by @nlok5923 @hawk

Special thanks to Alex for the helpful discussions and feedback that were instrumental in making this article possible.

### TL;DR

- Blockchain user intentions, or 'intents,' lack a standardised framework for efficient solver collaboration in Ethereum
- This article introduces Abstracted Transaction Objects (ATOs) to capture operation-specific information and optimise user intents.
- A trusted driver manages ATO bundles and broadcasts them to a diverse solver network, each specialised in different domains.
- Solvers generate scores based on user-specified or operation-specific fields, and the winning solution is determined by Degree of Expectation (DoE), a quantitative measure.
- Collaboration among multiple solvers ensures optimal solutions, censorship resistance, and system availability, driven by reputation scores.
- Solver incentives include transaction fees, prepaid balances, and an intent execution module integrated into smart contract wallets for streamlined transactions and automated solver rewards.

## Current Scenario

In the blockchain realm, user intentions, referred to as 'intents,' take on diverse forms, including DSL expressions, natural language, actions, conditions, and restrictions. An intent captures how users envision specific transaction actions. For instance, a common example is choosing 'Same day delivery' for an Amazon order. The incorporation of Account Abstraction(ERC 4337) extends the range of these expressive possibilities.

At present, there is no established architecture for solvers in the Ethereum ecosystem to collaborate and coordinate their efforts. The existing solver ecosystem operates in isolation, resulting in limited solver visibility and complicating the user's process of discovering suitable solvers for their intent fulfilment.

The MEV extraction thats taking place in the DeFi protocols is increasing and with flashbots new solution SUAVE does provide a way to mitigate this but those approaches are quite farsighted and depends on adoption. This leaves the users in the light of increasing loss of funds in the form of MEV. The MEV captures can be further be reduced by leveraging solver collaboration and facilitating data sharing among solvers which would introduce counter party discovery and could lead to potential coincident of wants.

Ethereum, as a stateful blockchain employing a Virtual Machine (VM), faces a key impediment to implementing this architecture within the EVM. This obstacle lies in the transaction construction process. In the Ethereum ecosystem, transactions adhere to a deterministic approach, capturing only a limited set of information required to effect a state change. This design resists further optimization. In contrast, Intents aim to capture users' desires, subsequently translating them into optimized transactions by solvers, thereby offering greater flexibility and value to users.

In order to align Intents with the EVM, we are introducing a novel structure known as Abstracted Transaction Objects (ATOs)

. ATOs exclusively capture information relevant to a specific operation. Solvers leverage this information to construct optimized transactions tailored to the operation's requirements.

### Tentative ATO structure

{ "operation": ENUM, "fieldsToOptimize": hex, "fieldsToOptimizeSchema": string, "chainId": number, "payload": hex, "payloadSchema": string }

The fieldToOptimize

field contains the fields for an operation which are necessary for score optimization in an encoded form. Score converts the solution of a solver into quantitative value which can be used by the driver to evaluate and compare the solutions received from different solvers. fieldToOptimizeSchema

contains the schema followed for decoding the fieldToOptimize

field. chainId

is the chain Id of the chain on which the user want to executes the intent. payload

is the extra information apart from fieldToOptimize

to convert that ATO into a valid transaction by the solver in an encoded format. payloadSchema

contains the schema followed for encoding the payload

field. The Intent(I)

captured at application level can be composed of n

ATO’s, which might correspond to same operation

more than once. Theoretically n can go till \infty

but in practical case scenarios it would be some definitive number.

I = [ATO_{1}+ATO_{2}+.....+ATO_{n}] \;\;n \in[1,\infty)

These ATO’s can be presented in the form of a private manner where information is hidden and executed in such a way that the user doesn’t reveal any information to the public blockchain.

