Chapter 4

Proof of Concept

In previous chapters we proposed a software architecture based on connection of DEMO methodology and blockchain smart contracts In this part we describe a proof of concept using a financial transaction, the process of mortgage, implemented in Ethereum Solidity programing language for smart contracts. The proof of concept is composed of

- **Process description** - contains description of the mortgage process and describes the possible benefits
- **DEMO models** - analyses mortgage process and DEMO models of the mortgage process based on its description
- **DEMO and Smart Contract** - connection of DEMO and smart contract and implementation of smart contract in Ethereum Solidity based on DEMO methodology.

Please take into consideration that this implementation serves as proof of concept, there is need of more extensive research and testing on real blockchain. This proof of concept also makes use hypotheses or expectations regarding blockchain technology such as digital identity.

4.1 Technologies used

4.1.0.1 Solidity

Solidity [31] is a programming language to implement smart contracts specially designed for the Ethereum Virtual Machine (EVM). It is a Turing-complete high-level language compiled to the EVM bytecode.

Solidity was chosen because it is developed under Ethereum and is the most used language for smart contracts for EVM, although there are some other languages, Solidity is the most developed language amongst them.

The building block in solidity is a contract which is similar to class in object-oriented programming. Contract contains persistent data in state variables, functions to operate on this data and it also supports inheritance. Contract can further contain function modifiers, events, struct types and other structures to allow implementation of complex contracts and full usage of EVM and blockchain capabilities.

A smart contract written in solidity can be created either through a ethereum transaction or by another already running contract, just like we would create an instance of a class. Either way the contract code is than compiled to the EVM bytecode, new transaction is created holding the code and deployed to blockchain, returning the address of the contract for further interaction.

Solidity contracts can also be created, deployed and interacted with programatically using the JavaScript API web3.js, which is an ethereum compatible library implementing the Generic JSON RPC spec, that provides a convenient interface for communication with Ethereum nodes [32].

4.1.0.2 Remix

Remix is a browser-based IDE for creating smart contracts with integrated debugging and testing environment. Remix offers development, compilation and deployment of solidity contracts as well as access to already deployed contracts. The testing environment allows running the transactions in a sandbox blockchain in the browser with JavaScript VM with a possibility to switch between virtual accounts and spend virtual ethers for full smart contract testing [33].

4.2 Process description

In this part the financial transaction is described and discussed to evaluate possible benefits of modeling this process with DEMO and implementing with smart contract. The chosen financial transaction is the process of mortgage, as it appears as a good candidate to demonstrate the advantages and compatibility of DEMO and smart contracts.

The mortgage contract is a rather confusing and complicated process involving several parties, dependent processes, level of trust between parties and a lot of documents proving results of auxiliary processes and above all notarization. This aspects all contribute to overall complexity and costs of the process. Thus it appears as a good use case where modeling by DEMO would capture the essence of the process and smart contract could offer a automated notarization, data sharing between parties and payment processing reducing the need of manual processes as illustrated by 4.1.

The description of the mortgage process is based on research made of several on-line mortgage guides [34] [35] and consultation with real estate agent. The description of the process was than modeled by flow chart to fully

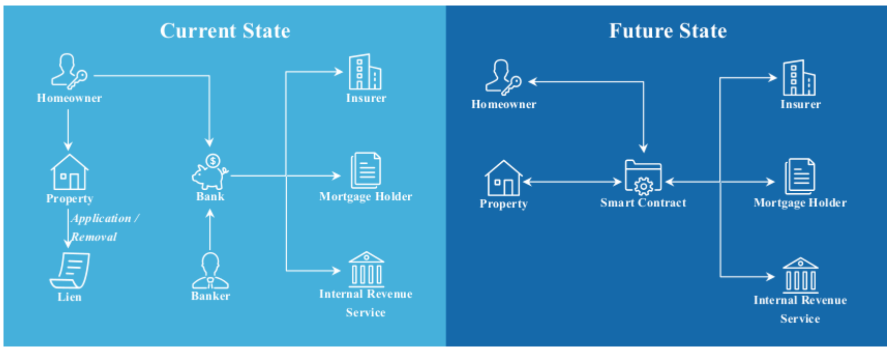


Figure 4.1: Mortgage process changed using smart contract [5]

understand and illustrate the process. For needed illustration we omit the common pre-approval faze.

Mortgage process description

1. **step** - Buyer applies for mortgage in bank. He fills in an application with basic information (personal data, employment, income, marital status, etc.). Based on primary application bank requests further documents, needed for full client screening (credit report, account statements, income certificates/invoices, etc.) and documents for desired property such as reservation contract, property appraisal, draft of sales contract. When the bank internally approves the mortgage, they prepare the Mortgage contract which states all the details such as the finance charge, annual percentage rate (APR), number of payments you will make, amount of each payment (for fixed-rate loans), late payment charges that may apply and total amount you will pay in principal and finance charges over the life of the loan and further conditions to be met to issue the funds such as property insurance and states the closing date.

2. **step** - Once the mortgage is approved by bank and closing is scheduled, client or bank secures a homeowner insurance for property.

3. **step** - At the closing, all of the needed parties (loaner, buyer, seller) meet at a notary. The sale contract, the mortgage contract and documents for cadastre of real estates are signed. After the documents for cadastre and lien are registered at the cadaster, the bank issues the funds and the deal is finalized.

4. **step** - The loan servicing, the steps taken to maintain a loan from the time it is closed until it is paid off, then starts. Client is obligated to pay monthly pay-off payments. Once the mortgage is fully paid-off the bank releases the property lien and client becomes a rightful owner of the property.

