Supplementary Material: Computational Governance and Violable Contracts for Blockchain Applications

Munindar P. Singh and Amit K. Chopra

1 Introduction

We document here the details of a proof of concept implementation of compacts over R3 Corda. The setting of the implementation is a buyer-seller interaction. We give (1) a norm (commitment) specification over business events, (2) buyer and seller agents in Corda that insert events into the relevant nodes in the blockchain, (3) a node database containing event instances, (4) lifecycle queries compiled into SQL from the norm, and (5) the results of computing these queries on the database. We executed the queries on the seller's database; the results would be identical if run on the buyer's database since they observe the same events.

2 Compact Specification

Listing 2 gives the schema of the relevant business events and then gives a simple commitment specification over those events: Seller commits to Buyer that if *Payment* is made within two days of *Offer* and the payment is for an amount that is equal to the price specified in the offer, then *Delivery* will be done within two days of *Payment*.

```
schema
 Offer(buyer, item, oID, price, seller)
  key oID time timestamp
 Payment(amount, buyer, oID, seller)
  key oID time timestamp
 Delivery(buyer, oID,seller)
  key oID time timestamp

 commitment OfferCom from seller to buyer
  create Offer
  detach Payment deadline Offer + 2d
          where price=amount
  discharge Delivery deadline Payment + 2d
```

3 Agents

The seller agent inserts *Offer* and *Delivery* event instances by invoking the respective flows. A realistic seller agent would encode the policies by which it takes decisions to insert those events, but we omit the policies since they are not relevant for our purposes. Notice the dates of the events.

The buyer agent inserts *Payment* event instances.

Below, we gives listings of both agents, implemented as Corda RPCclients.

```java
package com.example.flow;

import net.corda.client.rpc.CordaRPCClient;
import net.corda.client.rpc.CordaRPCClientConfiguration;
import net.corda.core.identity.Party;
import net.corda.core.messaging.CordaRPCOps;
import net.corda.core.utilities.NetworkHostAndPort;
import org.apache.activemq.artemis.api.core.ActiveMQException;

import java.util.Calendar;
import java.util.Date;
import java.util.concurrent.ExecutionException;

public class SellerRPCClient {

    public static void main(String[] args) throws ActiveMQException, InterruptedException
, ExecutionException {

        final NetworkHostAndPort nodeAddress = NetworkHostAndPort.parse(args[0]);
        final CordaRPCClient client = new CordaRPCClient(nodeAddress,
CordaRPCClientConfiguration.DEFAULT);

        final CordaRPCOps proxy = client.start("user1", "test").getProxy();

        Party sender = (Party)proxy.partiesFromName("PartyA", false).iterator().next();
        Party receiver = (Party) proxy.partiesFromName("PartyB", false).iterator().next()
;

        //Created
        Calendar cal = Calendar.getInstance();
        cal.set(Calendar.YEAR, 2019);
        cal.set(Calendar.MONTH, Calendar.MARCH);
        cal.set(Calendar.DAY_OF_MONTH,10);
        Date d = cal.getTime();
        proxy.startFlowDynamic(OfferFlow.Initiator.class, receiver, "art", "30", "1", d);

        //Created another
        cal = Calendar.getInstance();
        cal.set(Calendar.YEAR, 2019);
        cal.set(Calendar.MONTH, Calendar.MARCH);
        cal.set(Calendar.DAY_OF_MONTH, 10);
        d = cal.getTime();
        proxy.startFlowDynamic(OfferFlow.Initiator.class, receiver, "scooter", "1000", "2
", d);

        //Discharged
        cal = Calendar.getInstance();
        cal.set(Calendar.YEAR, 2019);
        cal.set(Calendar.MONTH, Calendar.MARCH);
        cal.set(Calendar.DAY_OF_MONTH, 12);
        d = cal.getTime();
        proxy.startFlowDynamic(DeliveryFlow.Initiator.class, receiver, "1", d);

        //Violated
        cal = Calendar.getInstance();
        cal.set(Calendar.YEAR, 2019);
        cal.set(Calendar.MONTH, Calendar.MARCH);
        cal.set(Calendar.DAY_OF_MONTH, 18);
        d = cal.getTime();
        proxy.startFlowDynamic(DeliveryFlow.Initiator.class, receiver, "2", d);

    }
}
```

```java
package com.example.flow;

import net.corda.client.rpc.CordaRPCClient;
import net.corda.client.rpc.CordaRPCClientConfiguration;
import net.corda.core.contracts.StateAndRef;
import net.corda.core.identity.Party;
import net.corda.core.messaging.CordaRPCOps;
import net.corda.core.utilities.NetworkHostAndPort;
import org.apache.activemq.artemis.api.core.ActiveMQException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Calendar;
import java.util.Date;
import java.util.concurrent.ExecutionException;

/**
 * Demonstration of how to use the CordaRPCClient to connect to a Corda Node and
 * stream the contents of the node's vault.
 */
public class BuyerRPCClient {

    public static void main(String[] args) throws ActiveMQException, InterruptedException
, ExecutionException {

        final NetworkHostAndPort nodeAddress = NetworkHostAndPort.parse(args[0]);
        final CordaRPCClient client = new CordaRPCClient(nodeAddress,
CordaRPCClientConfiguration.DEFAULT);

        final CordaRPCOps proxy = client.start("user1", "test").getProxy();

        Party receiver = (Party)proxy.partiesFromName("PartyA", false).iterator().next();
        Party sender = (Party) proxy.partiesFromName("PartyB", false).iterator().next();

        Calendar cal = Calendar.getInstance();
        cal.set(Calendar.YEAR, 2019);
        cal.set(Calendar.MONTH, Calendar.MARCH);
        cal.set(Calendar.DAY_OF_MONTH, 11);
        Date d = cal.getTime();
        //Invoke the flow that inserts a payment
        proxy.startFlowDynamic(PaymentFlow.Initiator.class, receiver, "1", "30", d);

        cal = Calendar.getInstance();
        cal.set(Calendar.YEAR, 2019);
        cal.set(Calendar.MONTH, Calendar.MARCH);
        cal.set(Calendar.DAY_OF_MONTH, 11);
        d = cal.getTime();
        //Invoke the flow that inserts a payment
        proxy.startFlowDynamic(PaymentFlow.Initiator.class, receiver, "2", "1000", d);

    }
}
```

