the mech they intend to interact with.

[3] Regardless of whether the implementation logic is updated while maintaining the optimistic approach, critical mech functions should still be protected by the onlyOperator modifier to ensure security and proper control.

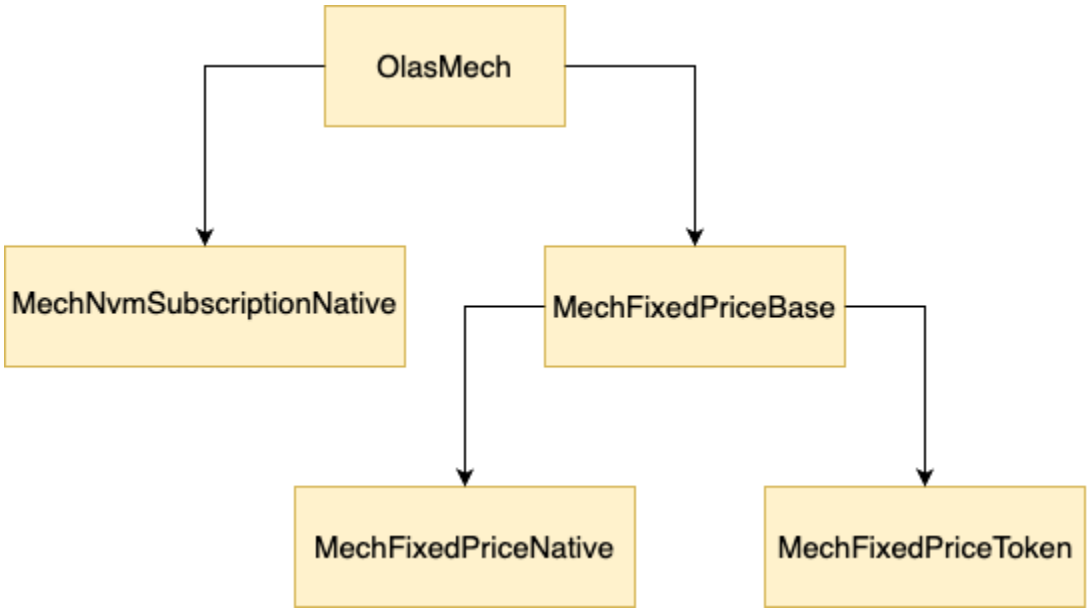
4. Description of contracts

The Mech Marketplace is constituted of on-chain contracts made up of the following components:

1. Mech contracts receive requests and payments from requesters via the Mech Marketplace contract and respond to these requests through it;
2. Mech Factory contracts enable the creation and deployment of Mech contracts;
3. Balance Trackers are requesters' and Mechs' accounts in the Mech Marketplace storing mechs'/requesters' balances; Mechs are able to withdraw their balances at any time; requesters are not allowed to withdraw their balances, just to use them for posting requests; when mechs payment processing happens, the Mech Marketplace takes a share which is accumulated in its balance in the Balance tracker, and can be withdrawn at any time;
4. The Mech Marketplace contract relays requests from requesters to Mechs, handles the competition between Mechs and checks that the amount available on each requester's balance is enough to relay a request to a Mech.
5. The Karma contract maintains and updates a reputation score of Mechs;

The functionality described in the following sections corresponds to v0.4.0-pre-deployment version of the code.

3.1. On-chain Mechs



For the (on-chain) Mech implementation contract, the **OlasMech** contract is an abstract contract. Each child contract corresponds to a *payment model*. At the moment there are two payment models (fixed price and Nevermined subscription) two child contracts:

1. **MechFixedPriceBase**: the price per request is fixed, meaning that it does not depend on the request;
2. **MechNvmSubscriptionNative**: the Mech implements dynamic pricing via a Nevermined subscription.

Furthermore, MechFixedPriceBase has also two child contracts, each corresponding to an asset type:

1. **MechFixedPriceNative**: the Mech is paid using native token;
2. **MechFixedPriceToken**: the Mech is paid using ERC20 token;

OlasMech main functions are the following ones:

changeMaxDeliveryRate(uint256 newMaxDeliveryRate**):**

this function is called only by the Mech service multisig in order to change its maximal price (= delivery rate) (which is equal to the delivery price for Mechs with fixed price) to newMaxDeliveryRate.

requestFromMarketplace(bytes32[] calldata requestIds, bytes[] calldata datas**):**

this is called only by the Mech Marketplace in order to trigger the processing of a batch of requests (requestIds) by a Mech (typically the priority mech of a request batch in the current implementation).

