Smart Legal Contracts: A Standardized Approach

Houman Shadab
Co-Director, the Accord Project
houman@accordproject.org
What is the Accord Project?

Sets the legal and technical foundation for **smart legal contracts** by interfacing with leading lawyers, industry organizations, and technologists.

Addresses the lack of a common approach for smart legal contracts and the widely divergent, potentially incompatible, approaches that are emerging.

Producing **open source core** for smart legal contracts that embodies a collaborative techno-legal foundation and meets the needs of the legal industry.

IN COLLABORATION WITH

[Clio] [IEEE] [IACCM] [ISO] [Hyperledger]
## Working Groups

### Supply Chain
MSAs and ancillary documents, tracking data standardization, upstream vs. downstream coordination, real-time system integration, secure data exchange, supply chain visibility, IoT standards

### Financial Services
Real-time incorporation of market data, dynamic pricing and collateralization, fund structures, clearing and settlement infrastructure, claim types, coverage adjustment, use of telematics

### Intellectual Property
Automating digital rights management, IP registration in a global database, automating the grant, refusal, termination, and assignment of IP, incorporation of real-time data about infringement

### Venture and Token Sales
Automation and integration of investment documents and connection to milestones; integration with equity holding platforms; automation of various forms of blockchain token sales and governance frameworks

### Dispute Resolution
Preventing and resolving disputes involving smart legal contracts; divergence between law and code; automated and distributed dispute resolution; smart ADR clauses; relationship with online dispute resolution.
Members and partners...

Baker McKenzie

Mishcon de Reya

Holland & Knight

McDermott Will & Emery

BakerHostetler

CMS

Freshfields

Linklaters

CLYDE&Co

DENTONS

SLAUGHTER AND MAY

Standard Chartered

Norton Rose Fulbright

...and many more.
Goals of the Accord Project

Open Source Community

Grow a community to develop freely available code, documentation, and other deliverables supporting the use of smart legal contracts globally across a wide variety of industries, use cases, and platforms. Subject to the Apache-2 license to ensure that individuals and companies have wide latitude in using the code for commercial, educational, and private purposes.

Smart Legal Contract Templating and Modeling

Develop a universally accessible and widely used open source library of modular smart legal contract and smart clause templates and models that reflect input from transactional attorneys and other experts that meets the needs of technology-enabled enterprises and specific business requirements. Built according to the Cicero specification.

Legal Contracting Language

Develop a domain specific language for smart legal contract execution that is accessible to non-technical professionals, compatible with a variety of execution targets such as SaaS platforms and distributed ledgers, and meets security, modularity, and other requirements. Built according to the Ergo language specification.
What is a Smart Legal Contract?
Built on Three Pillars

Legal Expertise

Technical Specifications

Open Source Software
Step-by-step

Progressive migration/evolution of existing legal and contract management practice:

1. Text

2. Text with digital signature

1. Text with variables (a model!) with digital signature

2. Text with variables and logic, with digital signature
   a. Automated handling of notifications and contract obligations

3. Distributed execution of contractual logic
Late Delivery and Penalty. In case of delayed delivery\["\text{except for Force Majeure cases,}\]: the Seller shall pay to the Buyer for every \{\text{penaltyDuration}\} of delay penalty amounting to \{\text{penaltyPercentage}\}\% of the total value of the Equipment whose delivery has been delayed. Any fractional part of a \{\text{fractionalPart}\} is to be considered a full \{\text{fractionalPart}\}. The total amount of penalty shall not however, exceed \{\text{capPercentage}\}\% of the total value of the Equipment involved in late delivery. If the delay is more than \{\text{termination}\}, the Buyer is entitled to terminate this Contract.
Model

concept SupplyModel {
 /** Does the clause include a force majeure provision? */
 o Boolean forceMajeure optional

 /* For every penaltyDuration that the goods are late */
 o Duration penaltyDuration

 /* Seller pays the buyer penaltyPercentage % of the value of the goods */
 o Double penaltyPercentage

 /** Up to capPercentage % of the value of the goods */
 o Double capPercentage

 /* If the goods are >= termination late then the buyer
 may terminate the contract */
 o Duration termination

 /* Fractional part of a ... is considered a whole ... */
 o TemporalUnit fractionalPart
}

/** Does the clause include a force majeure provision? */

/* For every penaltyDuration that the goods are late */

/* Seller pays the buyer penaltyPercentage % of the value of the goods */

/** Up to capPercentage % of the value of the goods */

/* If the goods are >= termination late then the buyer
 may terminate the contract */

/* Fractional part of a ... is considered a whole ... */
Logic

```plaintext
contract SupplyAgreement over SupplyModel {
    clause lateDeliveryAndPenalty(request: Request): Response {
        // Guard against force majeure
        enforce !contract.forceMajeure;
        define constant penalty =
            (diff / contract.penaltyDuration.amount)
            * contract.penaltyPercentage / 100.0 * request.goodsValue;

        // Penalty may be capped
        define constant capped =
            min([penalty,
                contract.capPercentage * request.goodsValue / 100.0]);

        // Return the response with the penalty
        // and termination determination
        return Response {
            penalty: capped,
            buyerMayTerminate: diff > contract.termination.amount
        }
    }
}
```

http://github.com/accordproject/ergo
Programming Model

Late Delivery and Penalty. In case of delayed delivery[[" except for Force Majeure cases,"?: forceMajeure]] the Seller shall pay to the Buyer for every [[penaltyDuration]] of delay penalty amounting to [[penaltyPercentage]]% of the ...