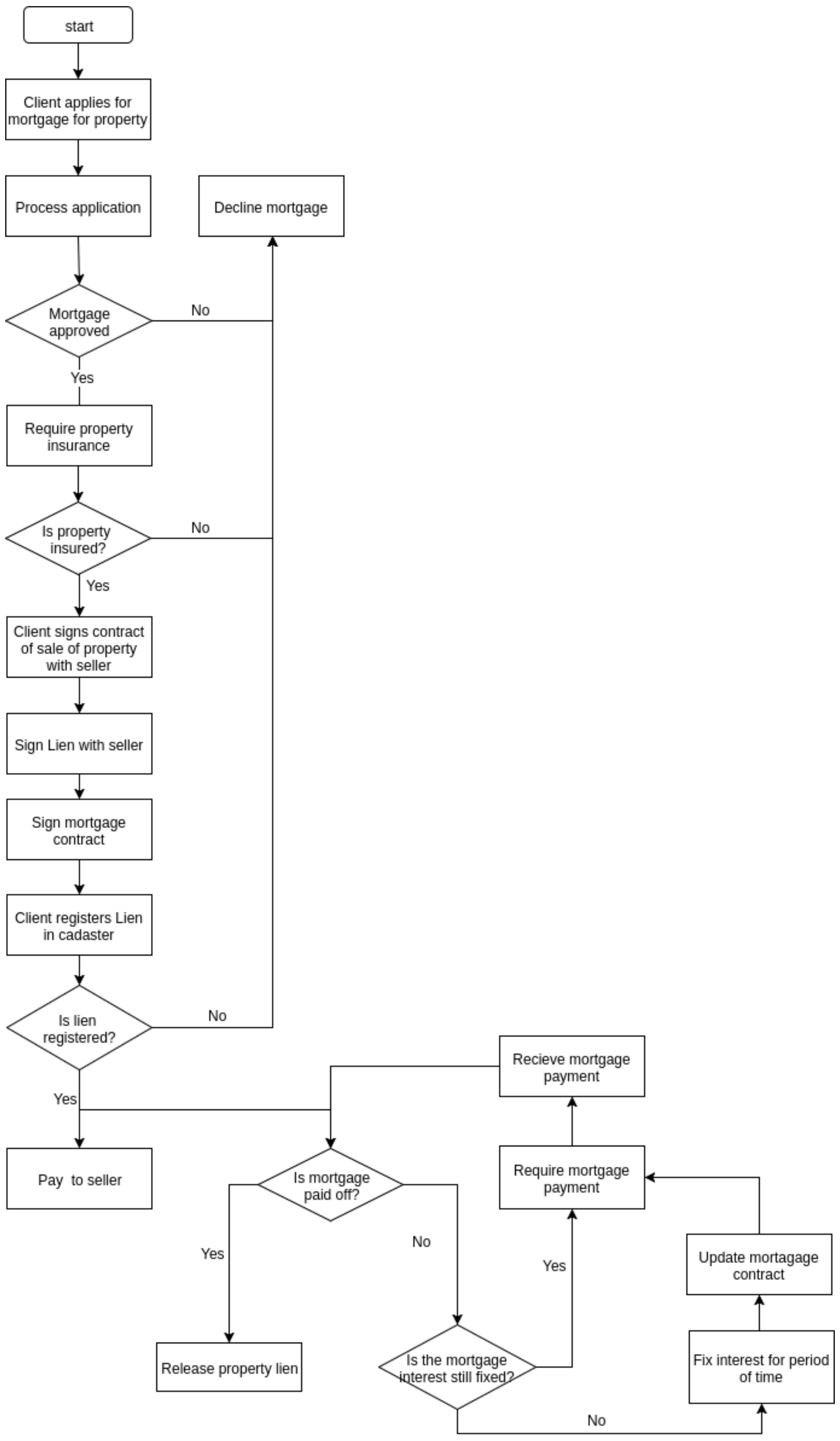


Figure 4.2: Mortgage process flow chart

4.3 DEMO Model

Once the mortgage process is described and conceived we can start to reveal the essence of the process and model it using DEMO methodology. We first analyze the process and than create all needed models according to the proposed method in section chapter 3.

Mortgage process analysis

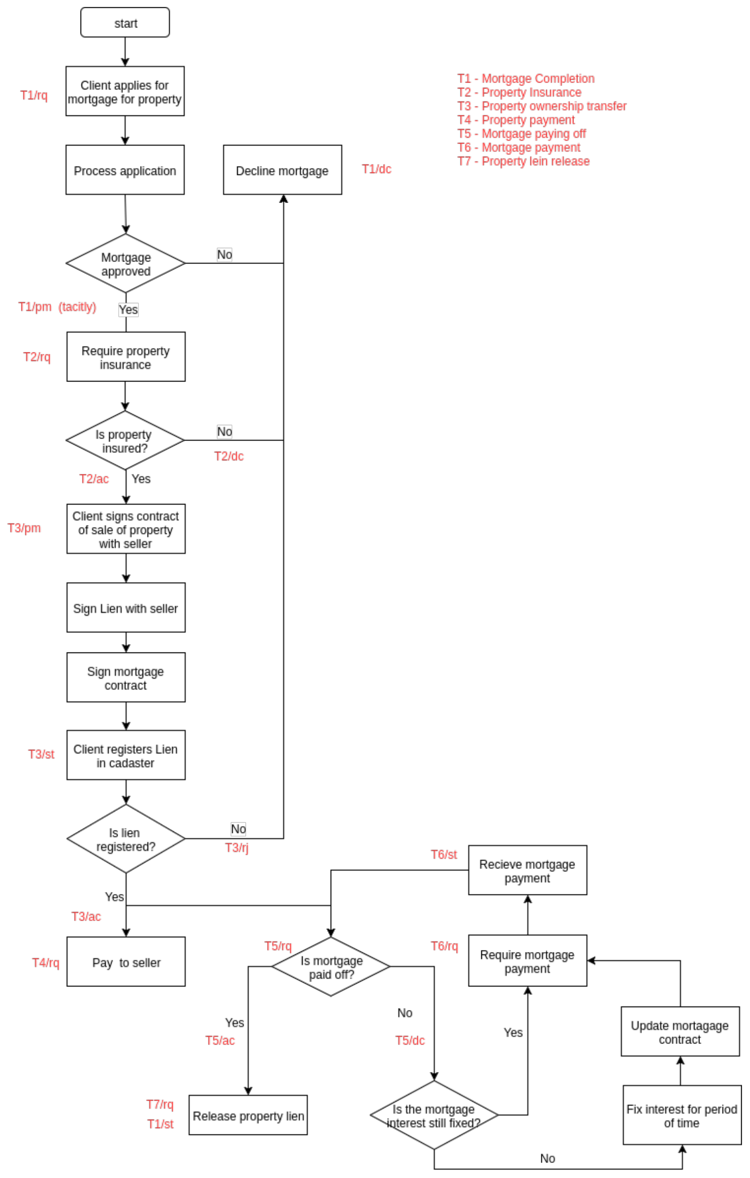


Figure 4.3: Mortgage process flow chart with identified C-acts

The most important part of revealing the essence of a process is to identify its core transactions and distinguish documental, informational and original acts. We will use the flow chart to analyze the process, for illustration we only mark out C-acts fig. 4.3

As we can see in fig. 4.3 there were seven different transactions identified with a transaction Mortgage completion representing the main process covering up all other transactions.

Construction model

Based on the mortgage process analysis and identified transactions we can produce the Transaction Product Table tab. 4.1, Bank content table tab. 4.2 and Actor Transaction Diagram fig. 4.4.

Transaction kind	**Production type**
T1 Mortgage completion	P1 Mortgage is completed
T2 Property insurance	P2 Property is insured
T3 Property ownership transfer	P3 Property ownership is transferred
T4 Property payment	P4 Property is paid
T5 Mortgage paying off	P5 Mortgage is paid off
T6 Mortgage payment	P6 Mortgage is paid
T7 Property lien release	P7 Property lien is released

Table 4.1: The Transaction Production Table

Bank	**Independent/Dependent facts**
T1 Mortgages completion	MORTGAGE Mortgage is completed The receiver of Mortgage The property of Mortgage The amount of Mortgage The annual percentage rate of Mortgage The final amount to pay off for Mortgage The number of payments for Mortgage The amount of payment for Mortgage

AT1 Client facts	CLIENT personal data of Client income of Client employment information of Client marital status of Client credit report of Client account statements of Client Income certificates of Client
T2 Property Insurance	PROPERTY Property is insured
T3 Property ownership transfer	PROPERTY Property ownership is transferred The owner of the Property The lien of the Property The amount to pay for Property
T4 Property payment	PROPERTY Property is paid The amount paid for Property
T5 Mortgage paying off	MORTGAGE Mortgage is paid off The amount paid off for Mortgage
T6 Mortgage payment	MORTGAGE Mortgage is paid The amount paid for Mortgage