deliverToMarketplace(bytes32[] calldata requestIds, bytes[] calldata datas**):**

this function is called by Mech multisigs in order to deliver responses to a batch of requests (specified by requestIds) by calling the function **deliverMarketplace()** of the **Mech Marketplace**;

deliverMarketplaceWithSignatures(address requester, DeliverWithSignature[] calldata DeliverWithSignatures, uint256[] calldata deliveryRates, bytes calldata paymentData**):**

called by Mech multisigs in order to deliver responses to a batch of requests (specified by requestDatas in DeliverWithSignature), using the **deliverMarketplaceWithSignatures()** of the **Mech Marketplace**. The main difference with the previous function is that requests are signed (meaning that they come with account signature), and this function checks the validity of the provided signatures (signatures in DeliverWithSignature) collected by the Mech along with requests. This approach enables the requester to send off-chain requests to the Mech service,

which responds immediately off-chain and then asynchronously settles requests and payment on-chain. Contrary to requests sent on-chain, it is not checked at the time the request is sent if the requester's balance is not sufficient for the payment. When this is the case, the delivery is rejected.

3.2. Factories

Mech contracts can be deployed using the following factories:

- **MechFactoryFixedPriceNative**
- **MechFactoryFixedPriceToken**
- **MechFactoryNvmSubscriptionNative**

3.3. Balance Trackers

3.3.1 Abstract contract

Balance Trackers are auxiliaries of the Mech Marketplace through which it receives funds from requesters and manages payments to Mechs for execution delivered. Additionally, Balance Trackers enable the drain of collected fees.

The abstract contract for balance trackers is **BalanceTrackerBase.** Again, each child corresponds to a payment model. At the moment there are three child contracts:

1. **BalanceTrackerFixedPriceNative**
2. **BalanceTrackerFixedPriceToken**
3. **BalanceTrackerNvmSubscriptionNative**

Balance Tracker's main functions are:

checkAndRecordDeliveryRates(address requester, uint256 numRequests, uint256 deliveryRate, bytes calldata paymentData**)**:

this function is called only by the MechMarketPlace contract, checks that the requester (requester) balance is sufficient to pay the priority Mech for the processed request, records the price (deliveryRate), and removes the corresponding amount from the requester balance. Only after this the price amount is added to the Mech's balance.

finalizeDeliveryRates(address mech, address[] calldata requesters, bool[] calldata deliveredRequests, uint256[] calldata mechDeliveryRates, uint256[] calldata requesterDeliveryRates**):**

This enables the finalization of the prices of requests in a batch at the time of delivery (it is guaranteed that this price is lower than the maximum delivery rate of the Mech, as it is checked when the request is received by the MechMarketplace, therefore any possible price adjustment

is valid); since the request price is recorded, Mechs can update the request price at the time of delivery, by setting their updated maxDeliveryRate, or when its payment model is Nevermined subscription; the price of request i is replaced with mechDeliveryRates[i], meaning that the the difference (between the price at the time of request and price at the time of delivery) is added to the requester's balance and the final price amount is sent to the Mech's balance (no amount is added before to this balance); all this is possible only if the delivery rate is lower than the maximum of the Mech;

adjustMechRequesterBalances(address mech, address requester, uint256[] calldata mechDeliveryRates, bytes calldara paymentData**) :**

this function is called only by the MechMarketplace contract, in order to adjust requester's (requester[i]) and Mech's (mech[i]) balances for each request i in a batch, by moving the amount corresponding to the price (mechDeliveryRates[i]) from the requester's balance to the Mech's one.

drain():

this function can be called in order to withdraw the fees collected by the Mech Marketplace on each request payment and send this amount to a designated **Drainer** address.

processPayment():

this function is called by a Mech (msg.sender) to collect all its payments at once (no micro-payments); first this function computes the Mech Marketplace fees, which correspond to a percentage (whose value is a constant of the MechMarketplace contract) of the total amount in the Mech's balance, and moves these fees from the Mech's balance to the Mech Marketplace's one; then the amount remaining in the Mech's balance is withdrawn.

processPaymentByMultisig(address mech**):**

same function but called by the Mech service multisig.

3.3.2 Child contracts

The child contracts have the following differences:

BalanceTrackerFixedPriceNative:

This contract has the additional (payable) function **depositFor(**address account**)** which allows anyone to deposit a certain amount to a requester's (account) balance in the Balance Tracker, and **receive()** (also payable) which allows a requester to add an amount to its balance.