# Managing Abstracted Transaction Objects (ATOs): The Role of the Driver in Intent-Driven Architecture

[

efw

3812×1924 143 KB

](https://ethresear.ch/uploads/default/original/2X/b/ba098e6ab88b189d168b9c41de99e9a1f0adcaaa.png)

The driver plays a pivotal role within the infrastructure, serving as a trusted party with several key responsibilities:

1. ATO Broadcasting:

The driver is tasked with broadcasting Abstracted Transaction Objects (ATOs) to the mempool, where all the solvers can initiate their execution processes to find optimal solutions.

1. Simulation and Verification:

It receives solutions from all the solvers, conducts off-chain simulations to ensure their validity and security, and subsequently posts the winning solution.

1. Aggregation of Solutions:

For a given Intent, the driver aggregates solutions from different ATOs, combining them into a unified execution plan for final implementation.

## ATO Bundling and broadcasting

[

ATO ordering and bundling

1600×921 96.6 KB

](https://ethresear.ch/uploads/default/original/2X/8/89b372db426bbc2f37e7d607a52c493325b52500.jpeg)

The driver efficiently manages bundles of Abstracted Transaction Objects (ATOs) with each bundle focusing on resolving a single user intent. In the case where the intent corresponds to a single action, the driver can also receive a single ATO.

Upon receiving these bundles, the driver undertakes the following tasks:

Parallel Ordering:

The driver performs parallel ordering of ATO bundles, to start extracting and bundling ATOs to broadcast.

Bundling ATOs:

It bundles the first ATO from each bundle based on their operation type. For example, if there are four bundles of ATOs, two of which have a swap operation as their first ATO, and the other two have a bridge operation as their first ATO, the driver will broadcast two bundles, one for the swap operation and one for the bridge operation, to their respective solvers. This organised approach streamlines the ATO distribution process to the solver network.

For the first ATO in each bundle, no new state information is required, as the solver can utilize the current blockchain state to find the optimal solution for that specific ATO.

The driver follows a structured process. For the initial ATO's of the bundle, it doesn't require additional state information, and the solver can proceed with the available blockchain state.

However, for subsequent ATOs in the bundle, the driver collects the corresponding solutions from all the previous ATOs before bundling the next set of ATOs. This ensures that the solver has updated state information before solving the new ATO, as some fields may depend on the information from preceding ATOs.

In essence, the driver maintains a continuous cycle for every non-initial ATO. Before broadcasting it, the driver ensures it has the winning solutions for all preceding ATOs, facilitating an organised and synchronised approach to solving user intents.

## How the Solver Network Enhances the Intent-Driven Architecture?

A solver is a participant within the context of this system. It operates within a solver network, a collective repository of diverse solvers, each specialising in distinct domains. Individuals with these pre-requisites can be a solver:

- Valid ethereum address:

Individuals with valid wallet addresses can partake in this ecosystem by registering themselves as solvers through a dedicated smart contract.

- Domain Experts:

These solvers are equipped with specialised code tailored to solve specific types of ATO for a particular operation (swap, bridge, stake etc). For example a solver specialising in derivative trading

- Win the DAO voting process:

Remarkably, the selection process for solvers extends to decentralised autonomous organisations (DAOs), where members can exercise their voting rights to determine the inclusion of a solver.

This collaborative approach ensures a diverse and expert-driven solver network, enriching the architecture's capabilities.

While drivers facilitate the routing of client-originated ATOs to their designated solvers, a vital consideration arises in the design of solver networks. Entrusting a solitary solver with the complete authority to address all ATOs introduces several challenges:

- Optimal Solution Exploration:

The absence of competition within a single solver's domain can compromise the attainment of the best solution for an ATO. Solver networks mitigate this concern by fostering an environment where multiple solvers contribute their expertise, enabling diverse approaches to be explored and the most effective solutions to be identified.

- Censorship Resistance:

The potential for a singular solver to enact censorship and selectively decline certain users' ATOs is a concerning aspect. Solver networks circumvent this issue by distributing ATOs across various solvers, promoting fair and equitable execution without the risk of undue censorship.

- Enhanced Availability:

Relying solely on a single solver presents a vulnerability; if that solver becomes unavailable, the entire protocol grinds to a halt. Solver networks avert this predicament by distributing tasks across a multitude of solvers, ensuring that the system remains operational even if certain solvers experience downtime.

