Democracy of Things (DoT) Vol. 5: Progressive Cybernetics – Smart Contracts (& The Blockchain)

A technical survey of Blockchain smart contract functionality in the context of Java
Computer Science 501: Systems Development with Emerging Technology

M.Sc.(IS) Program, Athabasca University
Professor: Tim Reimer

Submitted on: April 4th, 2018
Author/Student: Mickael Yusufidis, Richmond Hill, ON ➔ Student ID 2210970

Table of Contents

1 History & Introduction

1.1 Origins

On October, 2008 the first Bitcoin academic paper was published by Satoshi Nakamoto, the title, "Bitcoin: A Peer-to-Peer Electronic Cash System", [1]. The first Bitcoin network released in 2009 was originally devised and designed as a distributed shared ledger to immutably record financial transactions between peers without the need for an intermediary, thus completely decentralized given no financial intermediary. This was the beginnings of what has now evolved into the platform and paradigm that is Bitcoin's underlying technology framework, Blockchain or more specifically, Blockchain networks.

In very short order, Blockchain has evolved into much more, as the industry and market begin to realize that while Blockchain was pre-eminently being used for recording crypto-currency transactions, the shared ledger, "could be used to record any transaction and track the movement of any asset whether tangible, intangible or digital." [2]

In 2013, Vitalik Buterin, founder of the Ethereum Foundation, published what has become an elusive white paper the beginnings of a concept to evolve the bitcoin protocol being used "for more than just money", paving the way for the introduction of the differentiating and disruptive Ethereum Blockchain platform as a service, "providing its users with near-infinite power through an all-encompassing mechanism called [smart] contracts." [3]

1.2 Introduction

This essay will not focus on discussing Blockchain in the context of crypto-currency. The focus is specifically to discuss the smart contract as a function and object of code in the context of the Java programming language. Nevertheless, a background will be provided on the underlying technology, Blockchain, for context, followed by an overview and demonstration of smart contracts, functionality and application(s).

2 Background, Theory and Principles

2.1 Blockchain Described

Blockchain in its very essence is distributed ledger technology (DLT), advanced database management used to record and share data within mirrored ledgers on nodes as a means of collectively controlling the immutable history of data objects or information stored in what is referred to as blocks. The act of transactions drives the Blockchain universe. A transaction is sent by a particular application that is associated to a particular node, for example a smart contract, to another specific node. Each node on the Blockchain network has an entire copy of the Blockchain database, multiple transactions can feed into a node. Nodes assemble various transactions into blocks of information. Nodes then run validation logic against blocks, which is known as the process of mining. The validation logic results in valid transactions being encoded, labeled and hashed; a Merkle tree process maintains links, or a chain, to previous blocks hashes all the way back to the original transaction, known as the genesis transaction, rendering the transaction history widely accessible and immutable. [2] [4] [5]

2.2 Smart Contracts, Tokens, Public and Private Blockchain Networks

Smart Contract(s) are essentially an application or more specifically an object on a Blockchain that is controlled by code (vs. a human being). The code instructs how the smart contract behaves. Smart contracts are also known as decentralized applications (dApps) given that they are freely governed by the code within them and thus emulate a governing entity based on the rules and regulations defined by the code. Most of the recognized applications of smart contract technology today are token driven, “a unit of value issued by a project or company.” [6] Tokens, like currency, can appreciate or depreciate in value based on demand for the object and the leap of faith required to understand the true potential of smart contracts is that “tokens are not just the currency used in the network, but they are a unit of the business model on which the (Blockchain) network is built on.” [6]

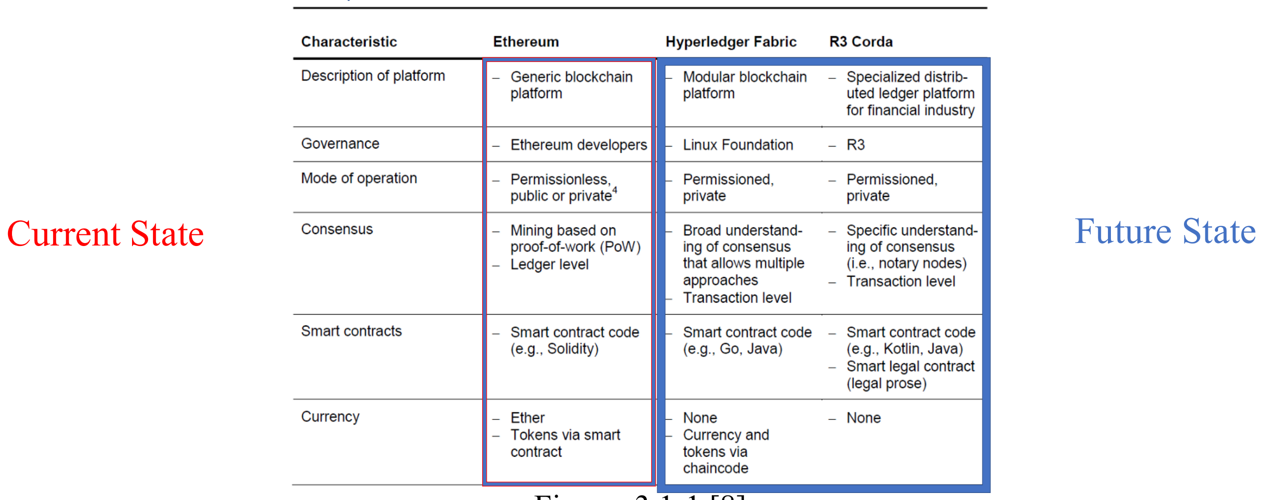
A notable standard of a public Blockchain and applications of smart contracts today is fungible assets, where smart contracts are tokenized and transactional exchange is facilitated for example by Ethereum’s Ether (tokens) as a common unit of transaction, representation of value or exchange, in essence a fungible asset, which “implies that two things are identical in specification, where individual units of the commodity or good can be mutually substituted,” [7]. This is, in essence, the most popular application of Blockchain and smart contract technology today, the network’s standard token is the tie that binds the public Blockchain network and facilitates the process of public exchange and transactions.

The smart contract in a public or permission less Blockchain network (or eco-system), today, in its form as a dApp or DAO (Decentralized Autonomous Organization) is primarily being used as a consensus mechanism, where essentially, “anybody is allowed to participate in the network.” [8] “There is no discrimination against robots or humans in the Ethereum ecosystem and contracts can create arbitrary actions like any other account would. Contracts can own tokens, participate in crowd sales, and even be voting members of other contracts”. [9] The contracts can be enhanced with code meant to drive consensus, democratic decision making and delegation of automated tasks, based on the behaviour prescribed by the code within the contract, one of many functions referred to by Ethereum as “Liquid Democracy” [8], all of which is governed by the proof of work mechanism as described in section 2.1.

Blockchain platform service providers such as the Linux Foundation’s Hyperledger Fabric or R3s Corda differentiate themselves from Ethereum with consensus mechanisms primarily geared for and focused on enterprise use. This includes friendliness towards non-fungible (digital) assets and managed access control to private/permissioned networks resulting in increased privacy, security and obviously performance given the controlled size of the private ledger or Blockchain network. [8] [10] Hyperledger Fabric and R3s Corda focus is “tasks and responsibilities in the distributed ledger world by executing code that models or emulates contract logic in the real world.” When compared to Ethereum, currency is not a soft requirement and the focus is on the contract, transaction and non-fungible (digital) assets or utility driven tokens such as data, professional services exchange, and or knowledge-based information as the value driver behind the transaction(s). [8]

3 Possible Applications, Future Vision & Expectations

3.1 Future Vision & Expectations


Current State
Future State

Characteristic	Ethereum	Hyperledger Fabric	R3 Corda
Description of platform	– Generic blockchain platform	– Modular blockchain platform	– Specialized distributed ledger platform for financial industry
Governance	– Ethereum developers	– Linux Foundation	– R3
Mode of operation	– Permissionless, public or private[4]	– Permissioned, private	– Permissioned, private
Consensus	– Mining based on proof-of-work (PoW) – Ledger level	– Broad understanding of consensus that allows multiple approaches – Transaction level	– Specific understanding of consensus (i.e., notary nodes) – Transaction level
Smart contracts	– Smart contract code (e.g., Solidity)	– Smart contract code (e.g., Go, Java)	– Smart contract code (e.g., Kotlin, Java) – Smart legal contract (legal prose)
Currency	– Ether – Tokens via smart contract	– None – Currency and tokens via chaincode	– None

Figure: 3.1-1 [8]

The current state of smart contract technology and key players is well summarized by Figure: 3.1-1. While Ethereum is dominating the market, that fact that it is currency driven may also be a significant constraint in it realizing potential in the enterprise market or non-fungible assets, something that the foundation is looking to address by exploring the development of standards like non-fungible tokens or atomic swaps, for example "ERC721," token standard [11]. Yet with primary focus on currency-based tokens and exchange, and a public mode of operation being at the heart of Ethereum's model, its making it difficult for Ethereum to maneuver into the enterprise and private Blockchain network market.

