Milestone 1 Report: Smart Contract Analysis for Cardano DEX Analytics Tool

MuesliSwap Team

December 29, 2024

Abstract

This document, corresponding to Milestone 1 of our Cardano Catalyst project, provides an in-depth analysis of the smart contract architectures used by several major Cardano DEX protocols. By examining their datums, transaction flows, and off-chain batcher implementations, we establish the foundational knowledge needed to build an open-source analytics tool that tracks, parses, and interprets DEX activity. The ultimate goal is to foster transparency and encourage innovation in the Cardano DeFi ecosystem, aligning with Catalyst's mission of driving Cardano's growth and adoption.

Contents

1 Introduction

The objective of our project is to develop a fully open-source decentralized exchange (DEX) analytics tool that supports all major Cardano DEX protocols. By providing users and developers with detailed data insights (e.g., trades, prices, volume, and other metrics), this tool aims to increase transparency and foster innovation in the Cardano DeFi ecosystem.

1.1 Context and Problem Statement

Current analytics solutions for Cardano DEXs are fragmented or proprietary. While many DEXs publish some data, there is no standardized tool that aggregates and analyzes this information from multiple platforms in one place. As a result:

- Users lack a comprehensive dashboard to view and compare different DEX markets.
- Developers find it challenging to understand the underlying smart contract datums and transaction patterns on each DEX, hindering innovation.
- Researchers lack robust tools for generating aggregated market insights.

Our project addresses these issues by providing a single open-source software package to analyze Cardano DEX data.

1.2 Scope of Milestone 1

This document corresponds to **Milestone 1: Smart Contract Analysis**. The primary goal of this milestone is to produce a comprehensive analysis of the smart contracts used by major Cardano DEX protocols. Specifically, we examine:

- The high-level mechanics of each DEX protocol.
- The datums (UTxO-level data structures) and transaction flows for typical actions (e.g., swaps, adding/removing liquidity).
- Details on batcher or aggregator implementations, if applicable.
- Notable protocol-specific nuances.

In later milestones, we will use this analysis to develop and refine our open-source analytics tool, ensuring it can track, parse, and interpret DEX data accurately.

2 Approach and Methodology

Our approach to analyzing the smart contracts and on-chain transactions for each DEX consists of the following steps:

1. **Protocol Review:** Understand the general design of each DEX, including Automated Market Maker (AMM) vs. Orderbook models, stable swaps, or hybrid approaches.
2. **On-Chain Data Gathering:** Collect real transaction samples from each DEX (e.g., from cardanoscan.io, cexplorer.io, or direct chain queries) to identify the structure of relevant UTxOs.
3. **Datum Inspection:** Examine the datum fields for liquidity pools, orders, or swaps, noting how each protocol encodes these data.
4. **Batcher Analysis (if present):** Investigate batcher scripts or aggregator transactions, identifying how they are triggered and how they update state.
5. **Documentation of Special Cases:** Identify any exceptions or special mechanics unique to each DEX, such as partial fills, multi-asset pools, or lock-release mechanics.

The sections below provide placeholders for our initial findings, which will be expanded upon with in-depth details in collaboration with the DEX teams and additional research.

3 DEX-by-DEX Smart Contract Analysis

In this section, we present an overview of the major Cardano DEX protocols. For each protocol, we summarize how the smart contracts work, how transactions and datums are structured, and what particular features stand out. These analyses will form the basis for building the relevant modules within our open-source analytics tool.

3.1 MuesliSwap Hybrid-DEX

3.1.1 Protocol Overview

MuesliSwap operates a *hybrid DEX* integrating both an orderbook system (currently in v2) and a liquidity-pool-based AMM. The orderbook architecture handles trading orders, which can be matched either directly with other users' orders or against available liquidity pools. This approach seeks to capture benefits of both the constant-product AMM model and a limit-order model, allowing for flexible combinations of liquidity provisioning and order-driven trading.

