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)
  Payment(amount, buyer, oID, seller)
  Delivery(buyer, oID, seller)

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.
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");

        //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);
    }
}
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:

<table>
<thead>
<tr>
<th>OUTPUT_INDEX</th>
<th>TRANSACTION_ID</th>
<th>BUYER</th>
<th>ITEM</th>
<th>LINEAR_ID</th>
<th>OID</th>
<th>PRICE</th>
<th>SELLER</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C0E57F5708B09830748F5D528E3606078E994256260FCAAEF7E8605DD891D</td>
<td>O=PartyB, L=New York, C=US</td>
<td>scooter</td>
<td>456c2b559f6a4fae039a8888696e0f12</td>
<td>2</td>
<td>1000</td>
<td>O=PartyA, L=London, C=GB</td>
<td>2019-03-10 11:12:26.626</td>
</tr>
</tbody>
</table>

(2 rows, 0 ms)

### SELECT * FROM Payment:

<table>
<thead>
<tr>
<th>OUTPUT_INDEX</th>
<th>TRANSACTION_ID</th>
<th>AMOUNT</th>
<th>BUYER</th>
<th>LINEAR_ID</th>
<th>OID</th>
<th>SELLER</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>886956D2DD9DA1744C04C27C472EB539065635D5EB4479CC60C717D155D30</td>
<td>30</td>
<td>O=PartyB, L=New York, C=US</td>
<td>926f1e65c10a4f4badfe0c6986227f01</td>
<td>1</td>
<td>O=PartyA, L=London, C=GB</td>
<td>2019-03-11 11:12:54.639</td>
</tr>
</tbody>
</table>

(2 rows, 1 ms)

### SELECT * FROM Delivery:

<table>
<thead>
<tr>
<th>OUTPUT_INDEX</th>
<th>TRANSACTION_ID</th>
<th>BUYER</th>
<th>LINEAR_ID</th>
<th>OID</th>
<th>SELLER</th>
<th>TIMESTAMP</th>
</tr>
</thead>
</table>

(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();

<table>
<thead>
<tr>
<th>BUYER</th>
<th>ITEM</th>
<th>OID</th>
<th>PRICE</th>
<th>SELLER</th>
<th>TIMESTAMP</th>
</tr>
</thead>
</table>
(2 rows, 1 ms)

/*DETACHED*/
SELECT buyer, item, oID, price, seller, amount, timestamp FROM (SELECT buyer, item, oID, price, seller, amount, GREATEST(Query2.buyer,Query9.timestamp1) AS timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query2 NATURAL JOIN (SELECT amount, buyer, oID, seller, timestamp FROM Payment) AS Query9 WHERE NOT(amount=price)) AS Query10 WHERE timestamp<NOW();

<table>
<thead>
<tr>
<th>BUYER</th>
<th>ITEM</th>
<th>OID</th>
<th>PRICE</th>
<th>SELLER</th>
<th>AMOUNT</th>
<th>TIMESTAMP</th>
</tr>
</thead>
</table>
(2 rows, 165 ms)

/*EXPIRED*/
SELECT buyer, item, oID, price, seller, timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query1 WHERE '1000-01-01 00:00:00'<=Query1.timestamp AND timestamp<'1000-01-01 00:00:00') AS Query2 UNION SELECT Query3.buyer, Query3.item, Query3.oID, Query3.price, Query3.seller, GREATEST(Query3.timestamp,Query38.timestamp5) AS timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query38 UNION SELECT buyer, item, oID, price, seller, timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query3 WHERE timestamp<NOW();

<table>
<thead>
<tr>
<th>BUYER</th>
<th>ITEM</th>
<th>OID</th>
<th>PRICE</th>
<th>SELLER</th>
<th>AMOUNT</th>
<th>TIMESTAMP</th>
</tr>
</thead>
</table>
(2 rows, 455 ms)

/*DISCHARGED*/
SELECT buyer, item, oID, price, seller, timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query11 WHERE '1000-01-01 00:00:00'<=Query11.timestamp AND Query11.timestamp<DATEADD('DAY',2,Query13.timestamp11) AS Query13 WHERE NOT EXISTS (SELECT buyer, oID, seller FROM (SELECT amount, Query21.buyer, Query21.oID, Query21.seller, item, price, GREATEST(Query21.timestamp,Query22.timestamp7) AS timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query22) AS Query14 WHERE '1000-01-01 00:00:00'<=Query14.timestamp AND timestamp<'1000-01-01 00:00:00') AS Query15 AS Query40) AS Query18) AS Query24 WHERE timestamp<NOW();

<table>
<thead>
<tr>
<th>BUYER</th>
<th>ITEM</th>
<th>OID</th>
<th>PRICE</th>
<th>SELLER</th>
<th>TIMESTAMP</th>
</tr>
</thead>
</table>
(1 row, 36 ms)

/*VIOLATED*/
SELECT buyer, item, oID, price, seller, amount, timestamp FROM (SELECT buyer, item, oID, price, seller, amount, GREATEST(Query54.buyer,Query54.timestamp12) AS timestamp FROM (SELECT buyer, item, oID, price, seller, timestamp FROM Offer) AS Query54) AS Query12 WHERE timestamp<NOW();

<table>
<thead>
<tr>
<th>BUYER</th>
<th>ITEM</th>
<th>OID</th>
<th>PRICE</th>
<th>SELLER</th>
<th>AMOUNT</th>
<th>TIMESTAMP</th>
</tr>
</thead>
</table>
SELECT buyer, item, oID, price, seller, amount, GREATEST(timestamp, timestamp10) AS timestamp FROM (SELECT amount, buyer, oID, seller, timestamp FROM Payment) AS Query61 NATURAL JOIN (SELECT amount, buyer, oID, seller, timestamp FROM Offer) AS Query54 NATURAL JOIN (SELECT amount, buyer, oID, seller, timestamp FROM Delivery) AS Query60 WHERE '1000-01-01 00:00:00' <= Query60.timestamp AND Query60.timestamp < DATEADD('DAY', 2, Query79.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 FROM Payment) AS Query61) AS Query69) AS Query70) AS Query72 WHERE timestamp < NOW();

<table>
<thead>
<tr>
<th>BUYER</th>
<th>ITEM</th>
<th>OID</th>
<th>PRICE</th>
<th>SELLER</th>
<th>AMOUNT</th>
<th>TIMESTAMP</th>
</tr>
</thead>
</table>

(1 row, 4751 ms)