### Mempool

Including multiple solvers in the system proves to be a sensible approach, capable of handling the diverse stream of ATOs originating from the driver. These solvers collectively form a solver network, collaborating to decipher ATOs and achieve the best possible solutions, an effort that earns them well-deserved rewards. Within this framework, ATOs find their home in a shared mempool, accessible to all participating solvers. Further we are planning for solver to have their own local mempools and in future we would try to enable user to directly sent their ATOs to one of the solver and from the local offchain mempools the solver which doesn't posses expertise in solving the ATO which they have received they can share those ATO's with their neighbour solver having expertise in solving that. Additionally their would be an auction period setup by driver by that auction period it's the responsibility of solver to return the solution for that particular ATO failing to do so might lead to the failing of ATO solution acceptance.

# Degree of Expectation: A Quantitative Approach for Optimising Intents

The solver optimises ATOs based on user-specified or operation-specific fields represented by a set of qualities T

for a particular ATO, the optimisable fields could be swap rates, percentage yield, contained in the ATO's fieldsToOptimize

field. Optimising this fields enables solvers to evaluate their solution and provide a score for their solutions, which the driver validates to prevent potential malicious score calculations by solvers.

T \subset set\;of\;user\;defined\;optimizable\;fields

d: number\;of\;default\;fields\;to\;optimise

of_{i}: optimisableField_{i} \;\;\forall \;\; i \in [1, n(T)+d] \; \;\;n(T): cardinality \;\;of\;\;set \;\;T

fieldsToOptimize: abi.encode(of_{1}, of_{2}.....of_{n(T)+d})

f_{i}:optimisableField_{i} \rightarrow fieldValues_{i} \;\; \;\;\forall \;\; i \in [1,n(T)+d]

A function f_{i}

takes in optimisableField_{i}

and returns a quantitative representation of the optimisation achieved for the overall ATO. The specific function for mapping the qualitative nature of these optimisableField

to fieldValues

will be determined through community and solver discussions and these fields may encompass various aspects such as bridge reputation and DEX slippage. If a field is not explicitly mentioned, it is excluded from the optimisation problem.

In situations where no optimisable field is explicitly stated in the Intent, default fields that can be optimised would be considered. These default fields are determined through community and solver consensus when enabling support for a new operation into the protocol

All the solvers calculate the same expression and then optimise over it. There will be a function which will represent the 'Degree of Expectation' (DoE)

which the solver is trying to optimise in T (auction time)

time for which the ATO is valid to the solver. The DoE only contains the value which maximise the user's expectation explicitly

DoE_{operation}\propto \frac{fieldValues_{positiveQuality}}{fieldValues_{negativeQuality}}

fieldValues_{positiveQuality}

: represents the numerical Values (values received on optimising the optimisableField) of the ATO which are directly proportional to the DoE of the operation. For example this can be reputation of the bridge in a bridging operation. Basically it is the quantised representation of the fields of ATO which upon optimising providing surplus value to the user.

fieldValues_{negativeQuality}

: represents the numerical values of the ATO which are inversely proportional to the DoE of the operation. for example: swapping with a fairly large slippage. Similarly it is the quantised representation of the fields of ATO which upon optimising in positive manner could results in user losses.

For a particular ATO_{i}

where ATO_{i} \in [ATO_{1}, ATO_{n}]

we have solutions getting from all the solvers from [solver_{1},solver_{m}]

and we represent it in the form of DoE of the solver_{j}

for the$ATO_{i}$ as DoE_{solver_{j}}^{ATO_{i}}

In a more simplified way we can express it as:

DoE_{solver_{j}}^{ATO_{i}}: Degree\;\;of \;\;expectation \;\;for\;\; a \;\;particular\;\; ATO_{i} \;\;provider\;\; by\;\; solver_{j}

For a particular ATO_{i}

we get all the DoEs possible from the solvers and the winning DoE for a particular ATO is defined as the maximum of all these values:

$$DoE^{ATO_{i}} = \max (DoE_{solver_{1}}^{ATO_{i}}, DoE_{solver_{2}}^{ATO_{i}},....., DoE_{solver_{m}}^{ATO_{i}})$$

Solver which will provide max DoE for it's ATO would be declared as winning solver and solution corresponding to that ATO would be accepted in the final transaction bundle.