Table 4.2: The Bank Contents Table

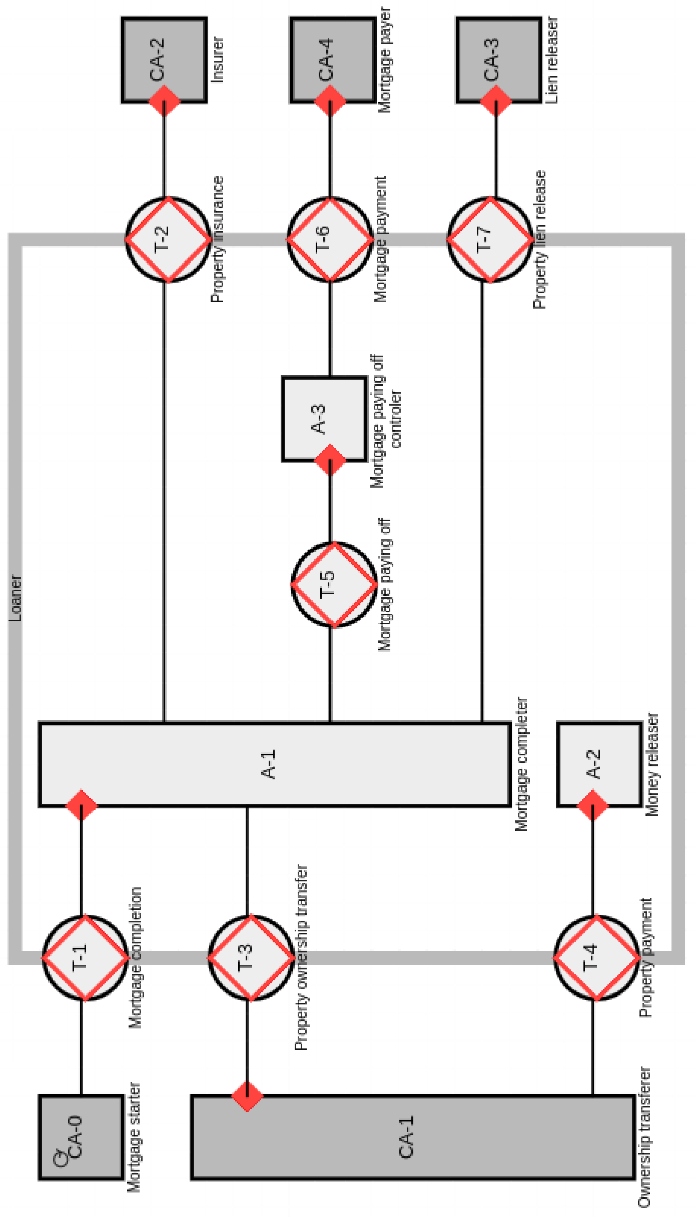


Figure 4.4: The Actor Transaction Diagram

Process model

To fully understand the transactions flow we use the Process Structure Diagram fig. 4.5 that illustrates the dependencies between transaction‘s states. This conditions for the order of transaction‘s state are important when implementing the smart contract.

