Jur Arbitration Contract Specification – MVP

Introduction

Jur A.G. intends to introduce JUR, an ERC20 compliant token on the Ethereum blockchain.

The JUR token will be used:

- as the currency in which escrow is denominated in a Jur Agreement Contract.
- to provide utility to resolve disputes, should they arise, in any Jur Agreement Contracts.

Use Case

Smart contracts running on the Ethereum blockchain allow simple escrow agreements, where funds are held by a smart contract, and then delivered to one or more parties based on a pre-determined set of codified rules.

However, for many real life cases, it is not possible to codify (within a Turing complete language) the rules which determine how and when escrow funds should be dispersed.

For example, suppose that Alice wishes to purchase a copywriting service from Bob, to write three articles on her new product. She agrees to pay 100 JUR tokens per article, on completion of all three articles, so 300 JUR tokens in total.

Alice and Bob would create a Jur Agreement Contract (JAC), attach the above agreement to the contract, and Alice would deposit 300 JUR tokens into the JAC, to be held in escrow until Bob completes the assignment.

There are then three possible outcomes:

1. *No dispute:* Bob completes the work, and delivers all three articles. Alice will then authorise the JAC to disperse the 300 escrowed JUR tokens to Bob, and the agreement is completed.

2. *Dispute resolved between Alice & Bob:* Bob is only able to complete a single article due to unexpected commitments. Alice proposes that instead of paying the original 300 JUR tokens, she will only pay 100 JUR to Bob for his services. Bob accepts this proposal, and the JAC refunds 200 JUR tokens to Alice, and disperses 100 JUR tokens to Bob.

3. *Dispute resolved by JUR token holders:* Again, Bob is only able to deliver a single article. In this scenario Alice is upset as she feels her time has been wasted, and only wants to pay Bob 50 JUR tokens for his services for the single article. Bob does not accept this proposal as he feels he should be paid 100 JUR tokens. Since they cannot agree, the JAC enters a dispute state, and any JUR token holders (including Bob & Alice) can vote on whose proposal they feel is most just. If Alice's proposal ends up with a majority of votes at the end of the voting period, then the JAC respects this proposal and refunds her 250 JUR tokens, and disperses 50 JUR tokens to Bob. Conversely if Bob's proposal receives a majority of votes the JAC will refund Alice 200 JUR tokens, and disperse 100 JUR tokens to Bob.

Initialising the Jur Agreement Contract

In order to initialise a Jur Agreement Contract, two parties (Party A and Party B) must agree on the following state:

- Party A (encoded as an Ethereum address)
- Party B (encoded as an Ethereum address)
- Agreement (details of the contractual agreement between Party A and Party B, encoded as a hash of arbitrary data stored off-chain)
- Dispersal (encoded as a pair [disperseA, disperseB] denominated in JUR, where disperseA / disperseB are the amounts returned to Party A and B respectively on completion of the contract)

- Escrowed Funds (encoded as a pair [fundingA, fundingB] denominated in JUR, where fundingA / fundingB are the amounts to be funded by Party A and B respectively)

Once the JAC has then been funded by Party A and Party B (as specified in Escrowed Funds) it is considered to be initialised, and funds are locked. Until this time, either Party may withdraw from the agreement.

The smart contract enforces the following initial constraints:

- Both Party A and Party B must have both signed off on the Agreement and Dispersal.
- The amount of Escrowed Funds must match the agreed Dispersal, i.e. (fundingA + fundingB) == (disperseA + disperseB).

At any time, following initialisation, if both parties sign off, the JAC will respect the agreed Dispersal, and distribute its escrowed funds accordingly.

Changing the Dispersal

At any time, following the initialisation of a Jur Agreement Contract, either party can propose a Amended Dispersal, encoded as a pair [amended_disperseA, amended_disperseB].
Alongside the proposed Amended Dispersal, the party proposing the Amended Dispersal must also submit a proposed Amended Escrowed Funds, encoded as a pair [amended_fundedA, amended_fundingB]. This is to account for any possible excess or deficit in the current Escrowed Funds held by the JAC relative to the new Amended Dispersal.
For a proposed Amended Dispersal to be considered finalised, both parties must agree to the Amended Dispersal, and the contract must be funded according to the Amended Escrowed Funds.

In order to maintain fairness, all deficits in funding must be paid into the JAC, before any excesses in funding can be withdraw. At any time before a proposed Amended Dispersal is finalised, either party that has funded a deficit in the new Amended Escrowed Funds, can withdraw this funding.