4 Database Contents

The following page shows the seller's database contents after running the agents. (The tables in the database contain additional columns such as *output_index* that are autogenerated by Corda and do not concern us.)

SELECT * FROM Offer;

OUTPUT_INDEX	TRANSACTION_ID	BUYER	ITEM	LINEAR_ID	OID	PRICE	SELLER	TIMESTAMP
0	B221AC8DD0AEB1DE9F07F1BDA4FA568A531D11411A6B812A08685767F9CD63A8	O=PartyB, L=New York, C=US	art	88d7272af32848d9a9be81732f600b74	1	30	O=PartyA, L=London, C=GB	2019-03-10 11:12:24.648
0	C0E67F5708B8B9830748F5D528E960078E9994256260FCAAEF7E860EFD2D891E	O=PartyB, L=New York, C=US	scooter	4b9c2b8595bb4a9eaf039e868096e0f0	2	1000	O=PartyA, L=London, C=GB	2019-03-10 11:12:26.626

(2 rows, 0 ms)

SELECT * FROM Payment;

OUTPUT_INDEX	TRANSACTION_ID	AMOUNT	BUYER	LINEAR_ID	OID	SELLER	TIMESTAMP
0	8686995D2DD99DA1744C04CD27C472EB5539065835D55EB4479CC60C7170415D	30	O=PartyB, L=New York, C=US	92e1a650c19c4f4badfe0c69b622f7f8	1	O=PartyA, L=London, C=GB	2019-03-11 11:11:54.53
0	28F152419FEBA763782658CB1AA03FFA4A771C682716D054B442980C33578A3A	1000	O=PartyB, L=New York, C=US	cc76d8568f894871905356a151d160ce	2	O=PartyA, L=London, C=GB	2019-03-11 11:11:56.203

(2 rows, 1 ms)

SELECT * FROM Delivery;

OUTPUT_INDEX	TRANSACTION_ID	BUYER	LINEAR_ID	OID	SELLER	TIMESTAMP
0	DD3586369559B2B9395D8037C2B707384BB2234A24A11B2AFE85FBD9851A29C7	O=PartyB, L=New York, C=US	ff172e98bff84e2fae84872f7897f4a0	1	O=PartyA, L=London, C=GB	2019-03-12 11:12:26.781
0	E95300C8B4FCDE0E5223D9F98105A059243D81402D9BC2711349B4E84B1E5825	O=PartyB, L=New York, C=US	44b441c504cb471497c826d782fac6c9	2	O=PartyA, L=London, C=GB	2019-03-18 11:12:26.844

(2 rows, 0 ms)

5 Lifecycle Queries and Results

The following pages show the queries compiled from the commitment specification in Section 2 and their results when run on the node database from Section 4.