3.2 Possible Progressive Cybernetics Applications

3.2.1 Breadth

Breadth wise, as enterprises and private organizations begin to adapt to and adopt Blockchain technology as a go to Distributed or Shared Ledger Technology (DLT/SLT) solution, we should expect increased popularity in usage of service platforms such as Hyperledger's Fabric and R3s Corda, whose particular edge, as implied by Figure-1 is looking past the crypto-currency hype and basing fundamental functionality on permissioned based networks and (smart contract) transaction capabilities that are utility, non-fungible and or non-currency/token based. Simply think about the potential for Blockchain application in non-currency-based scenarios or transactions. IBM points out some very interesting opportunities including [11] [12]:

- "Accelerated [Project Management and] Development"
- "Democratic governance"
- "Handling supply chain challenges"
- "Facilitating security-rich patient/provider data exchanges in healthcare"

We're even starting to see the social media giants such as Facebook, Twitter and Google banning advertisements for ICOs. While these actions may be argued as counterproductive or "conspiratorial", the outcome(s) may be a greater push towards "higher quality" and innovative applications of Blockchain technology, effectively breaking the token/currency-based craze that's resulted because of our money/profit/material driven mentality and lifestyle(s). Furthermore, we need to become clearer in describing the fine line between the Blockchain and cryptocurrency paradigms. [13] [14]

3.2.2 Depth

Depth wise, research into possible applications in the context of progressive cybernetics and specifically civic use brought to light a pleasant discovery of a technical report out of Berkeley. "WAVE: A Decentralized Authorization system for IoT via Blockchain Smart Contracts." [15] This report was published on December 29, 2017, only a couple weeks after Democracy of Things (DoT) Vol 1. was published, December 10, 2017, final assignment for COMP 607 Ethics Athabasca University.

The report describes the design of (check this 'DoT' coincidence out), "system which embeds smart contracts cryptographically-enforced delegations, called Delegations of Trust(DoTs). All the DoTs together form a global permission graph which spans different trust domains (such as "work" or "home"). A proof of authorization is a chain of DoTs." [15] The system manages "Decentralized Trust…by using Blockchain as a global ledger for all registered entities, DoTs [and revocations], guaranteeing that all participants know the current state of all permissions (or know that they do not know)." [15]

The demo case in section 4.2 implies that Blockhain can be used for project management in the form of a governance tool. Think of a project management plan and its activities. Each set of activities are representative of smart contract transactions and are bundled into blocks representative of a project phase. The smart contract attributes and functionality define and measure progress against plan. As activities are completed, they are added to blocks, as conditions are met, gating criteria is automatically fulfilled, all parties are notified of progress and project health automatically, in effect decentralizing, automating and optimizing the gating and governance of a project by a Decentralized Autonomous Organization (DAO) that can take on some of the more menial tasks of a Project Manager (PM) or Project Control Officer (PCO). The PM or PCO would essentially orchestrate the definition of the gating conditions within the smart contract(s). In essence, "Governance as a Service." [16]

Even though R3s Corda platform promotes the service as ideal for financial transactions, the functionality, demo and code overview prove that a smart contract transaction produced by Corda does not require a token or currency to be a component of the contract and or transaction attributes (state). The demo is structured to demonstrate very basic examples of project governance opportunities.

4 Functionality

4.1 Functionality (+ Figures & Screenshots)

As discussed so far, the most popular smart contracts platform is Ethereum and the language used to write smart contracts for the Ethereum platform is Solidity, "a high-level language whose syntax is similar to that of JavaScript and it is designed to target the Ethereum Virtual Machine(EVM)." [6] Other languages that commonly used for up and coming platforms including Serpent and Viper (similar to Python), Lisk (uses javascript) and Chain (multi-dimensional SDKs in Ruby, Java and NodeJS). [6] Ethereum does have a Java integration library available by way of the Ethereum(J) library, yet its application and use is still relatively complex and still being ironed out compared to Ethereum's enterprise grade competitors who've focused on building Blockchain platforms that are optimized for Java. Enterprise grade Blockchain platforms such as Chain, Hyperledger Fabric and Corda's smart contract code is much more Java friendly and this positions enterprises whose underlying technology frameworks are heavily based on Java to natively integrate Blockchain solutions with existing systems quickly, efficiently and obviously with less complexity. Nevertheless, the basic process of how smart contracts work on the Blockchain is relatively consistent across platforms and languages. Figure 4.1-1 (below) provides a high-level overview of the functionality process. [17]

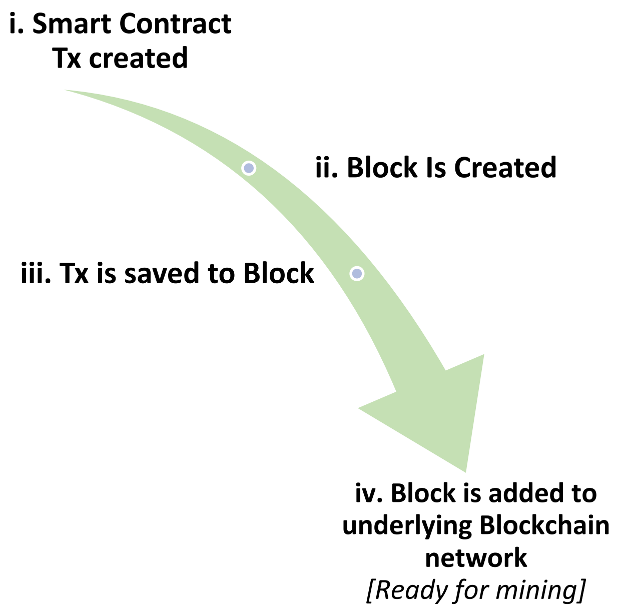

i. Smart Contract Tx created
ii. Block Is Created
iii. Tx is saved to Block
iv. Block is added to underlying Blockchain network
[Ready for mining]

Figure 4.1-1 [17] – Typical Smart Contract Flow

Figure 4.1-2 [18], below provides a visualization of Corda's smart contracts functionality process. Note that "Tx" (or transaction) is analogous to a Block and "Vault" to a private ledger or Blockchain network. Section 4.2 and 4.3 will take a deeper look at the actual structure (and code) of a Corda contract.