Request

Contract Template

Contract Parameters

State

Contract Obligations

Response

Contract Creation

{ "forceMajeure" : false,
  "penaltyDuration" : { amount :2,
                        unit : "days" },
  "penaltyPercentage" : 10.5,
  ... }

Contract Execution

{ "agreedDelivery": "December 17, 2017 23:59:00",
  "deliveredAt": "December 18, 2017 00:24:00",
  "goodsValue": 200.00 }

{ "penalty": 110.00000000000001,
  "buyerMayTerminate": true }
Ergo’s Goals

Safety

Efficiency

Usability

Openness

A good smart contract language is a $1 billion problem.

Why? Look at the amounts lost in some recent hacks:

- Parity - $300 million
- DAO - $50 million
- PoWHCoin - $1 million
Programming Model in Ergo

clause ::
  Request × State →
  ( Response × State
   × Obligation[] )
| Error

clause late(req : LateRequest)
  : LateResponse {
    emit BillingObligation
    {amount: req.weeks * 5.0};
    enforce req.weeks > 0.0
    else throw CheatError{};
    set state PenaltyPaid{};
    return LateResponse{};
}
call late(LateRequest{weeks:2.0});
Why a New Language?

- *Domain specific* meant for legal contract logic
- Integral with Accord Project specification: CML and Templates
- Ease of use for legal-tech (template) developers
- Portable, compiles to various runtimes (e.g., nodejs) or DLTs (e.g., Fabric, EVM)
- Formally specified, no run-time errors, all contract calls terminate, deterministic
- Suitable for analysis & verification (contract property, cost bounds)
- “Modern language”: Distributed as Node.js package, Tooling (mode for various code editors, REPL), Documentation, Modularity, Error reporting, Performance…
Ergo Contracts as Classes

```
contract Safte over SafteContract {
    clause tokenSale(request : TokenSale) : TokenShare {
        let discountRate = (100.0 - contract.discount) / 100.00;
        let discountPrice = request.tokenPrice * discountRate;
        return TokenShare{ tokenAmount : contract.purchaseAmount / discountPrice }
    }

    clause equityFinancing(request : EquityFinancing) : EquityShare {
        let discountRate = (100.0 - contract.discount) / 100.00;
        let discountPrice = request.sharePrice * discountRate;
        return EquityShare{ equityAmount : contract.purchaseAmount / discountPrice }
    }

    clause dissolutionEvent(request : DissolutionEvent) : PayOut {
        return PayOut{ amount : contract.purchaseAmount }
    }
}

call dissolutionEvent(DissolutionEvent{ cause : "Cold feet" });
call tokenSale(TokenSale{ tokenPrice: 3.14 });
call equityFinancing(EquityFinancing{ sharePrice: 2.98 });
```

https://blog.colony.io/a-simple-agreement-for-future-tokens-or-equity-b8ef08608347
Ergo Contracts as Rules

https://blog.colony.io/a-simple-agreement-for-future-tokens-or-equity-b8ef08608347

```latex
contract Safte over SafteContract
  rule tokenSale when TokenSale do
    let discountRate = (100.0 - contract.discount) / 100.00;
    let discountPrice = request.tokenPrice * discountRate;
    return TokenShare{ tokenAmount : contract.purchaseAmount / discountPrice };

  rule equityFinancing when EquityFinancing do
    let discountRate = (100.0 - contract.discount) / 100.00;
    let discountPrice = request.sharePrice * discountRate;
    return EquityShare{ equityAmount : contract.purchaseAmount / discountPrice };

  rule dissolutionEvent when DissolutionEvent do
    return PayOut{ amount : contract.purchaseAmount };

send DissolutionEvent{ cause : "Cold feet" };
send TokenSale{ tokenPrice: 3.14 };
send EquityFinancing{ sharePrice: 2.98 };
```
Blockchain Agnostic

Corda blockchain’s IOU implemented as logic in Ergo

```
bash-3.2$ ergoc --target java examples/corda-iou/model.cto examples/corda-iou/logic.ergo
Compiled Ergo 'examples/corda-iou/logic.ergo' -- created 'examples/corda-iou/logic.java'
bash-3.2$ javac -cp backends/java/bin:backends/java/lib/* examples/corda-iou/logic.java
bash-3.2$ java -cp backends/java/bin:backends/java/lib/*:examples/corda-iou org.accordproject.ergo.RunErgo -request examples/corda-iou/request.json -state examples/corda-iou/state.json -contract examples/corda-iou/contract.json logic
{"left":{"response":{"$class":"org.accordproject.cicero.runtime.Response"},"state":{"$class":"org.accordproject.ct.cicero.contract.AccordContractState","stateId":"1"},"emit":[[]]}
bash-3.2$ java -cp backends/java/bin:backends/java/lib/*:examples/corda-iou org.accordproject.ergo.RunErgo -request examples/corda-iou/request-wrong.json -state examples/corda-iou/state.json -contract examples/corda-iou/contract.json logic
{"right":{"message":"The IOU's value must be non-negative.","$class":"org.accordproject.ergo.stdlib.ErgoErrorResponse"}}
bash-3.2$ 
```
Future Work: Contract Composition

- Most contracts include various standard “reusable” or “boilerplate” clauses

- Examples: Installment payments, interest calculations, jurisdiction, etc.

- What is the right model to compose clauses in Ergo?
  - Clauses = Traits?
  - Clauses = Rules?
Future Work: More on Verification

Typed Ergo programs should:
(a) always terminate
(b) without any runtime errors

The Good News
Ergo is written in Coq, and built on Q*Cert which gives us:
- Data model
- Type foundations
- Optimization framework
- Proofs!