Two instances of the commitment are *created*, corresponding to each *Offer* event in the database. That's why the query for created instances returns two rows.

Both instances of the commitment are *detached* as a timely *Payment* event is recorded in the database for each *Offer* event (the events are correlated by *oID*, the key). That's why the query for detached instances returns two rows.

Since both instances are detached, there are no *expired* instances. Hence, the query for expired instances returns no rows.

For *oID*=1, a *Delivery* event is recorded within the two-day deadline from the occurrence of the (correlated) *Payment*. That's why the query for *discharged* instances returns one row with contents for *oID*=1.

For *oID*=2, a *Delivery* event is recorded after the two-day deadline from the occurrence of the *Payment*. That's why the query for *violated* instances returns one row with contents for *oID*=2.

Notice that even for a commitment specification as simple as the one in Section 2, the queries generated are quite long and complex. This highlights the high-level nature of commitment specifications and the effort that would be involved in recreating them manually in a general-purpose query language such as SQL.

```
/*For OfferCom, we obtain these queries:*/

/*CREATED*/

SELECT buyer, item, oID, price, seller, timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query0 WHERE
timestamp<NOW();
```

BUYER	ITEM	OID	PRICE	SELLER	TIMESTAMP
O=PartyB, L=New York, C=US	art	1	30	O=PartyA, L=London, C=GB	2019-03-10 11:12:24.648
O=PartyB, L=New York, C=US	scooter	2	1000	O=PartyA, L=London, C=GB	2019-03-10 11:12:26.626

(2 rows, 1 ms)

```
/*DETACHED*/

SELECT buyer, item, oID, price, seller, amount, timestamp FROM (SELECT Query2.buyer, item, Query2.oID, price, Query2.seller, amount,
GREATEST(Query2.timestamp,Query9.timestamp1) AS timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query2
NATURAL JOIN (SELECT amount, buyer, oID, seller ,timestamp AS timestamp1 FROM (SELECT amount, buyer, oID, seller, timestamp FROM (SELECT
amount, Query3.buyer, Query3.oID, Query3.seller, item, price, GREATEST(Query3.timestamp,Query11.timestamp2) AS timestamp FROM (SELECT amount,
buyer, oID, seller, timestamp FROM Payment) AS Query3 NATURAL JOIN (SELECT buyer, item, oID, price, seller ,timestamp AS timestamp2 FROM (SELECT
buyer, item, oID, price, seller, timestamp FROM Offer) AS Query4) AS Query11 WHERE '1000-01-01 00:00:00'<=Query3.timestamp AND
Query3.timestamp<DATEADD('DAY',2,Query11.timestamp2)) AS Query6 WHERE price=amount) AS Query6) AS Query9) AS Query7 WHERE
timestamp<NOW();
```

BUYER	ITEM	OID	PRICE	SELLER	AMOUNT	TIMESTAMP
O=PartyB, L=New York, C=US	art	1	30	O=PartyA, L=London, C=GB	30	2019-03-11 11:11:54.53
O=PartyB, L=New York, C=US	scooter	2	1000	O=PartyA, L=London, C=GB	1000	2019-03-11 11:11:56.203