The internal function **_drain(**uint256 amount**)** called by **drain()** wraps the amount to withdraw and then transfer from the wrapped token.

BalanceTrackerFixedPriceToken:

This contract also has an additional function **depositFor(**address account**)** which is similar to the one above, except that it uses the function **transferFrom()** of ERC20 (in this case the funds must be previously approved to be sent to the Balance Tracker). It also has a function **deposit()** which allows a requester to add an amount to its balance.

When checking that the balance of a requester is sufficient to send a request, this Balance Tracker first gets the necessary funds from the token before checks (when the Mech is paid with native tokens, all the funds are deposited first into the Balance Tracker).

The internal function **_drain(**uint256 amount**)** called by **drain()** only transfers from the token.

BalanceTrackerNvmSubscriptionNative:

This contract has the additional function **setSubscription(**address _subscriptionNFT, uint256 _subscriptionTokenId**)** which can be called only by the owner and fixes the Nevermined subscription of the Balance Tracker to be the one of the Mech Marketplace (via the parameters _subscriptionNFT and _subscriptionTokenId). The function **receive()** also allows a requester to deposit on this contract.

For this Balance Tracker, all the funds are held and moved externally, at the level of the subscription. In particular when: checking that the balance of a requester is sufficient to send a request; adjusting the price at the time of delivery; the Mech withdraws its payments.

Note that the **drain()** function is not implemented in this contract, as fees are handled at the level of the subscription. The function **withdraw()** does not have any effect as well for the same reason.

3.4. Mech Marketplace

MechMarketPlace is an upgradable and ownable contract that allows Mechs registration, requesters to submit requests, registered Mechs to deliver responses to requests, and manages payments for task execution.

Its implementation functions, besides getters (for request status, request id), are:

requestBatch(bytes[] memory requestDatas, uint256 maxDeliveryRate, bytes32 memory paymentType, address priorityMech, uint256 responseTimeout, bytes calldata paymentData**)**:

this function is called by requesters in order to submit batches of requests (requestDatas) with payments (paymentData), payment model (paymentType), and maximum price (maxDeliveryRate) to the Mech Marketplace, by providing the address of the priority Mech, to which this request has to be sent first (priorityMech); the priority Mech has to answer before the time limit (responseTimeout) has elapsed. Note that it is assumed all requests will be delivered after some time.

deliverMarketplace(bytes32[] calldata requestIds, uint256[] calldata deliveryRates**)**:

This function handles the delivery of requests in a marketplace, ensuring that only valid requests are processed, delivered by the appropriate mech, and that payments and karma are correctly updated.

deliverMarketplaceWithSignatures(address requester, DeliverWithSignature[] memory deliverWithSignatures, uint256[] calldata deliveryRates, bytes calldata paymentData**)**:

this function enables Mech to deliver signed requests sent off-chain, which implies that this function verifies signatures provided by the Mech (similarly as **deliverMarketplaceWithSignatures()** function of **OlasMech**); the delivery is rejected if the requester's balance is not sufficient for the payment.

checkMech(address mech**)**:

the function ensures the address is valid and authorized to operate within the marketplace. If the mech address is non-zero and the mech has been created and recorded via the MarketPlace, the corresponding multisig address for that mech is returned,

create(uint256 serviceId, address mechFactory, bytes memory payload**)**:

called by a Mech multisig (whose id in the Service Registry is serviceId) in order to register on the Mech in the Marketplace; this function triggers the function createMech of the Mech Factory contract (mechFactory) which corresponds to the payment model chosen by the Mech service. The input payload corresponds to the maximum delivery rate chosen by the Mech.

setMechFactoryStatuses(address[] memory mechFactories, bool[] memory statuses**)**:

can be called only by the owner (DAO) and changes the status of Mech factories in mechFactories: for all i, the status of the factory contract mechFactories[i] is set to statuses[i]. This status is a boolean which tells if the factory contract is whitelisted (and thus authorized to register Mech services on the Mech Marketplace) or not.

setPaymentTypeBalanceTrackers(bytes32[] calldata paymentTypes, address[] calldata balanceTrackers**)**:

can be called only by the owner, and sets the Balance Tracker contract (balanceTrackers[i]) corresponding to each payment model available (paymentTypes[i]).

3.5. Karma

The **Karma** contract records the reputation scores of Mechs and handles their changes.

The main function of the Karma contract is:

changeMechKarma(address mech, int256 karmaChange**):**

This function is called only by the Mech Marketplace and adds an integer karmaChange to the reputation score of mech.