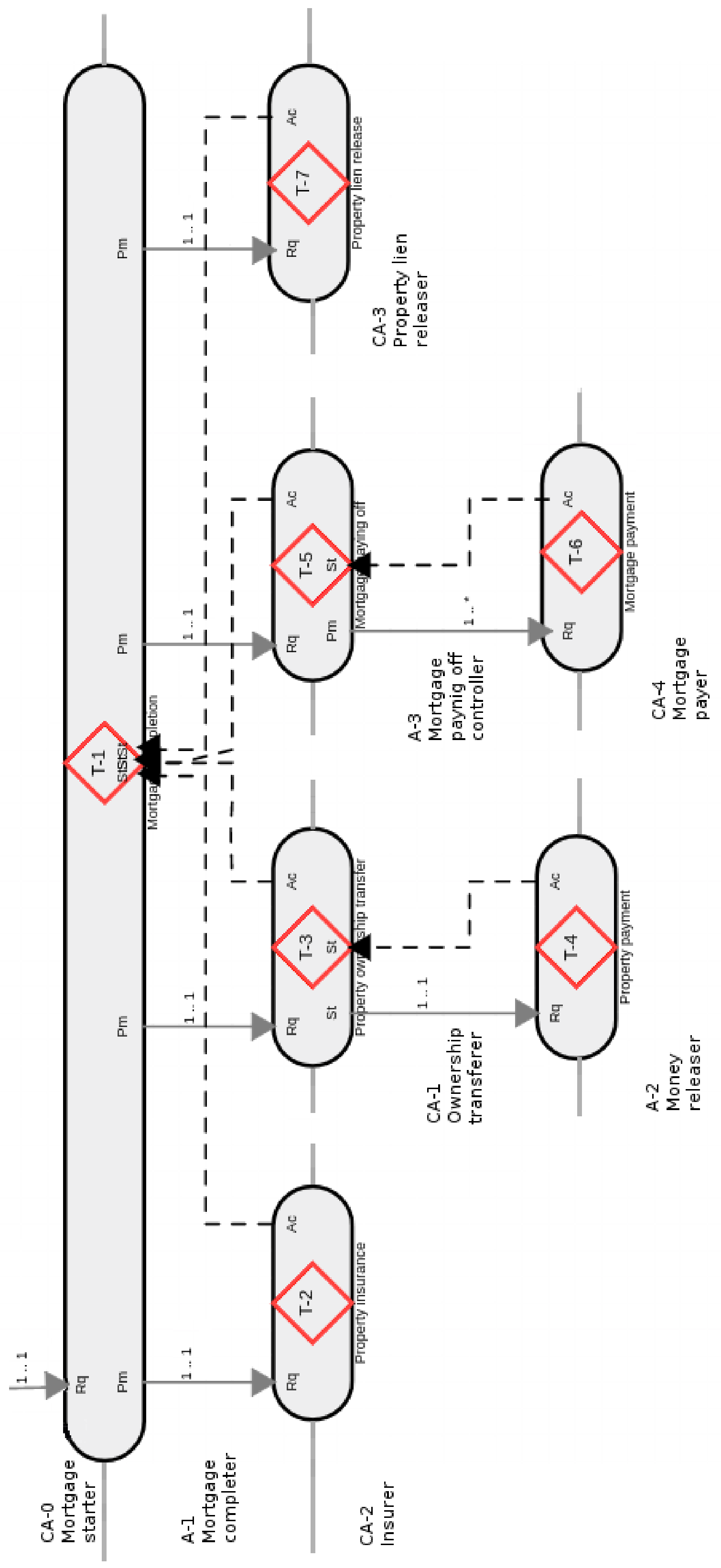


Figure 4.5: The Process Structure Diagram

Fact model

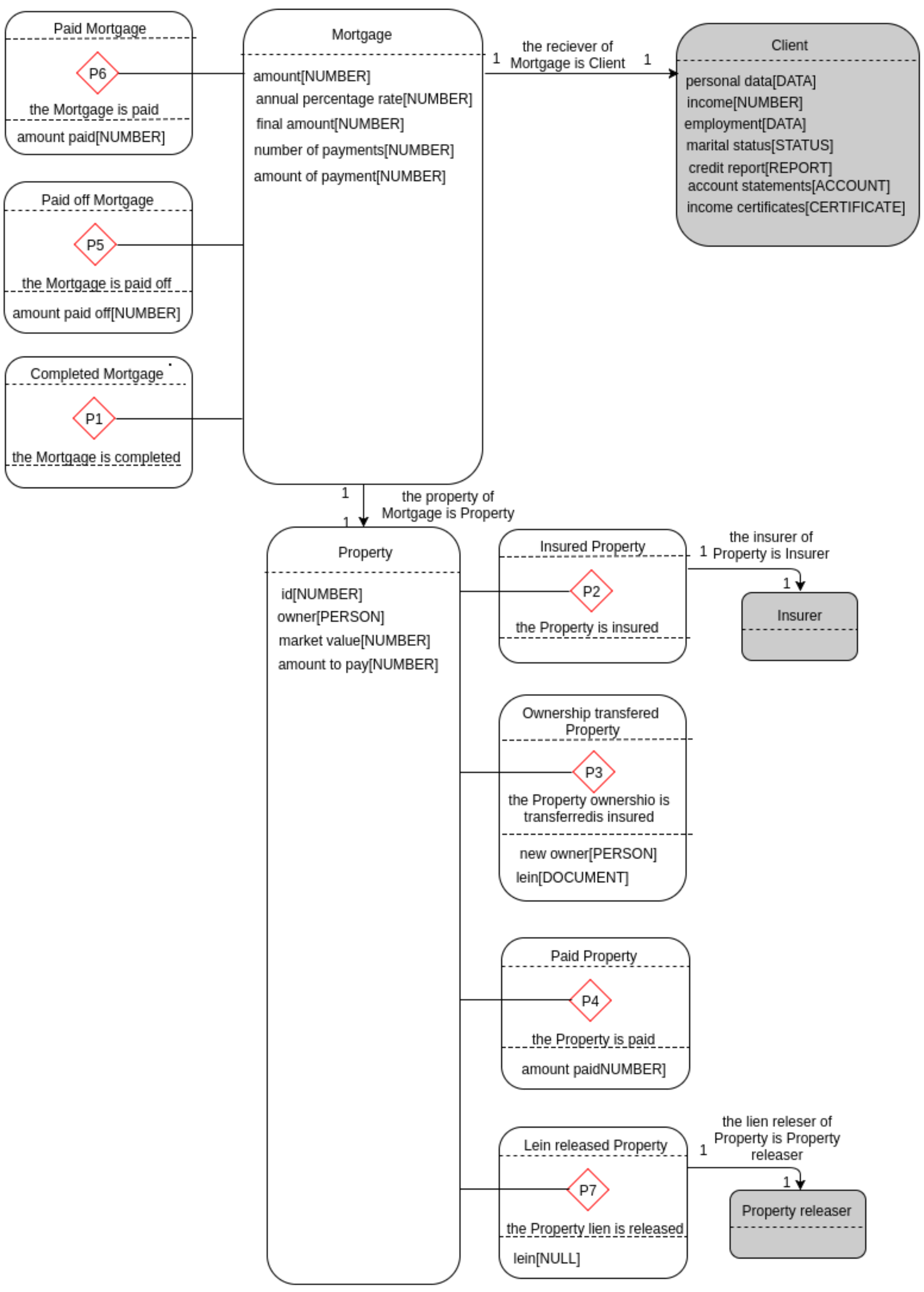


Figure 4.6: The Object Fact Diagram

Action model

The Action rules are a final DEMO model and is the one most needed for smart contract creation. The structure and content of the smart contract can be directly mapped to each action rule. By creating the action rules we basically create the logic on which the smart contract operates.

when	mortgage completion for new Mortgage is requested (T1/rq) with the receiver of Mortgage is Client the property of Mortgage is Property
assess	justice: the performer of the request is the receiver of Mortgage sincerity: <no specific conditions> truth: Mortgage for property for receiver is approved
if	complying with request is considered justifiable
then	promise mortgage completion of Mortgage [T1/pm]
else	decline mortgage completion for Mortgage [T1/dc]

Table 4.3: The Action Rule 1

when	mortgage completion for Mortgage is promised (T1/pm)
assess	justice: the performer of the promise is the mortgage completer of Mortgage sincerity: <no specific conditions> truth: <no specific conditions>
if	complying with request is considered justifiable
then	request property insurance of Property [T2/rq] with the property to insure is the property of Mortgage request property ownership transfer of Property [T3/rq] with the property to transfer ownership is the property of Mortgage the new owner to be of property is client of Mortgage the lien of the property will be for the loaner of Mortgage the amount to release to pay for property is the amount of Mortgage

Table 4.4: The Action Rule 2

when	property insurance for Property is stated (T2/pm)
assess	justice: the performer of the state is the insurer of Mortgage sincerity: <no specific conditions > truth: The property insured is the property of Mortgage
if	complying with request is considered justifiable
then	accept property insurance of Property [T2/ac]
else	reject property insurance of Property [T2/rj]

Table 4.5: The Action Rule 3

when	property ownership transfer is stated (T3/st)
assess	justice: the performer of the state is the ownership transferer of Property sincerity: <no specific conditions > truth: the property of transferred ownership is the property of Mortgage the new owner of property is client of Mortgage the lien of the property is for the loaner of Mortgage
if	complying with request is considered justifiable
then	accept property ownership transfer of Property [T3/ac]
else	reject property ownership transfer of Property [T3/rj]

Table 4.6: The Action Rule 4

when	mortgage completion for Mortgage is promised (T1/pm) while property insurance is accepted property ownership transfer is accepted
assess	justice: the performer of the promise is the mortgage completer of Mortgage sincerity: <no specific conditions > truth: textless no specific conditions > the new owner of property is client of Mortgage the lien of the property is for the loaner of Mortgage the amount released to pay for property is the amount of Mortgage

if	complying with request is considered justifiable
then	request mortgage paying off for Mortgage [T5/rq] with the amount to pay off is the amount to pay off of Mortgage

Table 4.7: The Action Rule 5

when	mortgage completion for Mortgage is promised (T1/pm) while property insurance is accepted property ownership transfer is accepted
assess	justice: the performer of the promise is the mortgage completer of Mortgage sincerity: <no specific conditions > truth: the new owner of property is client of Mortgage the lien of the property is for the loaner of Mortgage the amount released to pay for property is the amount of Mortgage
if	complying with request is considered justifiable
then	request mortgage paying off for Mortgage [T5/rq] with the amount to pay off is the amount to pay off of Mortgage

Table 4.8: The Action Rule 6

when	mortgage paying off for Mortgage is requested (T5/rq) with the amount to pay off equals to the amount to pay off for Mortgage
assess	justice: the performer of the request is mortgage completer of Mortgage sincerity: <no specific conditions > truth: the amount to pay off is equal the amount to pay off of Mortgage
if	complying with request is considered justifiable
then	promise mortgage completion of Mortgage [T5/pm]
else	decline mortgage completion for Mortgage [T5/dc]

Table 4.9: The Action Rule 7

when	mortgage paying off for Mortgage is promised (T5/pm)
assess	justice: the performer of the promise is the mortgage paying off controller of Mortgage sincerity: <no specific conditions > truth: <no specific conditions >
if	complying with request is considered justifiable
then	request mortgage payment for Mortgage [T6/rq] with the amount to pay for Mortgage is equal to the amount of each payment for Mortgage

Table 4.10: The Action Rule 8

when	mortgage payment for Mortgage is stated (T6/st) with the amount paid for Mortgage is some Money
assess	justice: the performer of the state is the payer for Mortgage sincerity: <no specific conditions > truth: with the amount paid for Mortgage is equal to the amount of each payment for Mortgage
if	complying with request is considered justifiable
then	accept mortgage payment for Mortgage [T6/ac]
else	reject mortgage payment for Mortgage [T6/rj]

Table 4.11: The Action Rule 9

when	mortgage paying off for Mortgage is promised (T5/pm) while mortgage payment for Mortgage is accepted
assess	justice: the performer of the promise is the mortgage paying off controller of Mortgage sincerity: <no specific conditions > truth: The summary of amount of mortgage payments for Mortgage is equal to the the amount to pay off for Mortgage
if	complying with request is considered justifiable
then	execute mortgage paying off for Mortgage [T5/ex] state mortgage paying off for Mortgage [T5/st]

else	request mortgage payment for Mortgage [T6/rq] with the amount to pay for Mortgage is equal to the amount of each payment for Mortgage

Table 4.12: The Action Rule 10

when	mortgage paying off for Mortgage is stated(T5/st)
assess	justice: the performer of the state is the mortgage paying off controller of Mortgage sincerity: <no specific conditions > truth: the amount paid off is equal to the amount to pay off for Mortgage
if	complying with request is considered justifiable
then	accept mortgage paying off for Mortgage [T3/ac] request property lien release of Property [T7/rq] with the property to release lien of is the property of Mortgage the lien releaser is the some Property Lien Releaser
else	reject mortgage paying off for Mortgage [T3/rj]

Table 4.13: The Action Rule 11

when	property lien release of Property is stated (T7/st)
assess	justice: the performer of the state is the property lien releaser of property sincerity: <no specific conditions > truth: the property to be released is equal to the property of Mortgage The lien released is the lien of loaner of Mortgage
if	complying with request is considered justifiable
then	accept property lien release of Property [T7/ac] execute mortgage completion of Mortgage [T1/ex] state mortgage completion of Mortgage [T1/st]
else	reject property lien release of Property [T7/rj]

Table 4.14: The Action Rule 12

4.4 DEMO and Smart Contract

Analyzing the process through DEMO models we can now evaluate the possible use of smart contract.

For illustration we will assign actors to the actor roles as follows:

- Loaner: A-1, A-2
- Client: CA-0, CA-1, CA-4
- Insurer: CA-2
- Property Releaser: CA-2

For illustration of possible benefits of blockchain the implementation relies on these pre-requisites:

- Implementation of digital identity
- Adaption of public key infrastructure between actors
- Possibility of mortgage payment in cryptocurrency

Here is what smart contracts based on DEMO models described by this theses look like. For clear connection to the DEMO methodology we use terms from the DEMO model. Full solidity code can be found on the enclosed CD.