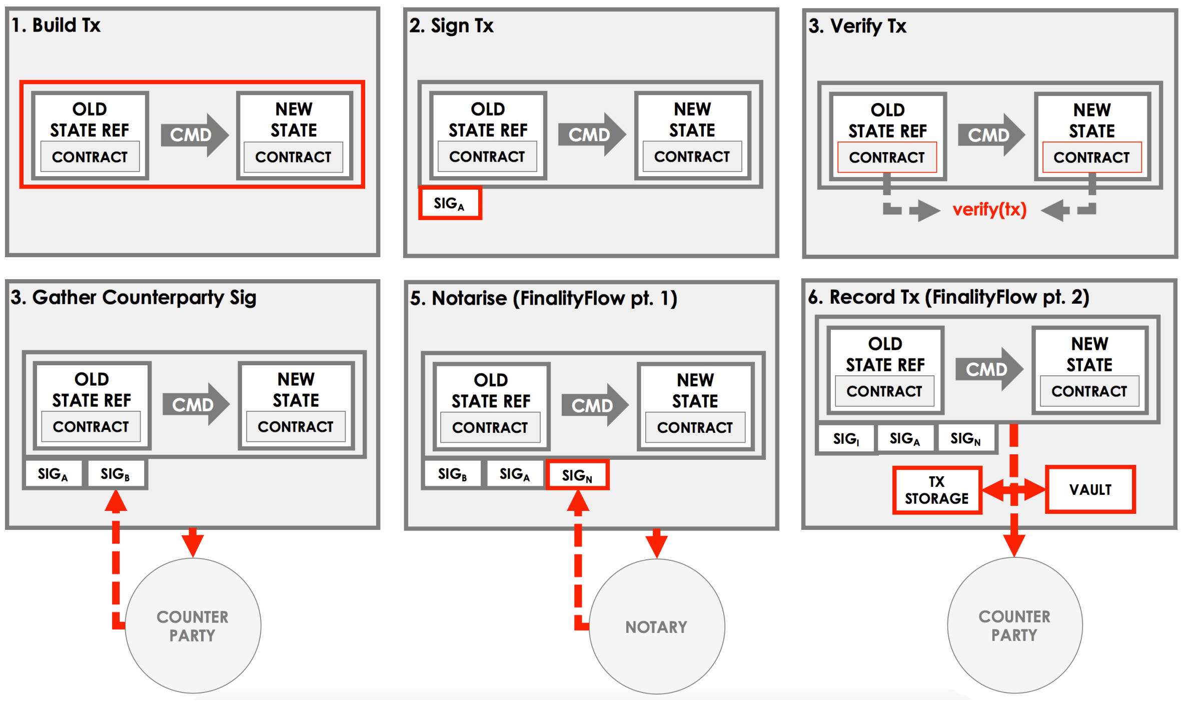


Figure 4.1-2 [18]

Steps 1 through 3 are representative of a transaction's "initiator flow", where [18]:

R3 Corda Contract Flow: Figure 4.1-2 [18]	**Typical Smart Contract Flow:** Figure 4.1-1
The transaction inputs and outputs based on the attributes of a contract are defined and built	[i] Smart Contract Tx created
The contract is signed by the initiator	
The transaction is verified by "running its contracts" [18]	[ii] Block is created

Steps 4 though 6 are representative of a transaction's "responder flow", where [18]:

- The counter party's signature is sought and validated by a notary function
- The transaction is recorded and stored on the vault

R3 Corda Contract Flow	**Typical Smart Contract Flow**
The counter party's signature is sought and validated by a notary function	[iii] Tx is saved to Block
The transaction is recorded and stored on the vault	[iv] Block is added to the underlying Blockchain network

4.2 Demo Java Program: R3 CorDapp example & code fragments

Contract source code serves as a (Java) Class object, and instances of that code (contract) are deployed onto a Blockchain network for validation. Attributes represent the state of the contract (i.e. parties, current state of the contract) and methods (functions) instruct how the contract instance behaves once deployed onto the network. The table below, provides links to R3 Corda's tutorials for building smart contracts from their template(s). Their IOU use case has been customized for demonstration purposes to a use case representative of how smart contract technology can be used for project management and governance. The demo code describes possible interactions between an incubator organization and start-up, specifically, provisioning (or requesting) seed funding and governance of the process. Corda's contract Java template is composed of several Class objects, of which the State, Contract and Flow will be the focus of the code examples and demo [19].

Class Object	**Description**
State [20]	**FR:** `cordapp-contracts-states/src/main/java/com/template/TemplateState.java` **TO:** **`ProjectState`**`.java`
Contract [21]	**FR:** `cordapp-contracts-states/src/main/java/com/template/TemplateContract.java` **TO:** **`ProjectContract`**`.java`
Flow Part One [22] Part Two [23]	**FR:**`cordapp/src/main/java/com/template/TemplateFlow.java` **TO:** **`ProjectFlow`**`.java`

See ***section 4.2.6*** *for more details.*

4.2.1 Contract State (TemplateState.java -> ProjectState.java)

Per Figure 4.2.1-1, the State Class is in essence the contract's core object Class file, which defines and stores the smart contract's state, including attributes, constructors, getters & an @Override call to lock down the participants into an immutable list [20]. This is the block make up. Ultimately, these definitions/attributes are recorded into the vault (or Blockchain network):

Figure 4.2.1-1 [20]

```java
// The contract's (state) definitions / attributes:
public class ProjectState implements ContractState {
    private final int seedCapital;
    private final Party incubator;
    private final Party startup;
    private final int progress;
    private final int approval;
    private final String jurisdiction;
    private final String instructions;
// The contract constructor
    public ProjectState(int seedCapital, Party incubator, Party startup, int progress, int
approval, String jurisdiction, String instructions) {
        this.seedCapital = seedCapital;
        this.incubator = incubator;
        this.startup = startup;
        this.progress = progress;
        this.approval = approval;
        this.jurisdiction = jurisdiction;
        this.instructions = instructions;
    }
// The contract getters
    public int getSeedCapital() { return seedCapital; }
    public Party getIncubator() { return incubator; }
    public Party getStartup() { return startup; }
    public int getProgress() { return progress; }
    public int getApproval() { return approval; }
    public String getJurisdiction() { return jurisdiction; }
    public String getInstructions() { return instructions; }

// A call to lock-down the participants (party's) into an immutable list
    @Override public List<AbstractParty> getParticipants() {
      return ImmutableList.of(incubator, startup);
      }
}
```

4.2.2 Contract (TemplateContract.java -> ProjectContract.java)

Per Figure 4.2.2-1, the Contract Class contains the method(s) that imposes rules on how the state of the contract and its attributes can change (or behave) over time. Note the verify() method and the lambda'esque *requireThat -> { check.using('conditions')}* code that drives the rules or as described by Corda, constraints on "the shape of the transaction, project specific constraints," and definition of the accountable parties as contract signers. [21]

```java
/** Define your contract here. */
// Replaced TemplateContract's definition with => ProjectContract:
public class ProjectContract implements Contract {
    public static final String SEED_FUNDING = "com.template.ProjectContract";

    // The Create (contract) command.
    public static class Create implements CommandData {
    }
    @Override // verification of the transaction-contract via constraints
    public void verify(LedgerTransaction tx) {
        final CommandWithParties<Create> command = requireSingleCommand(tx.getCommands(),
          ProjectContract.Create.class);

        requireThat(check -> {
            // Constraints on the shape of the transaction.
            check.using("No inputs should be consumed when issuing Seed Capital.",
                    tx.getInputs().isEmpty());
            check.using("There should be one output state of type ProjectState.",
                    tx.getOutputs().size() == 1);

            // Project-specific constraints.
            final ProjectState out = tx.outputsOfType(ProjectState.class).get(0);
            final Party incubator = out.getIncubator();
            final Party startup = out.getStartup();
            check.using("The Seed Capital's value must be non-negative.",
                    out.getSeedCapital() > 0);
            check.using("The Incubator and the Startup cannot be the same entity.",
                    incubator != startup);
            check.using("Progress must be 80% or higher for seed funding to be released",
                    out.getProgress() > 79);
            check.using("Approval is required to create a contract.  Approval value is 1.",
                    out.getApproval() == 1);
            check.using("Legal jurisdiction must be included.", out.getJurisdiction() != null);
            check.using("Instructions must be included.", out.getInstructions() != null);

            // Constraints on the signers.
            final List<PublicKey> signers = command.getSigners();
            check.using("There must be two signers.", signers.size() == 2);
            check.using("The Start-Up and Incubator must be Signers.", signers.containsAll(
                    ImmutableList.of(startup.getOwningKey(), incubator.getOwningKey())));
            return null;
        });
    }
}
```