Example 1:

Suppose that Alice has originally agreed to pay Bob 300 JUR tokens, so the originally agreed Dispersal was [0, 300], and Alice has paid 300 JUR tokens into escrow in the JAC, so the originally agreed Escrowed Funds was [300, 0].

Bob is now only delivering half of the agreed services, so proposes a new Amended Dispersal of [0, 150] with an Amended Escrowed Funds of [150, 0].

Alice agrees with this Amended Dispersal, and withdraws 150 JUR tokens from the JAC. At this point the Amended Dispersal / Amended Escrowed Funds are considered finalised, and recorded as the new Dispersal and Escrowed Funds of the JAC.

Example 2:

Suppose that Alice has originally agreed to pay Bob 300 JUR tokens, and that Bob has agreed to stake 300 JUR tokens against this as a bond. The original Dispersal is then [0, 600] and the original Escrowed Funds is [300, 300].

Bob and Alice have now agreed that Alice will pay a further 100 JUR tokens, and further more that Bob's bond should be reduced to 200 JUR tokens. So the Amended Dispersal is [0, 600], and the Amended Escrowed Funds is [400, 200].

In this case, Alice will need to fund the JAC with an additional 100 JUR tokens, at which point Bob can then withdraw 100 JUR tokens. After this, the Amended Dispersal / Amended Escrowed Funds are considered finalised, and recorded as the new Dispersal and Escrowed Funds of the JAC.

This process maintains JAC invariant of:

(fundingA + fundingB) == (disperseA + disperseB)

Entering into a Dispute

At any time, following the initialisation of a Jur Agreement Contract, either party can unilaterally move the JAC into a Dispute state.

To move a JAC into a Dispute state, the disputing party must provide a Dispute Dispersal. This Dispute Dispersal will be respected by the JAC if the disputer wins a majority of votes.

For example, suppose, as above, that Alice had agreed to pay Bob 300 JUR tokens, but Bob has only completed 50% of the work. Alice can place the JAC into

a Dispute state, with a Dispute Dispersal of [150, 150]. If she then wins the majority of votes, the JAC will disperse 150 JUR tokens to Alice, and 150 tokens to Bob.

Following a JAC being placed into a Dispute state by a given party, the other party then has 24 hours to respond with their Dispute Dispersal. If no response is received in 24 hours, the default Dispute Dispersal for a given party is to send all funds to that party.

For example, as above, Alice places the JAC into a Dispute state, with a Dispute Dispersal of [150, 150]. After 10 hours, Bob responds with his Dispute Dispersal of [100, 200] (as he feels his work is actually 66% complete, not 50% complete). If Bob were to not respond within 24 hours, his defaulted Dispute Dispersal would be [0, 300].

A JAC which has been in a Dispute state for 24 hours, therefore has the following state:

- Party A (encoded as an Ethereum address)
- Party B (encoded as an Ethereum address)
- Agreement (details of the contractual agreement between Party A and Party B, encoded as a hash of arbitrary data stored off-chain)
- [Original or Amended] Dispersal (encoded as a pair [disperseA, disperseB] denominated in JUR)
- Party A Dispute Dispersal (encoded as a pair [partyA_dispute_disperseA, partyA_dispute_disperseB]denominated in JUR)
- Party B Dispute Dispersal (encoded as a pair [partyB_dispute_disperseA, partyB_dispute_disperseB]denominated in JUR)

The smart contract will enforce that for both Dispute Dispersals, the total of each Dispute Dispersal is equal to the amount of Escrowed Funds.

i.e. (fundingA + fundingB) == (partyA_dispute_disperseA + partyA_dispute_disperseB) and (fundingA + fundingB) == (partyB_dispute_disperseA + partyB_dispute_disperseB).
Once both parties have provided a Dispute Dispersal (either explicitly within 24 hours, or if not by default after 24 hours), voting begins.

In addition to the above, when a party puts the JAC into a dispute state, they must additional vote on their own Dispute Dispersal. They can determine how much, or how little to vote, but the vote must be at least 1% of the Escrowed Funds.

Dispute Resolution

Once a JAC enters a Dispute state, JUR token holders will vote on one of three options:

- Party A Dispute Dispersal
- Party B Dispute Dispersal
- Reject

If either of the first two options ends up with a majority of votes, the JAC will disperse funds according to the respective Dispute Dispersal. If the Reject option ends up with the majority vote, then the JAC will disperse funds back in to Party A and Party B in amounts that match the Escrowed Funds (so Party A will receive `fundingA` and Party B will receive `fundingB`).