4.4.1 Generic Transaction as Smart Contract

We created a generic Transaction contract that serves as a parent class for contracts based on transactions. It holds some general functionality such as common state variables: name, product fact, current transaction fact, initiator and executor.

```
contract Transaction {

  enum C_facts { Initial, Requested, Promised, Declined, Stated,
  Accepted, Rejected }

  string name;
  string P_fact;
  C_facts public current_c_fact;
  address public initiator;
  address public executor;
```

The contract defines access modifiers to authorize the actor evoking the function call or simple coordination facts control. The authorization pattern is a common design pattern for Ethereum smart contracts to restrict code execution to only certain address according to the caller address accessible in `msg.sender` [24].

```
modifier onlyExecutor {
  require(msg.sender == executor);
  _;
}

modifier isRequested {
  require (current_c_fact == C_facts.Requested);
  _;
}
```

Further it contains functions to represent coordination acts of the standard transaction pattern. Here we use Solidity events, which allow libraries such as web3.js to listen to changes in transaction's facts and also to log the new coordination facts and thus keep the history of transaction states.

```
event NewFact(
  C_facts c_fact,
  string transaction_name
);

function request() internal {
  current_c_fact = C_facts.Requested;
  NewFact(C_facts.Requested, name);
}
```

4.4.2 Mortgage Completion as Smart Contract

We have 7 consequent transactions with the Mortgage completion being the parent transaction. The Mortgage completion transaction can be looked at as the contract between the Loaner and the Client with conditions needed to be fulfilled for the contract to be finished. This contract‘s underlying transaction needs notarization, data sharing and trustless control of the execution. So first transaction where we can use smart contract is Mortgage completion. The DEMO model itself would not change.

The idea is that this smart contract would be deployed to blockchain by the Loaner of the Mortgage, after the T1-Mortgage completion is requested.

We created a contract called `MortgageCompletion`. The state variables Mortgage and Property based on the BCT (fig. 4.2) and OFD (fig. 4.6) models represent a data needed for the contract. Further we declare that the contract has four sub-transactions its state depends on according to the PSD model (fig. 4.5). Three of them are represented by SubTransaction structure and one is represented by another contract, which will be discussed later. Finally we declare the addresses of the actors for the sub-transactions, for control of access and conditions control.

```
contract MortgageCompletion is Transaction {

  struct Property {
    string id;
    uint value;
    address owner;
    address lien;
    bool insured;
  }

  struct Mortgage {
    uint amount;
    uint annual_percentage_rate;
    uint final_amount;
    uint amount_of_payment;
  }

  Mortgage public mortgage;
  Property public property;

  SubTransaction public propertyInsurance = SubTransaction(
   "Property Insurance", C_facts.Initial);
  SubTransaction public propertyOwnershipTransfer = SubTransaction(
   "Property Ownership Transfer", C_facts.Initial);
  SubTransaction public propertyLeinRelease = SubTransaction(
   "Property Lein Release", C_facts.Initial);
  MortgagePaingOff public mortgagePaingOff;

  address client;
  address insurer;
  address property_releaser;
```

In the constructor of `MortgageCompletion` we declare the state variables that are already known and we define the actors addresses which are definite. (Please note that the address used are only for illustration).

```
function MortgageCompletion() Transaction("Mortgage completion",
         "Mortgage is completed",
         0x014723a09acff6d2a60dcdf7aa4aff308fddc160c,
         msg.sender)
```

```
{
  client = initiator;
  insurer = 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c;
  property_releaser = 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c;
  property = Property("1",
             0xdd870fa1b7c4700f2bd7f44238821c26f7392148,
             0x0, 1000000, false);
}
```

The last part are the transaction execution functions. The logic of each step of a transaction is based on the Action rules and data changes based on BCT (fig. 4.2) and OFD (fig. 4.6) models.

The function `promiseMortgageCompletion` represents the promise of Mortgage completion based on the Action Rule 1 (tab. 4.3) and the Action Rule 2 (tab. 4.4). Here the function also performs implicit transaction execution by requesting Property insurance and Property ownership transfer.

```
function promiseMortgageCompletion(uint amount,
         uint _annual_percentage_rate,
         uint _final_amount, uint _amount_of_payment)
         isRequested onlyExecutor
{
  promise();
  mortgage.amount = amount;
  mortgage.annual_percentage_rate = _annual_percentage_rate;
  mortgage.final_amount = _final_amount;
  mortgage.amount_of_payment = _amount_of_payment;
  requestPropertyInsurance();
  requestPropertyOwnershipTransfer();
}
```

Here we have two functions that represent the Mortgage ownership transfer state and accept based on the Action Rule 4 (tab. 4.6). Here the performer of the request is not considered an authority and the conditions defined in claim to truth cannot be checked automatically, so both state and accept must be explicit. Once the accept is performed therefore the conditions from claim to truth have been verified outside smart contract, the state variables are set to desired values defined by the action rule.

```
function statePropertyOwnershipTransfer() {
  require (msg.sender == client );
  stateSub(propertyOwnershipTransfer);
}
```

```
function acceptPropertyOwnershipTransfer() onlyExecutor {
  require( propertyOwnershipTransfer.current_c_fact ==
           C_facts.Stated );
  property.owner = client;
  property.lien = this;
  acceptSub(propertyOwnershipTransfer);
}
```

On the other hand the property releaser is considered an authority so when stating the Property lien release, based on Action Rule 12 (tab. 4.14), we can perform accept `acceptSub(propertyLeinRelease)` and `state` tacitly.

```
function statePropertyLeinRelease(string _property_id)  {
  require( propertyLeinRelease.current_c_fact ==
           C_facts.Requested );
  require (msg.sender == property_releaser );
  require( keccak256(_property_id) == keccak256(property.id));
  property.lien = 0x0;
  acceptSub(propertyLeinRelease);
  state();
}
```

As stated before the transaction Mortgage Paying off is represented by another smart contract, which is created from within the `MortgageContract`.

```
function requestMortgagePaingOff() onlyExecutor returns (address) {
  require( propertyInsurance.current_c_fact == C_facts.Accepted);
  require( propertyOwnershipTransfer.current_c_fact ==
           C_facts.Accepted );
  mortgagePaingOff = new MortgagePaingOff(this, mortgage.final_amount,
                     mortgage.amount_of_payment);
  return address(mortgagePaingOff);
}
```

4.4.3 Mortgage Paying Off as Smart Contract

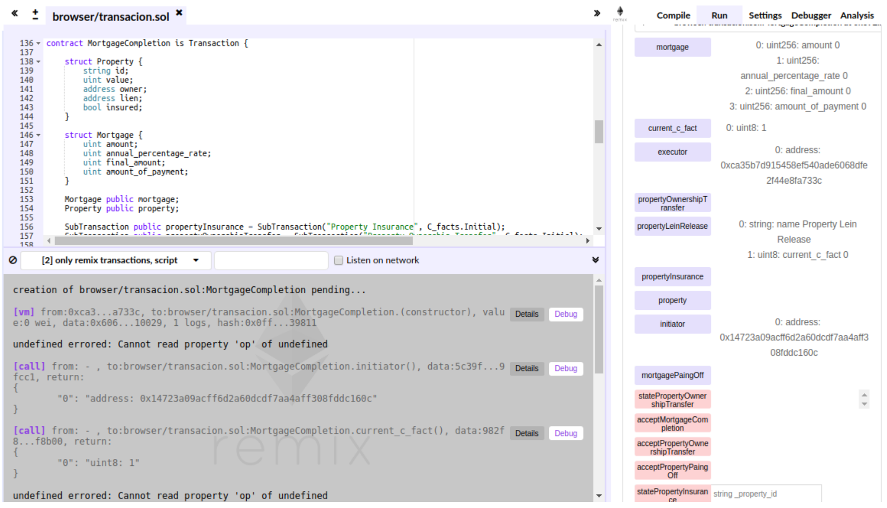
Mortgage paying off is another transaction that can be represented by smart contract. The difference is that here it is not just the transaction but also the actor A-3. This is given by the trustless payments in ethers on Ethereum blockchain and its usage in smart contract, all the executional logic applied by actor A-3 is then fully replaceable by smart contract. The DEMO model itself would not change, only the actor role of A-3 is assigned to smart contract. The initiator of this transaction is than the Mortgage completion smart contract and executor is the contract itself.