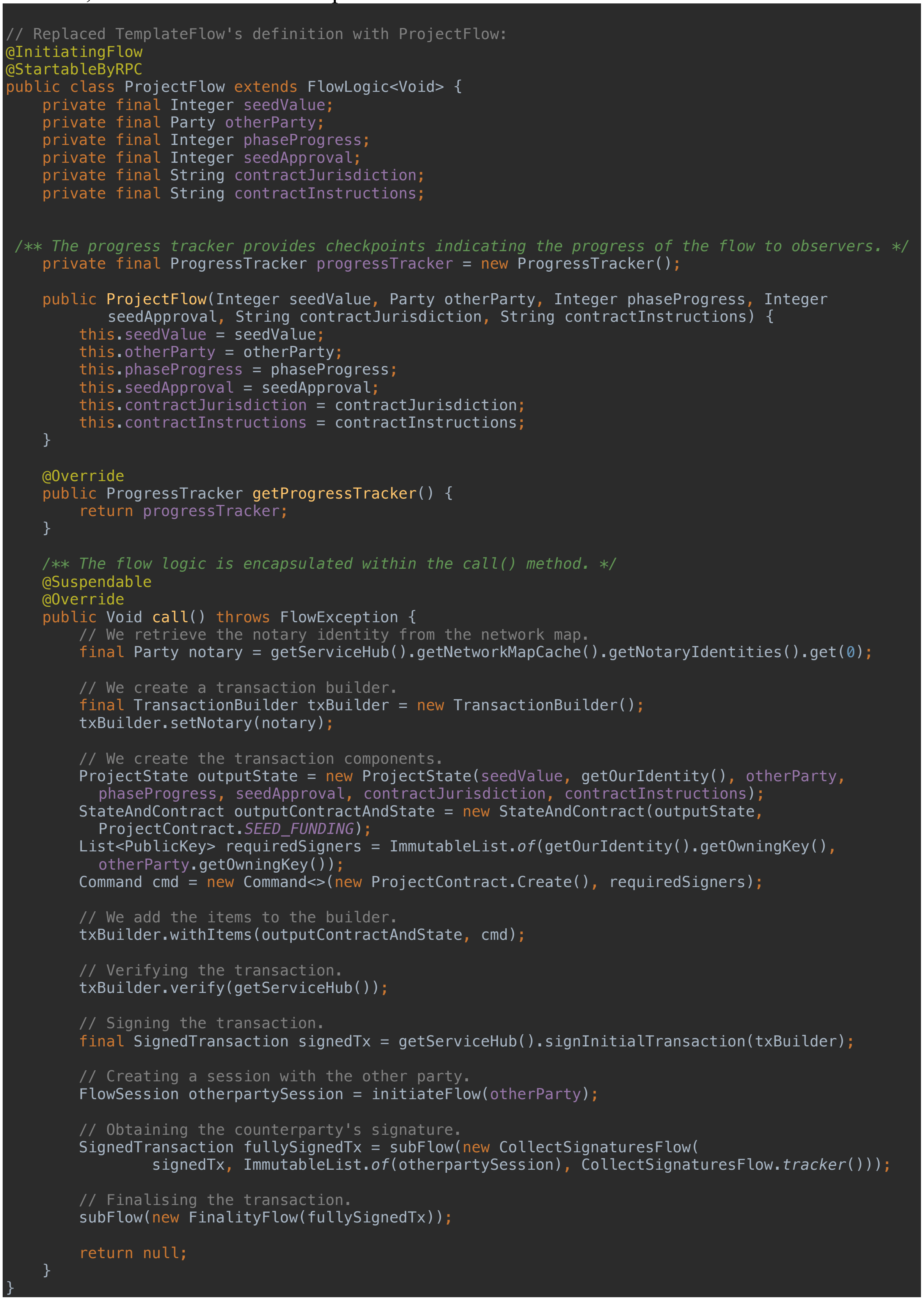
Figure 4.2.2-1

The call on and validation of the public and private keys that play into transactions security and privacy can also be observed, ensuring only those who are identified in the Contract State are privy to the transaction, granted access to the network as an active node and provisioned with the authority to sign a transaction.

4.2.3 Flow (TemplateFlow.java -> ProjectFlow.java)

Per Figure 4.2.3-1, the Flow Class orchestrates the transaction process (or flow). It defines the transaction inputs and proceeds to build the transaction via the call() method which identifies the intermediary notary node and pulls together the transaction components defined within the State and Contract Class, associating the inputs to the transaction details as part of the build process. It proceeds to verify and sign the transaction (notary) before automatically creating a session with the counterparty for a signature and finalizing the transaction before committing it to the vault. [22] [23]

Either party (i.e. incubator or startup) can initiate the transaction, for example, the startup, having met the conditions set out in the Contract, may put in a call for seed funding to be released; the scenario is tested as part of the demo in section 4.2.7.

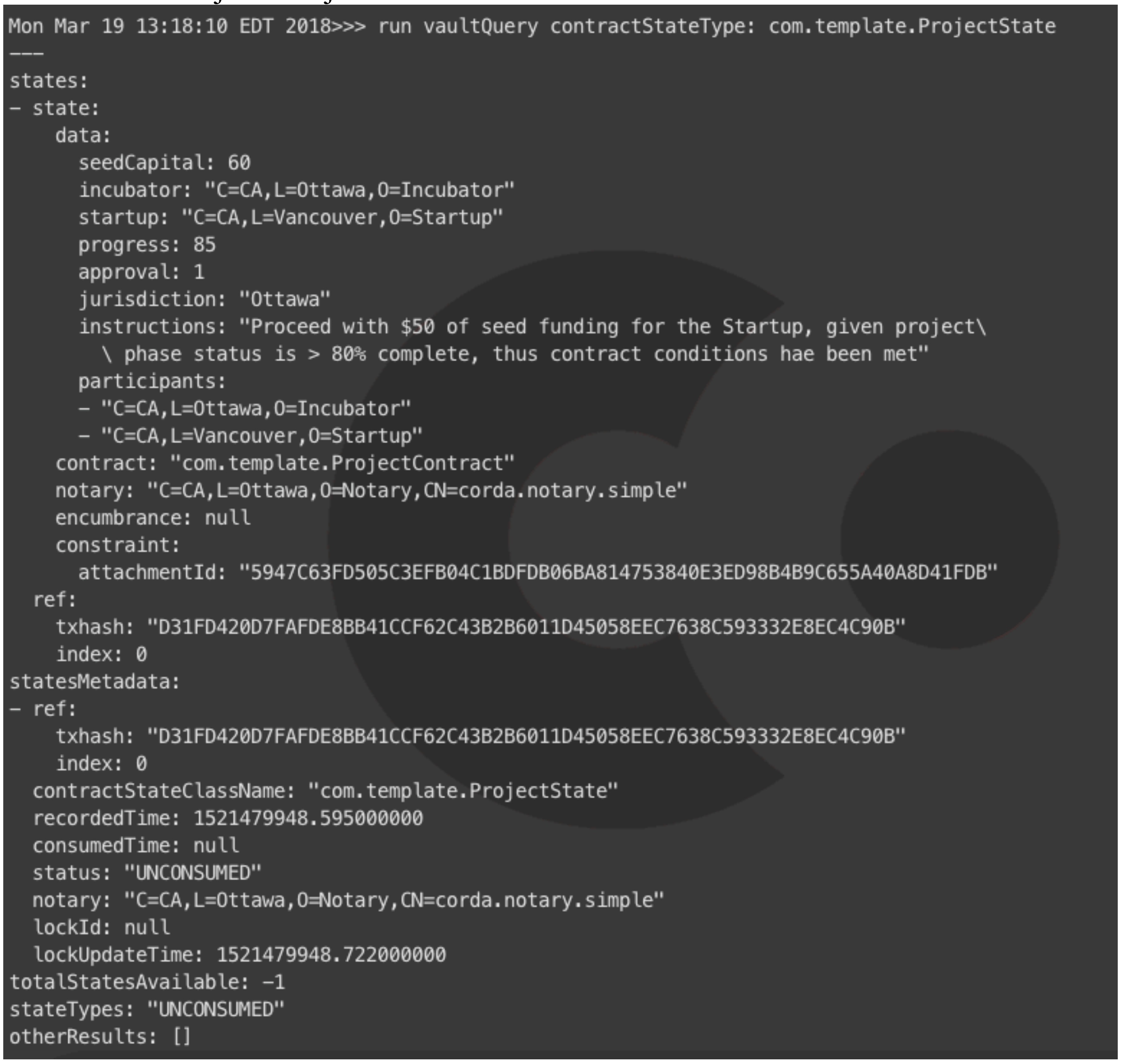


```
// Replaced TemplateFlow's definition with ProjectFlow:
@InitiatingFlow
@StartableByRPC
public class ProjectFlow extends FlowLogic<Void> {
    private final Integer seedValue;
    private final Party otherParty;
    private final Integer phaseProgress;
    private final Integer seedApproval;
    private final String contractJurisdiction;
    private final String contractInstructions;


 /** The progress tracker provides checkpoints indicating the progress of the flow to observers. */
    private final ProgressTracker progressTracker = new ProgressTracker();

    public ProjectFlow(Integer seedValue, Party otherParty, Integer phaseProgress, Integer
           seedApproval, String contractJurisdiction, String contractInstructions) {
        this.seedValue = seedValue;
        this.otherParty = otherParty;
        this.phaseProgress = phaseProgress;
        this.seedApproval = seedApproval;
        this.contractJurisdiction = contractJurisdiction;
        this.contractInstructions = contractInstructions;
    }

    @Override
    public ProgressTracker getProgressTracker() {
        return progressTracker;
    }

    /** The flow logic is encapsulated within the call() method. */
    @Suspendable
    @Override
    public Void call() throws FlowException {
        // We retrieve the notary identity from the network map.
        final Party notary = getServiceHub().getNetworkMapCache().getNotaryIdentities().get(0);

        // We create a transaction builder.
        final TransactionBuilder txBuilder = new TransactionBuilder();
        txBuilder.setNotary(notary);

        // We create the transaction components.
        ProjectState outputState = new ProjectState(seedValue, getOurIdentity(), otherParty,
          phaseProgress, seedApproval, contractJurisdiction, contractInstructions);
        StateAndContract outputContractAndState = new StateAndContract(outputState,
          ProjectContract.SEED_FUNDING);
        List<PublicKey> requiredSigners = ImmutableList.of(getOurIdentity().getOwningKey(),
          otherParty.getOwningKey());
        Command cmd = new Command<>(new ProjectContract.Create(), requiredSigners);

        // We add the items to the builder.
        txBuilder.withItems(outputContractAndState, cmd);

        // Verifying the transaction.
        txBuilder.verify(getServiceHub());

        // Signing the transaction.
        final SignedTransaction signedTx = getServiceHub().signInitialTransaction(txBuilder);

        // Creating a session with the other party.
        FlowSession otherpartySession = initiateFlow(otherParty);

        // Obtaining the counterparty's signature.
        SignedTransaction fullySignedTx = subFlow(new CollectSignaturesFlow(
                signedTx, ImmutableList.of(otherpartySession), CollectSignaturesFlow.tracker()));

        // Finalising the transaction.
        subFlow(new FinalityFlow(fullySignedTx));

        return null;
    }
}
```