Design Considerations

- Users place trading orders into the v2 orderbook. These orders can be partially or fully matched by off-chain matchmakers.
- AMM-style liquidity can be provided via a specific contract that uses an order-batcher solution for deposits and withdrawals.
- Matching transactions can pair user orders with either other user orders or the liquidity pool's reserves (constant product, concentrated liquidity, or other variations).

Datum Structures Two main data types are relevant: one for the liquidity pool (`PoolDatum`), and one for the orders (`OrderDatum`).

```
data PoolDatum = PoolDatum
  { pdCoinA :: AssetClass
  , pdCoinB :: AssetClass
  , pdTotalLiquidity :: Integer
  , pdRootKLast :: Integer
  , pdProfitSharing :: Maybe ProfitSharing
  , pdSwapFee :: Integer
  , pdExtraFeeDenom :: Integer
  }
```

PoolDatum tracks which two assets form the pool, total liquidity (LP token supply), optional fields for profit sharing, and fee parameters. When liquidity is added or removed, updates to `pdTotalLiquidity` and other fields are performed.

The *order* side uses:

```
data OrderStep
  = Deposit { dMinimumLP :: Integer }
  | Withdraw { wMinimumCoinA :: Integer
             , wMinimumCoinB :: Integer
             }

data OrderDatum = OrderDatum
  { odSender :: Address
  , odReceiver :: Address
  , odReceiverDatumHash :: Maybe DatumHash
  , odStep :: OrderStep
  , odBatcherFee :: Integer
  , odOutputADA :: Integer
  , odPoolNftTokenName :: TokenName
  , odScriptVersion :: BuiltinByteString
  }
```

For deposits and withdrawals, a user creates a UTxO carrying an `OrderDatum` with `odStep` set to `Deposit` or `Withdraw`. The contract logic and an off-chain batcher enforce minimum or maximum amounts received.

3.1.2 Batcher and Off-Chain Components

- A batcher script processes `OrderDatum` UTxOs for pool-based actions (deposit or withdraw). After validating parameters, it updates `PoolDatum` accordingly.
- For trading, limit orders are locked under the v2 orderbook script. Off-chain matchmakers see if these orders can match each other or if a pool can fill them at the requested price.
- The matchmaker then constructs a transaction consuming one or more order UTxOs and, optionally, the pool UTxO. Any partially filled orders are re-created on-chain with updated amounts.

Special Notes

- The `odScriptVersion` field can signal backwards compatibility or expansions for new pool types (e.g., concentrated liquidity).
- Combining order-based trades with AMM liquidity allows users to either trade via standard limit orders or rely on the pool to fill an order at current prices.
- This integrated approach aims to provide deeper liquidity than purely AMM or purely orderbook-based solutions, while keeping partial matches feasible in a single transaction.

3.2 Minswap v1

3.2.1 Protocol Overview

Minswap v1 is an automated market-maker (AMM) DEX on Cardano using the standard constant-product model $x \cdot y = \text{const}$. LP tokens represent liquidity providers' shares in each pool, and fees accrue back into the pools. The protocol uses a batcher mechanism to process user swap requests and maintain concurrency.

3.2.2 Transaction and Datum Structures

A typical swap UTxO includes a short datum that defines:

- *Sender and Beneficiary* addresses (public key hash plus optional stake key).
- *Assets to Swap*: Token policy/name and amounts.
- *Batcher Fee* and a *Deposit* in ADA.

When a user places a swap, this UTxO is locked under the Minswap contract. The batcher can then match it against the pool reserves, calculating out amounts based on the constant-product formula, and finalize the swap in one transaction.

3.2.3 Batcher and Off-Chain Flow

The off-chain batcher collects user swap requests, checks the pool states, and executes the desired trades. This design ensures concurrency by separating the order creation (placing the swap) from the actual execution (batching multiple user swaps).