An important function is the `stateMortgagePayment` based on the Action Rule 9 (tab. 4.11) and 10 (tab. 4.12). A withdrawal pattern, the recommended method by solidity to control funds transfer, was used here [36].

```
function stateMortgagePayment() payable {
  require( mortgagePayment.current_c_fact == C_facts.Requested );
  require( msg.value == amount_of_payment);
  acceptSub(mortgagePayment);
  amount_paid += msg.value;
  pendingWithdrawal += msg.value;

  if ( amount_paid == amount ) {
    state();
    mortgageCompletion.acceptPropertyPaingOff();
  } else {
   requestMortgagePayment();
  }
}
```

4.5 Simulation

In this section we are going to simulate the execution of the Mortgage Completion contract using Remix, to illustrate the behavior of blockchain and the contracts.



```
contract MortgageCompletion is Transaction {

    struct Property {
        string id;
        uint value;
        address owner;
        address lien;
        bool insured;
    }

    struct Mortgage {
        uint amount;
        uint annual_percentage_rate;
        uint final_amount;
        uint amount_of_payment;
    }

    Mortgage public mortgage;
    Property public property;

    SubTransaction public propertyInsurance = SubTransaction("Property Insurance", C_facts.Initial);
```

```
creation of browser/transacion.sol:MortgageCompletion pending...

[vm] from:0xca3...a733c, to:browser/transacion.sol:MortgageCompletion.(constructor), valu
e:0 wei, data:0x606...10029, 1 logs, hash:0x0ff...39811

undefined errored: Cannot read property 'op' of undefined

[call] from: - , to:browser/transacion.sol:MortgageCompletion.initiator(), data:5c39f...9
fcc1, return:
{
        "0": "address: 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c"
}

[call] from: - , to:browser/transacion.sol:MortgageCompletion.current_c_fact(), data:982f
8...f8b00, return:
{
        "0": "uint8: 1"
}

undefined errored: Cannot read property 'op' of undefined
```

mortgage: 0: uint256: amount 0; 1: uint256: annual_percentage_rate 0; 2: uint256: final_amount 0; 3: uint256: amount_of_payment 0

current_c_fact: 0: uint8: 1

executor: 0: address: 0xca35b7d915458ef540ade6068dfe2f44e8fa733c

propertyLeinRelease: 0: string: name Property Lein Release; 1: uint8: current_c_fact 0

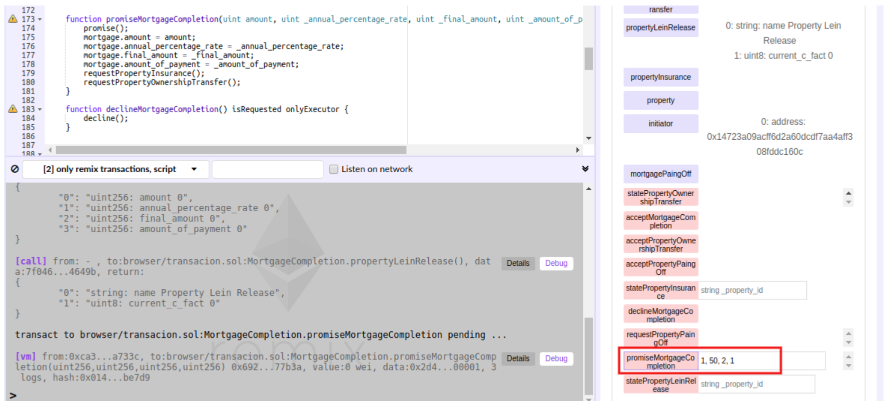
initiator: 0: address: 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c

statePropertyInsurance: string _property_id

Figure 4.7: Deployment of the contract

The first step is to deploy the contract to our virtual blockchain. We choose the address to deploy from, we set the gas limit and than create the contract

MortgageCompletion. As we can see in fig 4.7 the contract was deployed, its public state variables and methods are listed on the right side. The message “undefined errored: Cannot read property ’op’ of undefined” is a Remix bug, the contract deployment was successful. This step is taken once the Mortgage Completion has been requested. The contract is deployed by the loaner.



```
function promiseMortgageCompletion(uint amount, uint _annual_percentage_rate, uint _final_amount, uint _amount_of_p
    promise();
    mortgage.amount = amount;
    mortgage.annual_percentage_rate = _annual_percentage_rate;
    mortgage.final_amount = _final_amount;
    mortgage.amount_of_payment = _amount_of_payment;
    requestPropertyInsurance();
    requestPropertyOwnershipTransfer();
}

function declineMortgageCompletion() isRequested onlyExecutor {
    decline();
}
```

```
{
        "0": "uint256: amount 0",
        "1": "uint256: annual_percentage_rate 0",
        "2": "uint256: final_amount 0",
        "3": "uint256: amount_of_payment 0"
}

[call] from: - , to:browser/transacion.sol:MortgageCompletion.propertyLeinRelease(), dat
a:7f046...4649b, return:
{
        "0": "string: name Property Lein Release",
        "1": "uint8: current_c_fact 0"
}

transact to browser/transacion.sol:MortgageCompletion.promiseMortgageCompletion pending ...

[vm] from:0xca3...a733c, to:browser/transacion.sol:MortgageCompletion.promiseMortgageComp
letion(uint256,uint256,uint256,uint256) 0x692...77b3a, value:0 wei, data:0x2d4...00001, 3
 logs, hash:0x014...be7d9
>
```

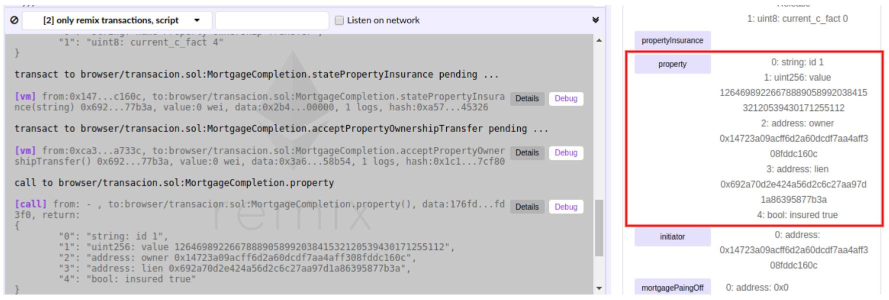
propertyLeinRelease: 0: string: name Property Lein Release 1: uint8: current_c_fact 0

initiator: 0: address: 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c

promiseMortgageCompletion: 1, 50, 2, 1

Figure 4.8: Promising the Mortgage Completion

When the request has been internally evaluated by the loaner and the conditions were finalized, the Mortgage Completion is either declined or promised. In the case of promise, data about the mortgage are sent in a blockchain transaction. For illustration we define the mortgage amount to 1 wei, with the annual percentage rate 50% and resulting on final amount to pay of 2 wei fig. 4.8. Tacit request of Property Insurance and Property Ownership Transfer is executed.



```
        "1": "uint8: current_c_fact 4"
}

transact to browser/transacion.sol:MortgageCompletion.statePropertyInsurance pending ...

[vm] from:0x147...c160c, to:browser/transacion.sol:MortgageCompletion.statePropertyInsura
nce(string) 0x692...77b3a, value:0 wei, data:0x2b4...00000, 1 logs, hash:0xa57...45326

transact to browser/transacion.sol:MortgageCompletion.acceptPropertyOwnershipTransfer pending ...

[vm] from:0xca3...a733c, to:browser/transacion.sol:MortgageCompletion.acceptPropertyOwner
shipTransfer() 0x692...77b3a, value:0 wei, data:0x3a6...58b54, 1 logs, hash:0x1c1...7cf80

call to browser/transacion.sol:MortgageCompletion.property

[call] from: - , to:browser/transacion.sol:MortgageCompletion.property(), data:176fd...fd
3f0, return:
{
        "0": "string: id 1",
        "1": "uint256: value 1264698922667888905899203841532120539430171255112",
        "2": "address: owner 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c",
        "3": "address: lien 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a",
        "4": "bool: insured true"
}
```