Voting Process

In the following sections, a vote is defined to be a JUR token staked to the JAC contract, and associated with one of the three above options. For example, if a voter stakes 20 JUR tokens to the Reject option, they are considered to have placed 20 votes on the Reject option.

The first vote in a dispute resolution, is always placed by the party (either Party A or Party B) who triggers the dispute, and as above, this must be at least 1% of the current Escrowed Funds (i.e. `Initial Vote >= 0.01 * (fundingA + fundingB)`).
Within the Voting Period, any holder of JUR tokens can vote on one of the above three options.

The rules governing the voting process are:

- any new voter must stake at least 1% of the total amount of votes accrued during the voting process so far.
- the Voting Period lasts 24 hours from the Dispute Dispersals being finalised, unless extended.
- the Voting Period is extended if, at the end of the Voting Period, either of the following is true. In both cases, the Voting Period is extended by a further 30 minutes, after which this logic is reapplied recursively:

    - more than 5% of the total votes placed during the Voting Period occurred in the last 30 minutes.
    - there is no option with a clear majority.
- a clear majority is considered to be when one single option has more than any other option.

Rewarding Voters

In order to incentivise JUR token holders to participate in resolving JAC disputes through voting, the JAC enforces a voter reward scheme.

At the end of the Voting Period, there will be one majority option (the option which received most votes), and two minority options.

Any votes placed on either of the two minority options are lost to the voters who placed them.

Any votes placed on the majority option are refunded back to the voters who placed them, and those voters may receive an additional reward calculated as below.

Note - if no votes are placed on minority options (i.e. all votes are placed only on the majority option), then there will be no additional reward for voters, and all voters would receive back their votes.

The reward for a majority voter, at the end of the Voting Period, is calculated as follows.

At the end of the Voting Period, we define:

Let majority_option be the the option that receives most votes at the end of the Voting Period.
Let minority_option_best be the option that receives the second most votes at the end of the Voting Period (if both minority options receive equal votes, this is chosen arbitrarily between the two minority options).
Let sum_votes_all_minority be the sum of all votes placed on the two minority options.
Let sum_votes_best_minority be the sum of all votes placed on minority_option_best.
Let sum_votes_majority be the sum of all votes placed on majority_option.
Let reward_multiplier be sum_votes_all_minority / (sum_votes_best_minority).

For our specific majority option voter we define:

Let user_start_votes_majority be the sum of all majority option votes before this voter placed their votes.
Let user_votes_majority be the amount of votes placed on the majority option by this voter.
Then, our voter is rewarded as follows:

If user_start_votes_majority + user_votes_majority <= sum_votes_best_minority then the voter receives user_votes_majority * reward_multiplier.
If user_start_votes_majority >= sum_votes_best_minority then the voter receives no reward.
If user_start_votes_majority < sum_votes_best_minority and user_start_votes_majority + user_votes_majority >= sum_votes_best_minority then the voter receives (sum_votes_best_minority) - user_start_votes_majority) * reward_multiplier.
In all cases, the voter receives back all of their voted tokens, plus any additional reward calculated as above.

Voters who placed votes on either of the two minority options lose their voted tokens, and receive no additional reward.

For voters who voted on the final majority option, they are eligible to withdraw their original votes (as JUR tokens) plus an additional reward.

However they can only withdraw their tokens 24 hours after the Voting Process completes.

Voting / Staking

To vote on the outcome of a bet, an JUR token holder must stake their tokens to the smart contract. This ensures that the same token can't be used to vote on multiple outcomes (since the voting token holder loses control of their tokens for the duration of the Voting Process).

The number of votes cast is equal to the number of JUR tokens that the voter stakes to the smart contract.

To transfer tokens to the JAC we will use the ERC223 tokenFallback method.

Security

In order to reduce attack surfaces, the contract should:

- conform to Solidity / Ethereum best practices.
- be independently audited by at least two Solidity experts.
- have a bug bounty process open to the public running for at least two weeks.
- only hold JUR tokens, and not Ether (enforced through the fallback function).
- use best practices in terms of staking (see Voting / Staking section).
- use pull rather than push for all dispersals.
- employ a factory contract to deploy all Jur Agreement Contracts.

NB - this section does not consider game theoretic attacks on the Jur Agreement Contracts.