Differences and Limitations

- v1 only supports the basic constant-product formula. Other formulas or multi-asset strategies discussed in the whitepaper are out of scope for this version.
- There is no specialized on-chain price oracle or multi-pool routing at the contract level.

3.3 Minswap Stableswap

3.3.1 Protocol Overview

Minswap's stableswap module introduces specialized pools for stable assets with minimal slippage around a 1:1 peg. The contract logic adjusts the constant-product formula to concentrate liquidity near parity. This primarily benefits traders swapping one stable token for another.

3.3.2 Transaction and Datum Structures

Each stableswap UTxO generally encodes:

- *Sender* and *Receiver* data.
- *Stable Swap Step*, referencing which stable token is sold and the minimum output amount.
- *Batcher Fee* and *Deposit*.

Like the main AMM, a specialized pool datum holds the amounts of each stable token in the pool and an amplification factor to minimize slippage around the peg.

3.3.3 Batcher or Off-Chain Components

Similar to Minswap v1, stableswap depends on an off-chain batcher to consume swap UTxOs, compare them to the pool's reserves, and finalize trades. Liquidity providers can deposit or withdraw stable tokens.

Remarks

- Pools typically charge lower fees due to the reduced impermanent loss risk around stable pairs.
- Each stable pair (e.g., DJED/iUSD) has its own validator address and pool ID token.

3.4 Minswap v2

3.4.1 Protocol Overview

Minswap v2 expands on the original AMM approach by adding advanced transaction batching, higher throughput, and user-defined order types (e.g., limit, stop-loss). It also includes a more flexible fee mechanism. This version is compiled under Plutus v2, which can reduce script size and allow for more complex transactions.

Key Differences from v1

- Additional order constructors (e.g., `SwapExactIn`, `StopLoss`, `OCO`, partial fill logic).
- Higher concurrency, sometimes reaching dozens of swaps in a single batch transaction.
- More granular control over fees (0.05% up to 20%), with potential governance-based adjustments.

3.4.2 Transaction and Datum Structures

A Minswap v2 order datum:

- *Sender/Beneficiary* addresses.
- An integer *constructor* indicating the order type.
- *Asset Specs* for tokens being sold and purchased.
- *Batcher Fee* and a possible *deposit* to handle expiration or fill-or-kill.

With Plutus v2 optimizations, up to 36 orders can be aggregated in a single transaction, each referencing its own constructor logic.

3.4.3 Batcher or Off-Chain Flow

A batcher collects pending UTxOs, identifies which can match instantly, and executes them in one transaction. Unfilled or partially filled orders are re-locked with updated amounts.

Remarks

- The new flexible order system attempts to replicate more advanced user-defined trading conditions within the constraints of eUTxO.
- The codebase aims to reduce storage overhead by using reference inputs, cutting down repeated scripts in each transaction.

3.5 SundaeSwap v1

3.5.1 Protocol Overview

SundaeSwap v1 is a constant-product AMM DEX on Cardano, introduced in early 2022. Similar to Uniswap's $x \cdot y = k$ formula, each pool has two tokens, and the user can swap one token for the other while incurring a fractional fee. The code was adapted to Cardano's eUTxO model with concurrency challenges mitigated by an off-chain batcher (often called a "scooper").

3.5.2 Transaction and Datum Structures

A typical SundaeSwap v1 swap UTxO uses a four-field datum:

1. *Pool Identifier* (which pair is being traded).
2. *Trader Data*, including a public key hash.
3. *Scooper Fee*, a small lovelace payment to the batcher.
4. *Token Amounts and Swap Direction*, specifying which token is sold vs. bought and in what amounts.

Liquidity pools also lock a *pool datum* with references to the pair of tokens, the current supply of LP tokens, and a fee. Additional tokens (e.g., a "factory token") ensure the pool is identified as part of SundaeSwap.

3.5.3 Batcher or Off-Chain Flow

A "scooper" repeatedly scans the chain for open swap requests and matches them against the pool. It then updates the pool's token balances in a single transaction and distributes outputs to the traders.