1: uint8: current_c_fact 0

property: 0: string: id 1 1: uint256: value 1264698922667888905899203841532120539430171255112 2: address: owner 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c 3: address: lien 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a 4: bool: insured true

initiator: 0: address: 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c

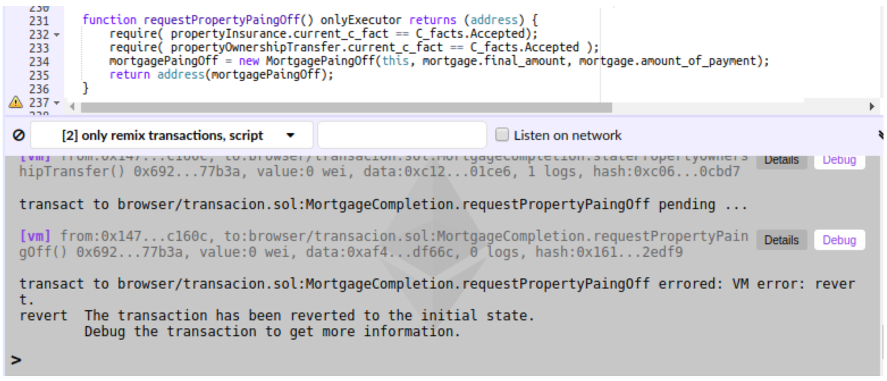
mortgagePaingOff: 0: address: 0x0

Figure 4.9: Change of state of Property

In the next step, the Property Insurance and Property Ownership Transfer are executed. To state these transactions successfully, the blockchain transaction must be sent from the address assigned to insurer and property transferer. As the insurer is considered an authority in our scenario, the accept of the Property Insurance is tacit. On the contrary, accept of the Property Ownership Transfer must be performed explicitly. When accepted the state variable

Property is updated and the address of the client is added as owner and address of the Mortgage Contract as the lien address fig. 4.9.

When both sub-transactions are accepted, the loaner can request the Mortgage Paying Off. If the loaner would try to request it sooner the blockchain transaction would not be successful and contract would be reverted fig. 4.10. In this step the Mortgage Paying off contract is created and deployed. We can get the contract‘s address from the main contract state variable and load it using the address. The next steps are then performed by the Mortgage Paying Off contract.



```
function requestPropertyPaingOff() onlyExecutor returns (address) {
    require( propertyInsurance.current_c_fact == C_facts.Accepted);
    require( propertyOwnershipTransfer.current_c_fact == C_facts.Accepted );
    mortgagePaingOff = new MortgagePaingOff(this, mortgage.final_amount, mortgage.amount_of_payment);
    return address(mortgagePaingOff);
}
```

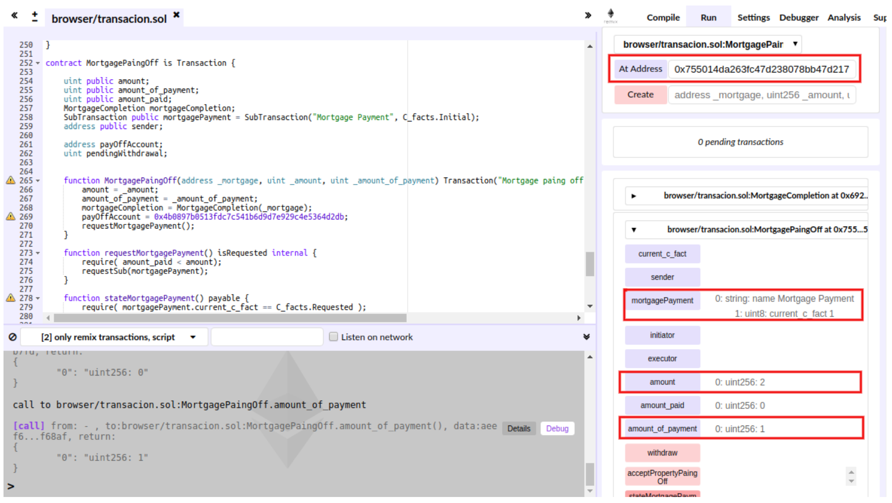
```
hipTransfer() 0x692...77b3a, value:0 wei, data:0xc12...01ce6, 1 logs, hash:0xc06...0cbd7

transact to browser/transacion.sol:MortgageCompletion.requestPropertyPaingOff pending ...

[vm] from:0x147...c160c, to:browser/transacion.sol:MortgageCompletion.requestPropertyPain
gOff() 0x692...77b3a, value:0 wei, data:0xaf4...df66c, 0 logs, hash:0x161...2edf9

transact to browser/transacion.sol:MortgageCompletion.requestPropertyPaingOff errored: VM error: rever
t.
revert  The transaction has been reverted to the initial state.
        Debug the transaction to get more information.

>
```

Figure 4.10: Reverting transaction

The Mortgage Paying Off contract covers the corresponding transaction and also acts like the actor A-3, because payments on blockchain can be handled automatically. When the Mortgage Paying Off is created, the final amount and amount of one payment are initialized from the Mortgage Completion contract, we define the address of the Loaner to pay the amount to and, finally, its sub-transaction Mortgage Payment is requested fig. 4.11

The mortgage payment can be sent from any address. In our scenario two payments with the amount of 1 wei must be paid. In order to accept the payment, the Mortgage payment must be requested and the sent value must be equal to the amount of one payment, otherwise the blockchain transaction is reverted and ethers are not transfered. To pay the payment we send a blockchain transaction with the desired value to the contract fig. 4.12.

Once the full amount has been paid, the Mortgage Paying Off is stated and automatically calls Mortgage Completion contract to accepted it. The Mortgage Lien Release is then requested tacitly. The Lien Releaser is considered an authority, therefore when Mortgage Lien Release is stated the accept is performed automatically, as well as the Mortgage Completion stating. As we can see in fig. 4.13 from the console detail of the blockchain two events were logged. One for accepting (5) of the Property Lien Release and one for stating (4) the Mortgage Completion. On the right side we can see that the state variable Property has been updated and the lien address was set to empty



```
}

contract MortgagePaingOff is Transaction {

    uint public amount;
    uint public amount_of_payment;
    uint public amount_paid;
    MortgageCompletion mortgageCompletion;
    SubTransaction public mortgagePayment = SubTransaction("Mortgage Payment", C_facts.Initial);
    address public sender;

    address payOffAccount;
    uint pendingWithdrawal;


    function MortgagePaingOff(address _mortgage, uint _amount, uint _amount_of_payment) Transaction("Mortgage paing off
        amount = _amount;
        amount_of_payment = _amount_of_payment;
        mortgageCompletion = MortgageCompletion(_mortgage);
        payOffAccount = 0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db;
        requestMortgagePayment();
    }

    function requestMortgagePayment() isRequested internal {
        require( amount_paid < amount);
        requestSub(mortgagePayment);
    }

    function stateMortgagePayment() payable {
        require( mortgagePayment.current_c_fact == C_facts.Requested );
```

```
{
        "0": "uint256: 0"
}

call to browser/transacion.sol:MortgagePaingOff.amount_of_payment

[call] from: - , to:browser/transacion.sol:MortgagePaingOff.amount_of_payment(), data:aee
f6...f68af, return:
{
        "0": "uint256: 1"
}
>
```

browser/transacion.sol:MortgagePair
At Address 0x755014da263fc47d238078bb47d217

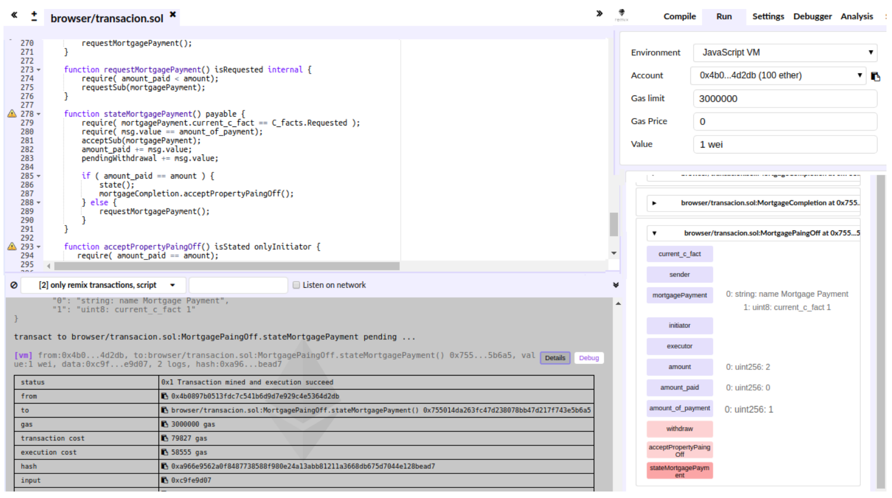
0 pending transactions

browser/transacion.sol:MortgageCompletion at 0x692..