Figure 4.2.3-1

4.2.5 Contract / Transaction Record

The following is a screen print of what is recorded in the ‘vault/ledger’ or Blockchain upon finalization of the transaction and is representative of the contract’s variable attributes defined in the ProjectState.java file.



```
Mon Mar 19 13:18:10 EDT 2018>>> run vaultQuery contractStateType: com.template.ProjectState
---
states:
- state:
    data:
      seedCapital: 60
      incubator: "C=CA,L=Ottawa,O=Incubator"
      startup: "C=CA,L=Vancouver,O=Startup"
      progress: 85
      approval: 1
      jurisdiction: "Ottawa"
      instructions: "Proceed with $50 of seed funding for the Startup, given project\
        \ phase status is > 80% complete, thus contract conditions hae been met"
      participants:
      - "C=CA,L=Ottawa,O=Incubator"
      - "C=CA,L=Vancouver,O=Startup"
    contract: "com.template.ProjectContract"
    notary: "C=CA,L=Ottawa,O=Notary,CN=corda.notary.simple"
    encumbrance: null
    constraint:
      attachmentId: "5947C63FD505C3EFB04C1BDFDB06BA814753840E3ED98B4B9C655A40A8D41FDB"
  ref:
    txhash: "D31FD420D7FAFDE8BB41CCF62C43B2B6011D45058EEC7638C593332E8EC4C90B"
    index: 0
statesMetadata:
- ref:
    txhash: "D31FD420D7FAFDE8BB41CCF62C43B2B6011D45058EEC7638C593332E8EC4C90B"
    index: 0
  contractStateClassName: "com.template.ProjectState"
  recordedTime: 1521479948.595000000
  consumedTime: null
  status: "UNCONSUMED"
  notary: "C=CA,L=Ottawa,O=Notary,CN=corda.notary.simple"
  lockId: null
  lockUpdateTime: 1521479948.722000000
totalStatesAvailable: -1
stateTypes: "UNCONSUMED"
otherResults: []
```

4.2.6 Demo Program Overview

Content in sections 4.2.6.1 are adapted from the content and context provided by R3 Corda tutorials. [19] [24] [25]

R3 Corda offers a demo app called DemoBench for Corda 3.0, downloadable from here. There is an installer for both Windows and Mac OS X.

For the purposes of this essay, the described use case in sections 4.2.1 through 4.2.3 is an adaptation of the IOUState smart contract java template tutorials offered by R3 Corda into a functioning demo that can be launched by DemoBench.

- The demo builds on parts 1 and 2 of the Hello, World! Corda tutorial and transforms the tutorial template files that can be found on github here.
- The tutorial(s) can be found here ➔ Part 1 [24] and Part 2. [25]

The assignment zip file bundles all of the built and compiled code as requirements to study the demo case code and execute the demo using a provided DemoBench program by Corda. The assets are found in the 'CordaAppDemo' folder, breakdown as follows:

Folder	Description
/COMP501A3P1Bundle/CordAppDemo/cordapp-bc-startup/	Complete and compiled build baseline for the demo. The build is Gradle based and can be rebuilt following the instructions provided by Corda here; as well as in **section 4.2.6.1**
• /CordAppDemo/DemoJars/MacBuildJARs o /cordapp-0.1.jar o /cordapp-contracts-states-0.1.jar o /cordapp-template-java-0.1.jar • /CordAppDemo/DemoJars/WinBuildJARs o /cordapp-0.1.jar o /cordapp-contracts-states-0.1.jar o /cordapp-template-java-0.1.jar	An extract of the built JAR files required to run the demo by the Corda DemoBench program. Whether running the demo on Mac or Windows, it is recommended to use the JAR files built within Windows. See section 4.2.7 for details and step by step instructions to run the demo.
/COMP501A3P1Bundle/CordAppDemo/DemoJavaClassFiles/	An extract of the built .java files modified for the demo, including, documentation and test plans.

The demo is based on adjustments to the following four template .java files:

From	To
/cordapp-template-java/cordapp-contracts-states/src/main/java/com/template/IOUState.java	/cordapp-bc-startup/cordapp-contracts-states/src/main/java/com/template/ProjectState.java
/cordapp-template-java/cordapp-contracts-states/src/main/java/com/template/IOUContract.java	/cordapp-bc-startup/cordapp-contracts-states/src/main/java/com/template/ProjectContract.java
/cordapp-template-java/cordapp/src/main/java/com/template/IOUFlow.java	/cordapp-bc-startup/cordapp/src/main/java/com/template/ProjectFlow.java
/cordapp-template-java/cordapp/src/main/java/com/template/IOUFlowResponder.java	/cordapp-bc-startup/cordapp/src/main/java/com/template/ProjectFlowResponder.java

4.2.6.1 Instructions to build and compile the demo program on your own

Please note: Everything required to run and test the demo are already provided in the assignment zip file (package) and with instructions in **section 4.2.7**. That being said, the requirements and process to build and compile your own version are as follows:

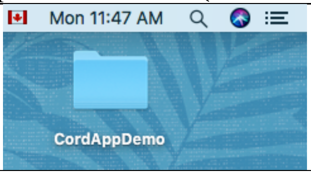
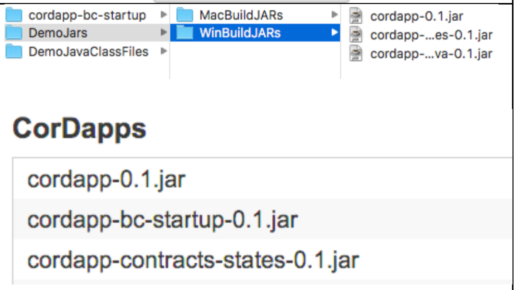
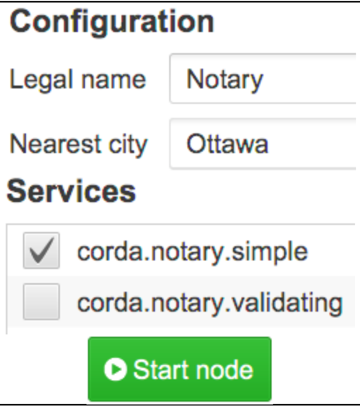
- Download the Corda Java template via terminal as follows:
  - `git clone https://github.com/corda/cordapp-template-java.git`
- The root folder **/cordapp-template-java/** will be created
- Open the template from the root folder using IntelliJ IDEA with Java JDK 1.8.x installed. Note(s):
  - Do not import the project.
  - Ensure to open the project and initiate the event notification that results to perform the Gradle import.
  - During the Gradle import, when asked to associate a JDK, ensure to select the JDK (1.8.x) installed on your system.
  - The Gradle import into IntelliJ IDEA and build will not work properly if JDK 9 is the main SDK for the operating system.
  - When the Gradle import is complete, proceed to the following step.
- Delete the following file:
  - `/cordapp/src/main/java/com/template/TemplateClient.java`
- Replace the following files with the provided .java files in the assignment zip file (package), folder:
  - `/COMP501A3P1Bundle/CordAppDemo/DemoJavaClassFiles/`