### Greedy vs DP approach

If we take the summation of all winning solutions of all the $ATO_{i}$

forming an Intent I

to calculate the Total Degree of Expectation(TDoE) of the intent we get this expression:

$$TDoE=\sum_{i=0}^{n} DoE^{ATO_{i}}$$

Currently, We are taking a greedy approach to solve the problem. In the current approach we are focused on optimising the DoE for the current ATO, For now we are not considering the implications of previous DoE over the current DoE calculations with respect to optimisations. For now we are just using the previous ATO solution to complete the current ATO's optimisation problem.

We can understand this problem with an example:

Intent: I have 10 USDC on polygon use that and quickly give me max USDT on Gnosis.

The above example results in the formation of two ATO first would be for bridging USDC token from polygon to Gnosis quickly and another would be for swapping USDC token to USDT on Gnosis for best rates

With the current infra we are thinking that the bridge solver would solve the first ATO and come up with the solution for bridging tokens with fastest bridge but that bridge may not be providing tokens on destination chain with best rates. Although the swapping on the destination chain would occur with lowest slippage (a.k.a best rates) considering the ATO.

This in general is a relatively hard problem and at we will be first going with the greedy approach for implementation.

## Solution Simulation and Agreement

Before accepting any solution, it falls upon the Driver to simulate the solution, ensuring it aligns with the score commitment shared by the solver. It's essential to note that both the driver and the solver operate with a shared scoring mechanism. This mutual understanding mandates the solver to initially evaluate the solution on their end. This assessment is based on the optimisation techniques employed by the solver and the state information provided by the driver. Once satisfied with the results, the solver then commits the score, along with the solution, to the driver.

Interestingly, the onus of score computation lies solely with the solver. The rationale behind this approach is to prevent overwhelming the driver with the responsibility of evaluating each ATO. Upon receipt of the solution, the driver embarks on simulating the highest-scoring solution, referencing the previous state data. If this simulation resonates with the score initially shared by the solver, the solution is heralded as the winning one. Furthermore, the solver responsible for that winning solution is subsequently marked as the beneficiary of the incentives amassed post-solving that particular intent of which that ATO was part of.

However, discrepancies might arise. If the driver's simulation yields a score that doesn't match the solver's commitment, penalties will be levied on the solver who provided the solution. The driver will then proceed to simulate the next highest-scoring solution. This process of simulation and score verification continues until the driver identifies a solution that matches the originally committed score.

## Solution Acceptance and aggregation

Upon receipt of ATO bundles, the driver is tasked with appending specific tags to each ATO prior to broadcasting them to the solver. These tags play a pivotal role, enabling the driver to later discern the linkage of each ATO to its corresponding intent and its order within the bundle. Specifically, ATOs can be traced using tags like (bundleHash, bundleOrder).

Here, the bundleHash

represents the bundle to which the ATO belongs, while the bundleOrder

indicates the ATO's position within that bundle.

These tags subsequently guide the driver in collating all the ATO solutions into their respective bundles. Once aggregated, the driver then returns these consolidated solutions to the client, presenting them as resolutions for their specific intents.

Additionally, the solvers are held in check through a reputation score, maintained and updated on-chain via the protocol contract. This score undergoes revision with each victorious solution provided by a solver. Not only does this reputation score aid the driver in the incentivisation process for solvers, but it also serves as a performance benchmark. Should a solver's reputation plummet below a set threshold, they risk expulsion from the system.