browser/transacion.sol:MortgagePaingOff at 0x755...5

mortgagePayment	0: string: name Mortgage Payment 1: uint8: current_c_fact 1
amount	0: uint256: 2
amount_paid	0: uint256: 0
amount_of_payment	0: uint256: 1

Figure 4.11: Mortgage Paying Off initialization



```
        requestMortgagePayment();
    }

    function requestMortgagePayment() isRequested internal {
        require( amount_paid < amount);
        requestSub(mortgagePayment);
    }

    function stateMortgagePayment() payable {
        require( mortgagePayment.current_c_fact == C_facts.Requested );
        require( msg.value == amount_of_payment);
        acceptSub(mortgagePayment);
        amount_paid += msg.value;
        pendingWithdrawal += msg.value;

        if ( amount_paid == amount ) {
            state();
            mortgageCompletion.acceptPropertyPaingOff();
        } else {
            requestMortgagePayment();
        }
    }

    function acceptPropertyPaingOff() isStated onlyInitiator {
       require( amount_paid == amount);
```

```
        "0": "string: name Mortgage Payment",
        "1": "uint8: current_c_fact 1"
}

transact to browser/transacion.sol:MortgagePaingOff.stateMortgagePayment pending ...

[vm] from:0x4b0...4d2db, to:browser/transacion.sol:MortgagePaingOff.stateMortgagePayment() 0x755...5b6a5, val
ue:1 wei, data:0xc9f...e9d07, 2 logs, hash:0xa96...bead7
```

status	0x1 Transaction mined and execution succeed
from	0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db
to	browser/transacion.sol:MortgagePaingOff.stateMortgagePayment() 0x755014da263fc47d238078bb47d217f743e5b6a5
gas	3000000 gas
transaction cost	79827 gas
execution cost	58555 gas
hash	0xa966e9562a0f8487738588f980e24a13abb81211a3668db675d7044e128bead7
input	0xc9fe9d07

Environment	JavaScript VM
Account	0x4b0...4d2db (100 ether)
Gas limit	3000000
Gas Price	0
Value	1 wei

browser/transacion.sol:MortgageCompletion at 0x755..

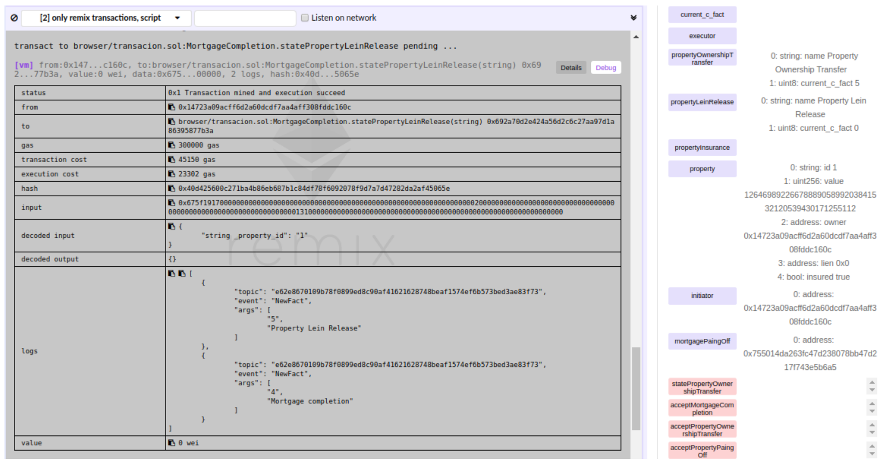
browser/transacion.sol:MortgagePaingOff at 0x755...5

mortgagePayment	0: string: name Mortgage Payment 1: uint8: current_c_fact 1
amount	0: uint256: 2
amount_paid	0: uint256: 0
amount_of_payment	0: uint256: 1

Figure 4.12: Mortgage Payment blockchain transaction

address.

Now the final step is for the client to accept the Mortgage Completion and the transaction is finished.



```
transact to browser/transacion.sol:MortgageCompletion.statePropertyLeinRelease pending ...

[vm] from:0x147...c160c, to:browser/transacion.sol:MortgageCompletion.statePropertyLeinRelease(string) 0x69
2...77b3a, value:0 wei, data:0x675...00000, 2 logs, hash:0x40d...5065e
```

status	0x1 Transaction mined and execution succeed
from	0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
to	browser/transacion.sol:MortgageCompletion.statePropertyLeinRelease(string) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
gas	300000 gas
transaction cost	45150 gas
execution cost	23302 gas
hash	0x40d425600c271ba4b86eb687b1c84df78f6092078f9d7a7d47282da2af45065e
input	0x675f191700000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000001310000000000000000000000000000000000000000000000000000000000000000
decoded input	{ "string _property_id": "1" }
decoded output	{}
logs	see below
value	0 wei

```
[
	{
		"topic": "e62e8670109b78f0899ed8c90af41621628748beaf1574ef6b573bed3ae83f73",
		"event": "NewFact",
		"args": [
			"5",
			"Property Lein Release"
		]
	},
	{
		"topic": "e62e8670109b78f0899ed8c90af41621628748beaf1574ef6b573bed3ae83f73",
		"event": "NewFact",
		"args": [
			"4",
			"Mortgage completion"
		]
	}
]
```

```
current_c_fact
executor
propertyOwnershipTransfer    0: string: name Property Ownership Transfer
                             1: uint8: current_c_fact 5
propertyLeinRelease          0: string: name Property Lein Release
                             1: uint8: current_c_fact 0
propertyInsurance
property                     0: string: id 1
                             1: uint256: value 1264698922667888905899203841532120539430171255112
                             2: address: owner 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
                             3: address: lien 0x0
                             4: bool: insured true
initiator                    0: address: 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
mortgagePaingOff             0: address: 0x755014da263fc47d238078bb47d217f743e5b6a5
statePropertyOwnershipTransfer
acceptMortgageCompletion
acceptPropertyOwnershipTransfer
acceptPropertyPaingOff
```

Figure 4.13: State of Lien Release

4.5.1 Summary

We have used the DEMO methodology to understand and reveal the essence of a mortgage process. Using the proposed principles of creating contract introduced in chapter 3 we were able to create working smart contracts based on the DEMO models.

We created a trustless notarization of the process. The contract holds immutably the agreed mortgage conditions such as amount of payment and interest rate. Further it controls execution of some parts, such as automatic mortgage payment control and automatic lien release request. This way the Client can be sure that once the mortgage has been paid off the lien will be released. It also defines a single point of access to the data and coordination for all parties as well as simplifies some steps as automatic control can be performed, allowing us to carry out some acts tacitly. Using smart contract would not change the DEMO model or the essence of the process it could help to simplify the current implementation. For example the Client would not have to bring the confirmation about insurance to the Loaner, because this is done by the smart contract, this reduces the overall process steps behind mortgage and eliminates possible bureaucracy.