Remarks

- Each liquidity pool is controlled by a single UTxO, potentially creating concurrency bottlenecks if many swaps occur simultaneously.
- Future updates might allow migrating this liquidity to new contracts via an "allow list" mechanism.

3.6 SundaeSwap v3

3.6.1 Protocol Overview

SundaeSwap v3 rewrote the contracts in Aiken for lower fees and better concurrency. The DEX can process more orders per "scoop," and the fee mechanism has been updated. A user-friendly migration tool enables liquidity providers to move from older contract versions.

Differences from v1

- Higher throughput (the code aims for up to 35 orders in a single scoop).
- A dynamic fee approach dividing fees into a base portion plus an incremental portion per order.
- Simplified migration for older liquidity, letting LPs shift positions with minimal overhead.

3.6.2 Transaction and Datum Structures

Swap orders are more detailed, containing, for example, a `max_protocol_fee`, a `destination` indicating whether tokens should return to the user or go elsewhere, and a `details` object specifying input and output tokens. Pools themselves track additional data such as `bid_fees_per_10_thousand`, `ask_fees_per_10_thousand`, and more flexible fee logic.

3.6.3 Batcher or Off-Chain Flow

A "scooper" still processes the UTxOs, but now can batch significantly more orders in a single transaction. The `max_protocol_fee` field ensures that if fees are too high, the user's order is not executed.

Remarks

- The Aiken-based approach focuses on script size efficiency.
- Up to six scoops per block times 35 orders each suggests a large theoretical daily maximum throughput, though real-world performance depends on network conditions.

3.7 WingRiders v1

3.7.1 Protocol Overview

WingRiders v1 is an AMM-based DEX on Cardano using a standard $x \times y = \text{const}$ formula. The protocol handles concurrency with user "request" UTxOs or batch transactions. It also considers ADA staking in its design, though the exact on-chain structure for that is partially outlined.

3.7.2 Transaction and Datum Structures

- *Swap Datums*: Typically contain trader properties (public key hash, optional beneficiary) and fields for determining the swap direction and amounts.
- *Pool Datums*: Track the pair of tokens, total liquidity, and a *fee* rate. They can also store extra amounts designated as "treasury" or "reserve."

3.7.3 Batcher or Off-Chain Flow

WingRiders v1 uses an off-chain agent that can combine multiple user orders in one transaction. Each order references which tokens are sold or expected. The batcher updates the pool's state after the swap, ensuring concurrency is not limited to one user per block.

Remarks

- Some datums include a `deadline` to limit how long an order remains valid.
- ADA staking from the liquidity pool portion has been proposed, though further on-chain code details may be introduced in new versions.

3.8 WingRiders v2

3.8.1 Protocol Overview

WingRiders v2 refines the AMM approach by supporting more advanced features around fee structures, concurrency, and staking/farming. The contract tracks additional treasury parameters, allowing more flexible distribution of rewards or fee splitting.

Key Differences from v1

- More granular "treasury" tracking (e.g., `treasury_a`, `project_treasury_a`, `reserve_treasury_a`).
- Extended concurrency handling, letting a batcher handle a higher volume of user requests in a single block.
- Staking or yield-farming logic is further developed, enabling more epoch-based reward schemes.

3.8.2 Transaction and Datum Structures

- *Swap Orders*: Contain user address info, tokens being swapped, a `buy_amount`, and an `oil` field (the batcher fee).
- *Pool Datums*: Use fields like `swap_fee_in_basis` and `fee_basis`, plus references to tokens and treasury fields, to compute the actual swap outcome.

3.8.3 Batcher or Off-Chain Flow

As in v1, the batcher consumes user swap orders in bulk. The new "oil" fee helps reimburse aggregator transactions. Once processing is complete, the updated pool UTxO is created, reflecting new reserves.

Remarks

- Governance improvements may allow DAO-like oversight on how treasury fields are used or distributed.
- The batcher logic is similar to v1, but with additional concurrency optimizations to handle more complex transaction loads.