(2 rows, 165 ms)

```
/*EXPIRED*/

SELECT buyer, item, oID, price, seller, timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM (SELECT Query20.buyer, item,
Query20.oID, price, Query20.seller, amount, GREATEST(Query20.timestamp,Query35.timestamp4) AS timestamp FROM (SELECT buyer, item, oID, price,
seller, timestamp FROM Offer) AS Query20 NATURAL JOIN (SELECT amount, buyer, oID, seller ,timestamp AS timestamp4 FROM (SELECT amount, buyer,
oID, seller, timestamp FROM Payment WHERE '1000-01-01 00:00:00'<=timestamp AND timestamp<'1000-01-01 00:00:00') AS Query26) AS Query35) AS
Query27 UNION SELECT Query30.buyer, Query30.item, Query30.oID, Query30.price, Query30.seller, GREATEST(Query30.timestamp,Query32.timestamp3)
AS timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query37
WHERE NOT EXISTS (SELECT buyer, oID, seller FROM (SELECT amount, Query21.buyer, Query21.oID, Query21.seller, item, price,
GREATEST(Query21.timestamp,Query38.timestamp5) AS timestamp FROM (SELECT amount, buyer, oID, seller, timestamp FROM Payment) AS Query21
NATURAL JOIN (SELECT buyer, item, oID, price, seller ,timestamp AS timestamp5 FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS
Query22) AS Query38 WHERE '1000-01-01 00:00:00'<=Query21.timestamp AND Query21.timestamp<DATEADD('DAY',2,Query38.timestamp5)) AS Query29
WHERE Query37.buyer=Query29.buyer AND Query37.oID=Query29.oID AND Query37.seller=Query29.seller)) AS Query30 NATURAL JOIN (SELECT buyer,
item, oID, price, seller ,timestamp AS timestamp3 FROM (SELECT buyer, item, oID, price, seller ,DATEADD('DAY',2,timestamp) AS timestamp FROM (SELECT
buyer, item, oID, price, seller, timestamp FROM Offer) ASQuery22) AS Query31) AS Query32 UNION SELECT buyer, item, oID, price, seller, timestamp FROM
(SELECT Query13.buyer, item, Query13.oID, price, Query13.seller, amount, GREATEST(Query13.timestamp,Query40.timestamp6) AS timestamp FROM
(SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query13 NATURAL JOIN (SELECT amount, buyer, oID, seller ,timestamp AS timestamp6
FROM (SELECT amount, buyer, oID, seller, timestamp FROM (SELECT amount, Query14.buyer, Query14.oID, Query14.seller, item, price,
GREATEST(Query14.timestamp,Query42.timestamp7) AS timestamp FROM (SELECT amount, buyer, oID, seller, timestamp FROM Payment) AS Query14
NATURAL JOIN (SELECT buyer, item, oID, price, seller ,timestamp AS timestamp7 FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS
Query15) AS Query42 WHERE '1000-01-01 00:00:00'<=Query14.timestamp AND Query14.timestamp<DATEADD('DAY',2,Query42.timestamp7)) AS Query17
WHERE NOT(price=amount)) AS Query17) AS Query40) AS Query18) AS Query24 WHERE timestamp<NOW();
```

BUYER	ITEM	OID	PRICE	SELLER	TIMESTAMP

(no rows, 455 ms)

```
/*DISCHARGED*/

SELECT buyer, item, oID, price, seller, timestamp FROM (SELECT Query44.buyer, item, Query44.oID, price, Query44.seller,
GREATEST(Query44.timestamp,Query50.timestamp8) AS timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query44
NATURAL JOIN (SELECT buyer, oID, seller ,timestamp AS timestamp8 FROM (SELECT Query45.buyer, Query45.oID, Query45.seller, amount,
GREATEST(Query45.timestamp,Query52.timestamp9) AS timestamp FROM (SELECT buyer, oID, seller, timestamp FROM Delivery) AS Query45 NATURAL
JOIN (SELECT amount, buyer, oID, seller ,timestamp AS timestamp9 FROM (SELECT amount, buyer, oID, seller, timestamp FROM Payment) AS Query46) AS
Query52 WHERE '1000-01-01 00:00:00'<=Query45.timestamp AND Query45.timestamp<DATEADD('DAY',2,Query52.timestamp9)) AS Query47) AS Query50)
AS Query48 WHERE timestamp<NOW();
```

BUYER	ITEM	OID	PRICE	SELLER	TIMESTAMP
O=PartyB, L=New York, C=US	art	1	30	O=PartyA, L=London, C=GB	2019-03-12 11:12:26.781