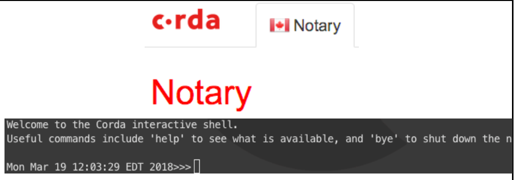
From (root directory ➔ /cordapp-template/java)	**Replace with (.java files found in /DemoJavaClassFiles)**
/cordapp-contracts-states/src/main/java/com/template/**TemplateState.java**	/cordapp-contracts-states/src/main/java/com/template/**ProjectState.java**
/cordapp-contracts-states/src/main/java/com/template/**TemplateContract.java**	/cordapp-contracts-states/src/main/java/com/template/**ProjectContract.java**
/cordapp/src/main/java/com/template/**TemplateFlow.java**	/cordapp/src/main/java/com/template/**ProjectFlow.java**
Add the [Project]FlowResponder.java file to the following directory (this .java file is created during the tutorial)	/cordapp/src/main/java/com/template/**ProjectFlowResponder.java**

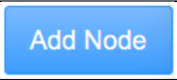
- Via terminal from the root directory ➔ **/cordapp-template-java/**, run a build with the following command:
  - For Windows ➔ gradlew clean deployNodes
  - For Mac ➔ ./gradlew clean deployNodes
- Find and copy the following built JAR files into a separate and accessible directory on your computer:
  - /cordapp-template-java/build/libs/cordapp-template-java-0.1.jar
  - /cordapp-template-java/cordapp/build/libs/cordapp-0.1.jar
  - /cordapp-template-java/cordapp-contract-states/build/libs/cordapp-contract-states-0.1.jar
- Proceed to **section 4.2.7** for instructions and a test plan on how to run the demo by loading these JAR files into the Corda DemoBench

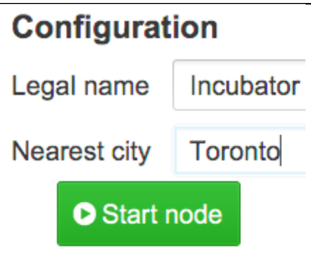
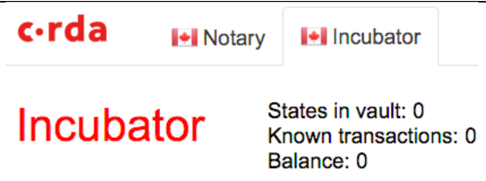
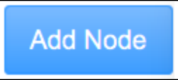
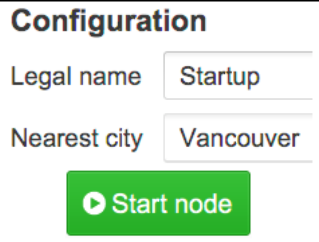
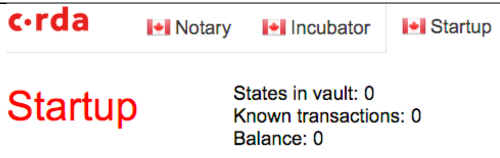
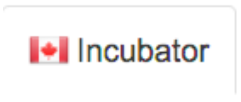
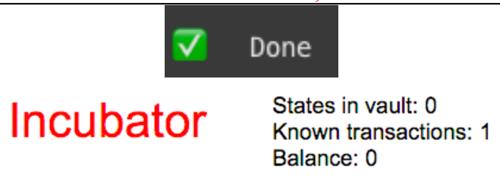
4.2.7 Demo Program Instructions & Test Plan

The following is a step by step guide for running the demo program via Corda's DemoBench to recreate the contract / transaction in section 4.2.5:

Instructions & Input Steps	Expected Results (& Visuals)
Locate the CordAppDemo file directory which is part of the assignment zip file. Place the directory content in an accessible location on your computer, i.e. Desktop.	
Either install the DemoBench program from the CordAppDemo package or download and install it from R3 Corda's DemoBench site here.	• Windows Installer (exe) • Mac OS X Installer (dmg)
Launch the (installed) DemoBench program.	
Click on the '+Add CorDapp' button.	
Locate the essay's demo package and load each .jar file into the DemoBench individually. **Important note:** • Within the DemoJarFiles directory, there are two sub-directories => WIN & MAC; each directory is a ***Gradle*** build of the JAR files conducted in the respective OS. • Command line behavior within the interactive shells is different depending on the collection of JAR files chosen to be loaded; this is due to how the **Gradle** build process behaves depending on the OS the package is built on • Commands are provided for both WIN & MAC • If WIN BUILD JARs are loaded onto a MAC OS version of Corda DemoBench, the WIN commands **will** work, and **they are friendlier**	 cordapp-bc-startup DemoJars DemoJavaClassFiles MacBuildJARs WinBuildJARs cordapp-0.1.jar cordapp-...es-0.1.jar cordapp-...va-0.1.jar CorDapps cordapp-0.1.jar cordapp-bc-startup-0.1.jar cordapp-contracts-states-0.1.jar
In the 'Configuration' section, leave the Legal name as 'Notary'. In the 'Nearest city' field type 'Ottawa' (or scroll and find Ottawa in the drop-down list). In the 'Services' section, 'corda.notary.simple' should be checked. Click on the 'Start node' button.	 Configuration Legal name: Notary Nearest city: Ottawa Services ✓ corda.notary.simple ☐ corda.notary.validating

An interactive shell will load for the Notary. Successful loading of the node is represented by the following console content:
 	
Welcome to the Corda interactive shell.
Useful commands include 'help' to see what is available, and 'bye' to shut down the node.
 	
Mon Mar 19 12:03:29 EDT 2018>>>
	
	`Welcome to the Corda interactive shell.`
	`Useful commands include 'help' to see what is available, and 'bye' to shut down the n`
	
	`Mon Mar 19 12:03:29 EDT 2018>>>`

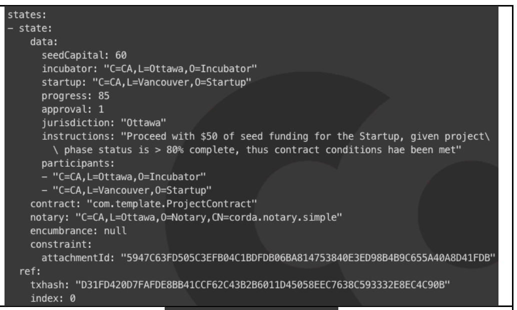
Continue by clicking on the 'Add Node" button.	