## Solver Incentivisation: Rewarding solver for their computation

We are exploring various strategies for solver incentivisation and have identified several potential approaches. We remain open to dialogue and welcome suggestions for alternative methods. Here are some of the methods we're considering:

Transaction Fee Model

Drawing inspiration from platforms like CowSwap, this method involves providing users with an estimated transaction fee (likely determined and returned by the Driver). Based on this estimate, we can then deduct the requisite fees from the user's initiated operation.

Prepaid Balance System

Envisioned as a "gas tank", this approach allows users to deposit funds in advance, effectively prepaying for access to our infrastructure. As users initiate operations, the associated fees for solving their intents are automatically deducted from this pre-deposited balance.

Intent Execution Module

We propose the integration of a specialised module within the SCW. In this setup, when a user submits an intent to the driver RPC, they receive a unique hash. This hash serves as a reference linking to the addresses of the winning solvers. For on-chain execution of the intent, users must invoke the module's function, inputting the relevant hash. The module then liaises with the driver to obtain a fee quotation. Once determined, the fee, aligning with the provided quote, is dispatched to the driver via module for distribution. Following successful fee transfer, the driver then supplies the module with the calldata for the bundled transaction for facilitating the execution.

## MSCA: Module facilitating payments and executions

We are in the development phase of a specialised4337 compatible module designed to enhance smart contract wallets by introducing the capability of intent-based transaction execution. Many companies like Rhinestone, Safe, Biconomy etc. are designing modular smart contract wallets, and the goal will be to make it compatible with their architecture. This module is envisioned to streamline the process of executing user intents, calldata retrieval while also seamlessly integrating a mechanism for solver incentivisation. Here's a breakdown of the pivotal roles and functionalities the module aims to offer:

1. Module Activation

: * Upon activation, the module will signal that the associated smart contract wallet is now equipped to support intent-based transaction execution. This declaration serves as a green flag for external entities, ensuring compatibility and readiness for intent-based interactions.

1. Upon activation, the module will signal that the associated smart contract wallet is now equipped to support intent-based transaction execution. This declaration serves as a green flag for external entities, ensuring compatibility and readiness for intent-based interactions.

2. Quotation and Fee Handling

: * The module will incorporate predefined methods that facilitate the retrieval of fee quotations necessary for intent resolution. Before the intent's execution, module will automatically deduct the quoted fee from the smart contract wallet's balance. Subsequently, this amount will be channeled to designated driver contracts, serving as an incentive for solvers.

1. The module will incorporate predefined methods that facilitate the retrieval of fee quotations necessary for intent resolution. Before the intent's execution, module will automatically deduct the quoted fee from the smart contract wallet's balance. Subsequently, this amount will be channeled to designated driver contracts, serving as an incentive for solvers.

2. Intent Execution

: * One of the core functionalities of the module is to actualise the execution of user intents. This encompasses processing the intent, converting it into actionable transactions, and ensuring their proper execution using the method exposed by the module.

1. One of the core functionalities of the module is to actualise the execution of user intents. This encompasses processing the intent, converting it into actionable transactions, and ensuring their proper execution using the method exposed by the module.

2. CallData Retrieval

: * To ensure the accurate and complete realisation of user intents, the module will possess the capability to gather the necessary calldata for the user intent from the driver.

1. To ensure the accurate and complete realisation of user intents, the module will possess the capability to gather the necessary calldata for the user intent from the driver.

# Conclusion

In this exploration of our intent-based architecture, we have examined various components, including the solver network, mempool, the representation of intents as ATO, and the nuances of solver incentivisation, among others. While we have mapped out certain aspects, there are still elements, such as the mechanisms for incentivisation and solution evaluation, where the finer details are under deliberation.

In future, the proposed solution can be integrated in various ways, contingent upon the ATO generation method. At the heart of our infrastructure lies the ambition to aggregate diverse constructs—be it liquidity, tokens, or data—across multiple domains. By integrating a wide range of solvers, we aim to handle an expansive variety of operations, ensuring a seamless and efficient experience for the end-user.

One should take into account that this document might potentially possess some inaccuracies, as its main purpose lies in community feedback.

Feel free to reach-out to us on telegram @creator5923

and @rishotics

for further discussion