3.9 VyFi

3.9.1 Protocol Overview

VyFi is a Cardano-based DEX that uses a core AMM approach for swapping tokens. Additional features such as yield farming or staking are available but are only loosely related to the DEX's on-chain swap logic. Users can supply liquidity to create or enhance pools, then place swaps referencing these pools.

3.9.2 Transaction and Datum Structures

- A "swap order" is locked under the VyFi order validator address. The user's public key hash and staking key hash are combined in a single *trader bytes* field, along with a `constructor` determining the swap direction.
- The protocol references an external configuration service to know which pair of tokens the user is swapping (e.g., `token_a`, `token_b`), plus a batcher fee.
- Each liquidity pool UTxO stores the two tokens, relevant liquidity information, and a *main NFT* that identifies the pool.

Batcher or Off-Chain Processing An aggregator service monitors user swap requests. It checks the `constructor` to see if the user sells `token_a` for `token_b` or the reverse, then merges that with the pool's reserves in a single transaction. If multiple user requests can be processed, they can be aggregated, subject to network constraints.

Remarks Swapping in VyFi can be extended with additional functionalities like *farm* or *BAR staking*, but that is separate from the core DEX logic. The DEX itself uses a standard approach of referencing a pool UTxO and user order UTxO in the same transaction.

3.10 Spectrum v1

3.10.1 Protocol Overview

Spectrum v1 aims at cross-chain interoperability by maintaining a sidechain-like environment on Cardano. The system envisions "shards" or committees for each external chain, notarizing events (deposits, withdrawals) and aggregating them into a super ledger (L^+).

3.10.2 Transaction and Datum Structures

- *Cells and Value Transfers*: Spectrum tracks user balances as *cells*, which can be active or terminal. Active cells can be mutated on-chain, while terminal cells indicate tokens to be exported back to another chain.
- *Swap Datum Example*: `PlutusSpectrumOrder` includes the base and quote tokens, associated fee parameters, and user addresses for receiving leftover tokens or rewards.

Batcher or Off-Chain Components

- Each external chain has a committee that notarizes on-chain events. Reports are broadcast to all committees and eventually consolidated into the Spectrum ledger.
- An agent or relay collects these "reports," enabling cross-chain state changes, such as bridging tokens or finalizing liquidity positions.

Remarks

- Spectrum's cell-based approach allows partial reverts if a source chain transaction is rolled back before being fully finalized.
- The global clock sync mechanism coordinates epoch boundaries across connected chains, limiting potential forks or timing attacks.

4 Preliminary Observations

Although specific details will be filled in as our investigation deepens, we note a few preliminary observations from our early analysis:

- Varied Datum Formats: Each DEX structures its datums differently. A unified parsing strategy will need modular designs per DEX.
- Batcher Necessities: Many protocols rely on off-chain or aggregator scripts that periodically update on-chain states. Understanding scheduling and logic is vital for accurate analytics.
- Concurrency Solutions: Some DEXs implement concurrency mechanisms (like multiple UTxOs or specialized scripts) to allow simultaneous interactions. The chosen method directly impacts how we track transaction histories.

5 Next Steps and Conclusion

This report marks the successful completion of **Milestone 1: Smart Contract Analysis**. We have laid the groundwork for understanding the various smart contract architectures across major DEXs on Cardano. In the next milestones, we will:

- Translate these findings into a common codebase that can parse, interpret, and store DEX transaction data in a well-structured format.
- Develop a fully functional analytics API that provides endpoint access to data such as trades, user histories, price histories, and volume analytics.
- Perform system testing, deployment scripting, and integrate additional analytical features (e.g., arbitrage transaction detection).

By documenting and analyzing these protocols, we aim to build an open-source reference for Cardano developers and users to easily monitor and study DeFi activity. This shared knowledge supports a more transparent and innovative ecosystem.