In the 'Configuration' section, replace the Legal name with 'Incubator'. In the 'Nearest city' field type 'Toronto' (or scroll and find Toronto in the drop-down list). Leave the 'Services' section blank. Do not check any of the 'Services' options. Click on the 'Start node' button.	 **Configuration** Legal name: Incubator Nearest city: Toronto
An interactive shell will load for the Incubator.	 Incubator States in vault: 0 Known transactions: 0 Balance: 0
Continue by clicking on the 'Add Node" button.	
In the 'Configuration' section, replace the Legal name with 'Startup'. In the 'Nearest city' field type 'Vancouver' (or scroll and find Vancouver in the drop-down list). Leave the 'Services' section blank. Do not check any of the 'Services' options. Click on the 'Start node' button.	 **Configuration** Legal name: Startup Nearest city: Vancouver
An interactive shell will load for the Startup. All three nodes should now be active and ready to perform a transaction (testing).	 Startup States in vault: 0 Known transactions: 0 Balance: 0
Select the 'Incubator' tab to activate the 'Incubator' interactive shell.	
Note: All entries into the interactive shell for testing must be entered **EXACTLY** as shown, **no line breaks**.	
Test 1➔ Incubator to Startup seed funding approval Select the 'Incubator' tab. Type in the following commands in the 'Incubator' interactive shell: ***Mac Commands:*** **start ProjectFlow arg0: 60, arg1: "Startup", arg2: 85, arg3: 1, arg4: "Ottawa", arg5: "Proceed with seed funding. Project status is > 80%. Contract conditions met."** ***Win Commands:*** **start ProjectFlow seedValue: 60, otherParty: Startup, phaseProgress: 85, seedApproval: 1, contractJurisdiction: Ottawa, contractInstructions: Proceed with seed funding. Project status is > 80%. Contract conditions met.**	 Incubator States in vault: 0 Known transactions: 1 Balance: 0
Test 2➔ Startup views the transaction content of each node's vault (there should only be 1 transaction)	**Zoom in to view or see section 4.2.5**

Select the 'Startup' tab. Type in the following commands in the 'Startup' interactive shell:

Mac and Win Commands:

run vaultQuery contractStateType: com.template.ProjectState



```
states:
- state:
    data:
      seedCapital: 60
      incubator: "C=CA,L=Ottawa,O=Incubator"
      startup: "C=CA,L=Vancouver,O=Startup"
      progress: 85
      approval: 1
      jurisdiction: "Ottawa"
      instructions: "Proceed with $50 of seed funding for the Startup, given project\
        \ phase status is > 80% complete, thus contract conditions hae been met"
      participants:
      - "C=CA,L=Ottawa,O=Incubator"
      - "C=CA,L=Vancouver,O=Startup"
    contract: "com.template.ProjectContract"
    notary: "C=CA,L=Ottawa,O=Notary,CN=corda.notary.simple"
    encumbrance: null
    constraint:
      attachmentId: "5947C63FD505C3EFB04C1BDFDB06BA814753840E3ED98B4B9C655A40A8D41FDB"
  ref:
    txhash: "D31FD420D7FAFDE8BB41CCF62C43B2B6011D45058EEC7638C593332E8EC4C90B"
    index: 0
```

Test 3➔ Startup to Incubator seed funding request

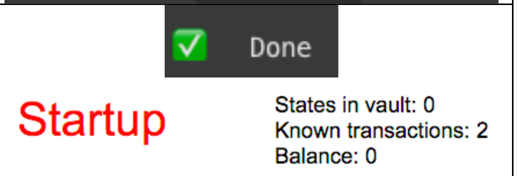
Select the 'Startup' tab. Type in the following commands in the 'Startup' interactive shell:

Mac Commands:

start ProjectFlow arg0: 40, arg1: "Incubator", arg2: 95, arg3: 1, arg4: "Ottawa", arg5: "Requesting seed funding balance from Incubator. Project phase status is > 90% complete. Balance can be released."

Win Commands:

start ProjectFlow seedValue: 40, otherParty: Incubator, phaseProgress: 95, seedApproval: 1, contractJurisdiction: Ottawa, contractInstructions: Requesting seed funding balance from Incubator. Project phase status is > 90% complete. Balance can be released.



States in vault: 0
Known transactions: 2
Balance: 0

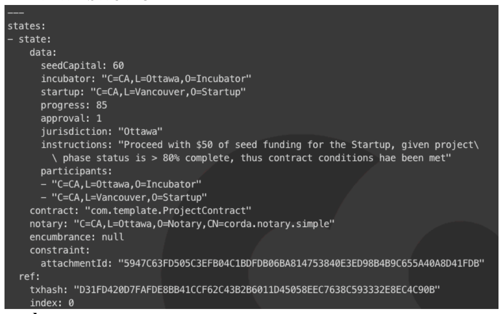
Test 4➔ Incubator views the transaction content of each node's vault (there should be 2 transaction)

Select the 'Incubator' tab. Type in the following commands in the 'Incubator' interactive shell:

Mac and Win Commands:

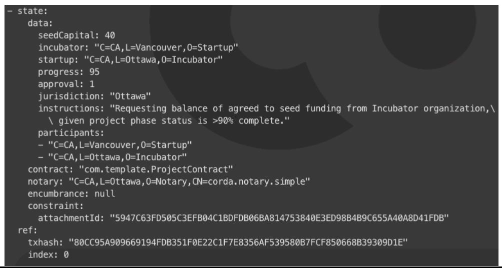
run vaultQuery contractStateType: com.template.ProjectState

1st transaction



```
---
states:
- state:
    data:
      seedCapital: 60
      incubator: "C=CA,L=Ottawa,O=Incubator"
      startup: "C=CA,L=Vancouver,O=Startup"
      progress: 85
      approval: 1
      jurisdiction: "Ottawa"
      instructions: "Proceed with $50 of seed funding for the Startup, given project\
        \ phase status is > 80% complete, thus contract conditions hae been met"
      participants:
      - "C=CA,L=Ottawa,O=Incubator"
      - "C=CA,L=Vancouver,O=Startup"
    contract: "com.template.ProjectContract"
    notary: "C=CA,L=Ottawa,O=Notary,CN=corda.notary.simple"
    encumbrance: null
    constraint:
      attachmentId: "5947C63FD505C3EFB04C1BDFDB06BA814753840E3ED98B4B9C655A40A8D41FDB"
  ref:
    txhash: "D31FD420D7FAFDE8BB41CCF62C43B2B6011D45058EEC7638C593332E8EC4C90B"
    index: 0
```

2nd transaction



```
- state:
    data:
      seedCapital: 40
      incubator: "C=CA,L=Vancouver,O=Startup"
      startup: "C=CA,L=Ottawa,O=Incubator"
      progress: 95
      approval: 1
      jurisdiction: "Ottawa"
      instructions: "Requesting balance of agreed to seed funding from Incubator organization,\
        \ given project phase status is >90% complete."
      participants:
      - "C=CA,L=Vancouver,O=Startup"
      - "C=CA,L=Ottawa,O=Incubator"
    contract: "com.template.ProjectContract"
    notary: "C=CA,L=Ottawa,O=Notary,CN=corda.notary.simple"
    encumbrance: null
    constraint:
      attachmentId: "5947C63FD505C3EFB04C1BDFDB06BA814753840E3ED98B4B9C655A40A8D41FDB"
  ref:
    txhash: "80CC95A909669194FDB351F0E22C1F7E8356AF539580B7FCF850668B39309D1E"
    index: 0
```

Test 5➔ Condition check: Same entity transaction

Select the 'Startup' tab. Type in the following commands in the 'Startup' interactive shell:

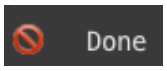
Mac Commands:

start ProjectFlow arg0: 10, arg1: "Startup", arg2: 99, arg3: 1, arg4: "Ottawa", arg5: "Testing same entity transaction."

Win Commands:

start ProjectFlow seedValue: 10, otherParty: Startup, phaseProgress: 99, seedApproval: 1, contractJurisdiction: Ottawa, contractInstructions: Testing same entity transaction.

Console output:



☠Contract verification failed: Failed requirement: The Incubator and the Startup cannot be the same entity., contract: com.template.ProjectContract@18951083, transaction: 256B0EE20CD081A3EE15990BDBD3BCAE8F067BF6DA91B10FB9099299C378FEDD