(1 row, 36 ms)

```
/*VIOLATED*/

SELECT buyer, item, oID, price, seller, amount, timestamp FROM (SELECT buyer, item, oID, price, seller, amount, timestamp FROM (SELECT Query59.buyer,
item, Query59.oID, price, Query59.seller, amount, GREATEST(Query59.timestamp,Query73.timestamp11) AS timestamp FROM (SELECT Query54.buyer, item,
Query54.oID, price, Query54.seller, amount, GREATEST(Query54.timestamp,Query75.timestamp12) AS timestamp FROM (SELECT buyer, item, oID, price,
seller, timestamp FROM Offer) AS Query54 NATURAL JOIN (SELECT amount, buyer, oID, seller ,timestamp AS timestamp12 FROM (SELECT amount, buyer,
oID, seller, timestamp FROM (SELECT amount, Query55.buyer, Query55.oID, Query55.seller, item, price,
GREATEST(Query55.timestamp,Query77.timestamp13) AS timestamp FROM (SELECT amount, buyer, oID, seller, timestamp FROM Payment) AS Query55
NATURAL JOIN (SELECT buyer, item, oID, price, seller ,timestamp AS timestamp13 FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS
Query56) AS Query77 WHERE '1000-01-01 00:00:00'<=Query55.timestamp AND Query55.timestamp<DATEADD('DAY',2,Query77.timestamp13)) AS Query58
WHERE price=amount) AS Query58) AS Query75) AS Query59 NATURAL JOIN (SELECT buyer, oID, seller ,timestamp AS timestamp11 FROM (SELECT
buyer, oID, seller, timestamp FROM Delivery WHERE '1000-01-01 00:00:00'<=timestamp AND timestamp<'1000-01-01 00:00:00') AS Query64) AS Query73)
AS Query65 UNION SELECT Query68.buyer, item, Query68.oID, price, Query68.seller, Query68.amount,
GREATEST(Query68.timestamp,Query70.timestamp10) AS timestamp FROM (SELECT buyer, item, oID, price, seller, amount, timestamp FROM (SELECT
Query54.buyer, item, Query54.oID, price, Query54.seller, amount, GREATEST(Query54.timestamp,Query80.timestamp14) AS timestamp FROM (SELECT
buyer, item, oID, price, seller, timestamp FROM Offer) AS Query54 NATURAL JOIN (SELECT amount, buyer, oID, seller ,timestamp AS timestamp14 FROM
(SELECT amount, buyer, oID, seller, timestamp FROM (SELECT amount, Query55.buyer, Query55.oID, Query55.seller, item, price,
GREATEST(Query55.timestamp,Query82.timestamp15) AS timestamp FROM (SELECT amount, buyer, oID, seller, timestamp FROM Payment) AS Query55
NATURAL JOIN (SELECT buyer, item, oID, price, seller ,timestamp AS timestamp15 FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS
Query56) AS Query82 WHERE '1000-01-01 00:00:00'<=Query55.timestamp AND Query55.timestamp<DATEADD('DAY',2,Query82.timestamp15)) AS Query58
WHERE price=amount) AS Query58) AS Query80) AS Query79 WHERE NOT EXISTS (SELECT buyer, oID, seller FROM (SELECT Query60.buyer,
Query60.oID, Query60.seller, amount, GREATEST(Query60.timestamp,Query84.timestamp16) AS timestamp FROM (SELECT buyer, oID, seller, timestamp
FROM Delivery) AS Query60 NATURAL JOIN (SELECT amount, buyer, oID, seller ,timestamp AS timestamp16 FROM (SELECT amount, buyer, oID, seller,
timestamp FROM Payment) AS Query61) AS Query84 WHERE '1000-01-01 00:00:00'<=Query60.timestamp AND
Query60.timestamp<DATEADD('DAY',2,Query84.timestamp16)) AS Query67 WHERE Query79.buyer=Query67.buyer AND Query79.oID=Query67.oID AND
Query79.seller=Query67.seller)) AS Query68 NATURAL JOIN (SELECT amount, buyer, oID, seller ,timestamp AS timestamp10 FROM (SELECT amount,
buyer, oID, seller ,DATEADD('DAY',2,timestamp) AS timestamp FROM (SELECT amount, buyer, oID, seller, timestamp FROM Payment) ASQuery61) AS
Query69) AS Query70) AS Query62 WHERE timestamp<NOW();
```

BUYER	ITEM	OID	PRICE	SELLER	AMOUNT	TIMESTAMP
O=PartyB, L=New York, C=US	scooter	2	1000	O=PartyA, L=London, C=GB	1000	2019-03-13 11:11:56.203

(1 row, 4751 ms)