Test 6➔ Condition Check: Not approved: Value of 0 Select the ‘Incubator’ tab. Type in the commands in the ‘Incubator’ interactive shell: ***Mac Commands:*** **start ProjectFlow arg0: 20, arg1: "Startup", arg2: 95, arg3: 0, arg4: "Ottawa", arg5: "Testing not approved. arg3 or seedApproval value of 0 condition.”** ***Win Commands:*** **start ProjectFlow seedValue: 20, otherParty: Startup, phaseProgress: 95, seedApproval: 0, contractJurisdiction: Ottawa, contractInstructions: Testing not approved. arg3 or seedApproval value of 0 condition.**	**Console output:** 🚫 Done ☠Contract verification failed: Failed requirement: Approval is required to create a contract. Approval value is 1, contract: com.template.ProjectContract@3e8aad54, transaction: 6B2D0FCB21E6E18733FD6636BF5D55F3 DFCAF8362A8E9BF19F40BB18538C428A
Test 7➔ Condition check: Phase progress less than 80% Select the ‘Startup’ tab. Type in the following commands in the ‘Startup’ interactive shell: ***Mac Commands:*** **start ProjectFlow arg0: 30, arg1: "Incubator”, arg2: 75, arg3: 1, arg4: "Ottawa", arg5: "Testing project phase progress less than 80% @ 75%"** ***Win Commands:*** **start ProjectFlow seedValue: 30, otherParty: Incubator, phaseProgress: 75, seedApproval: 1, contractJurisdiction: Ottawa, contractInstructions: Testing project phase progress less than 80% @ 75%**	**Console output:** 🚫 Done ☠Contract verification failed: Failed requirement: Progress must be 80% or higher for seed funding to be released, contract: com.template.ProjectContract@42de1148, transaction: C05EC5A366E4990A714B568BDB7052F6B 98940811CD6C1CE16406EECE812D7AC
Test 8➔ Condition check: Null value for jurisdiction Select the ‘Startup’ tab. Type in the following commands in the ‘Startup’ interactive shell: ***Mac Commands:*** **start ProjectFlow arg0: 40, arg1: "Incubator”, arg2: 95, arg3: 1, arg4: , arg5: "Testing null value for jurisdiction."** ***Win Commands:*** **start ProjectFlow seedValue: 40, otherParty: Incubator, phaseProgress: 95, seedApproval: 1, contractJurisdiction: , contractInstructions: Testing null value for jurisdiction.**	**Console output:** 🚫 Done ☠ Contract verification failed: Failed requirement: Legal jurisdiction must be included., contract: com.template.PrjectContract@7bdf7b45, transaction: D2EA513D91C8E4BD9AC93980C62B420D FFF7BCBED71E4ED409BEFD204B0E0B6 B
Test 9➔ Condition check: Null value for instructions Select the ‘Startup’ tab. Type in the following commands in the ‘Startup’ interactive shell: ***Mac Commands:*** **start ProjectFlow arg0: 50, arg1: "Incubator”, arg2: 95, arg3: 1, arg4: “Ottawa”, arg5:** ***Win Commands:*** **start ProjectFlow seedValue: 50, otherParty: Incubator, phaseProgress: 95, seedApproval: 1, contractJurisdiction: Ottawa, contractInstructions:**	**Console output:** 🚫 Done Contract verification failed: Failed requirement: Instructions must be included., contract: com.template.ProjectCntract@70e484b2, transaction: 58F99A7871335B247F310FC7E872315D70 79B2AC527CC4CC3FA7B230ED015B23

4 Conclusion

Current popular applications and investment is being made in public Blockchain network's with ICOs being managed by smart contract driven DAOs, this is resulting in an isolative nature of being constrained to currency as tokens.

Currency, in effect may be constraining the realization of great potential for smart contract and dApp technology. As the brief description of non-currency based applications imply, by simply taking BitCoin (token) out of the Blockchain picture, and putting emphasis on the Blockchain mechanism managing smart contract code, more innovative and promising applications are slowly and surely being thought about and realized.

5 Works Cited

[1] S. Nakamoto, ""Bitcoin: A Peer-to-Peer Electronic Cash System," www.bitcoin.org, 2008.

[2] M. Gupta, "Blockchain For Dummies, IBM Limited Edition," John Wiley & Sons, Inc., 2017.

[3] V. Buterin, "Ethereum: The Ultimate Smart Contract and Decentralized Application Platform," December 2013. [Online]. Available: http://web.archive.org/web/20131228111141/http://vbuterin.com/ethereum.html. [Accessed 28 February 2018].

[4] World Bank Group, "Distributed Ledger Technology (DLT) and Blockchain," International Bank for Reconstruction and Development / the World Bank, Washington, DC, 2017.

[5] Wikipedia, "Wikipedia: Blockchain," Wikipedia, 03 November 2017. [Online]. Available: https://en.wikipedia.org/wiki/Blockchain. [Accessed 28 February 2018].

[6] HowToToken Team, "How To Write A Smart-Contract For Your ICO? An Ultimate guide," Howtotoken, 2018. [Online]. Available: https://howtotoken.com/ico/how-to-write-a-smart-contract-for-your-ico-an-ultimate-guide/.

[7] Investopedia, "Investopedia: Fungibility," [Online]. Available: https://www.investopedia.com/terms/f/fungibility.asp. [Accessed 14 March 2018].

[8] P. Sandner, "Medium: Comparison of Ethereum, Hyperledger Faric and Corda," Frankfurt School Blockchain Center, 25 June 2017. [Online]. Available: https://medium.com/@philippsandner/comparison-of-ethereum-hyperledger-fabric-and-corda-21c1bb9442f6. [Accessed 2 March 2018].

[9] Ethereum Foundation, "Ethereum: Start a democratic organization," Ethereum Foundation, 2018. [Online]. Available: https://www.ethereum.org/dao.

[10] P. Jayachandran, "The difference between public and private blockchain," IBM, 31 May 2017. [Online]. Available: https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/.

[11] T. V. BJORØY, "VB: CryptoKitties shows everything can — and will — be tokenized," 4 December 2017. [Online]. Available: https://venturebeat.com/2017/12/04/cryptokitties-shows-everything-can-and-will-be-tokenized/.

[12] IBM, "IBM Blockchain Platform," IBM, [Online]. Available: https://www.ibm.com/blockchain/platform/?S_PKG=AW&cm_mmc=Search_Google-_-Blockchain_Blockchain-_-WW_NA-_-+Smart++Contracts_Broad_AW&cm_mmca1=000026VG&cm_mmca2=10007330&cm_mmca7=1002334&cm_mmca8=kwd-389998830422&cm_mmca9=4358b4e5-bae1-4f95-a122-b925ab32689.

[13] D. Hollerith, "What Big Tech's Ban Might Mean for Cryptocurrency Advertising," Bitcoin Magazine, 20 March 2018. [Online]. Available: https://bitcoinmagazine.com/articles/what-big-techs-ban-might-mean-cryptocurrency-advertising/.

[14] R. Leathern, "New Ads Policy: Improving Integrity and Security of Financial Product and Services Ads," Facebook (Business), 30 January 2018. [Online]. Available: https://www.facebook.com/business/news/new-ads-policy-improving-integrity-and-security-of-financial-product-and-services-ads.

[15] M. P. Anderson, J. Kolb, K. Chen, G. Fierro, D. E. Culler and R. Ada Popa, "WAVE: A Decentralized Authorization System for IoT via Blockchain Smart Contracts," EECS Department: University of California, Technical Report No. UCB/EECS-2017-234, December 29, 2017, Berkeley, 2017.

[16] A. Norta, A. B. Othman and K. Taveter, "Conflict-Resolution Lifecycles for Governed Decentralized Autonomous Organization Collaboration.," in *ACM: In Proceedings of the 2015 2nd International Conference on Electronic Governance and Open Society: Challenges in Eurasia (EGOSE '15)*, New York, 2015.

[17] A. Khalid, "Understanding Blockchain with EthereumJ Open Source Development: Release 0.3," Medium, 21 April 2017. [Online]. Available: https://medium.com/@arsalanskhalid/understanding-blockchain-with-ethereumj-open-source-development-release-0-3-2f94bddbd28c.

[18] R3 Limited, "Corda: API Flows: An example flow," R3 Limited, 2018. [Online]. Available: https://docs.corda.net/releases/release-M13.0/api-flows.html.

[19] R3 Limited, "Corda: The CorDapp Template," R3 Limited, 2018. [Online]. Available: https://docs.corda.net/hello-world-template.html.

[20] R3 Limited, "Corda: Writing the state," R3 Limited, 2018. [Online]. Available: https://docs.corda.net/hello-world-state.html.

[21] R3 Limited, "Corda: Writing the contract," R3 Limited, 2018. [Online]. Available: https://docs.corda.net/tut-two-party-contract.html.

[22] R3 Limited, "Corda: Writing the flow," R3 Limited, 2018. [Online]. Available: https://docs.corda.net/hello-world-flow.html.

[23] R3 Limited, "Corda: Updating the flow," R3 Limited, 2018. [Online]. Available: https://docs.corda.net/tut-two-party-flow.html.

[24] R3 Limited, "Corda (Tutorial): Hello, World!," 2018. [Online]. Available: https://docs.corda.net/hello-world-introduction.html.

[25] R3 Limited, "Hello, World! Pt. 2 - Contract constraints," R3 Limited, 2018. [Online]. Available: https://docs.corda.net/tut-two-party-introduction.html.