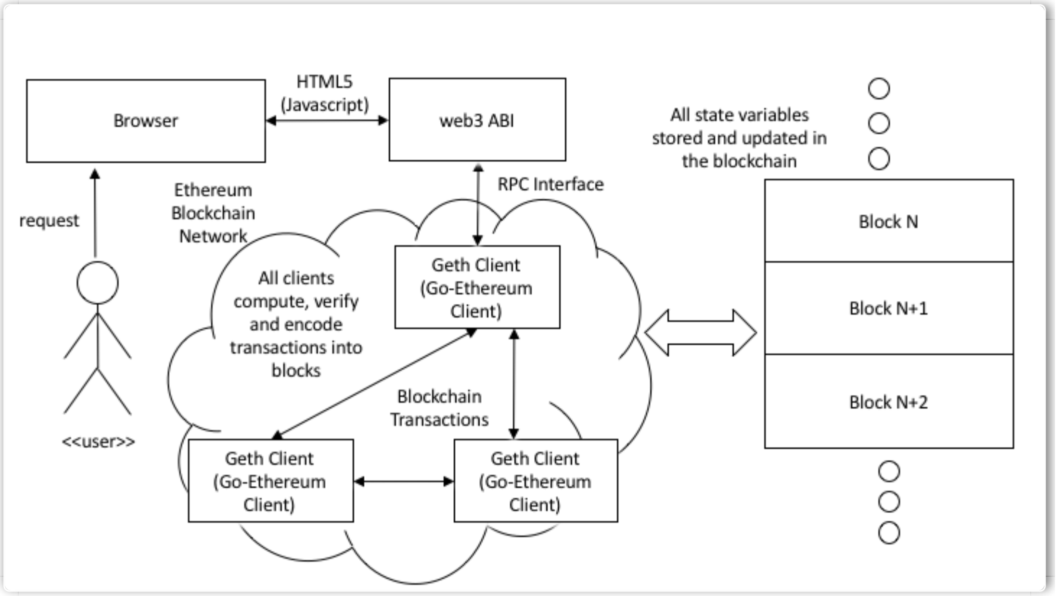
Blockchain Course Wiki

Log in to access more services

Web3

Your apps need some way of interacting with the blockchain. Here is a common architecture:


Browser
HTML5 (Javascript)
web3 ABI
All state variables stored and updated in the blockchain
request
Ethereum Blockchain Network
RPC Interface
Block N
Block N+1
Block N+2
All clients compute, verify and encode transactions into blocks
Geth Client (Go-Ethereum Client)
Blockchain Transactions
<<user>>
Geth Client (Go-Ethereum Client)
Geth Client (Go-Ethereum Client)

An Example Interface:

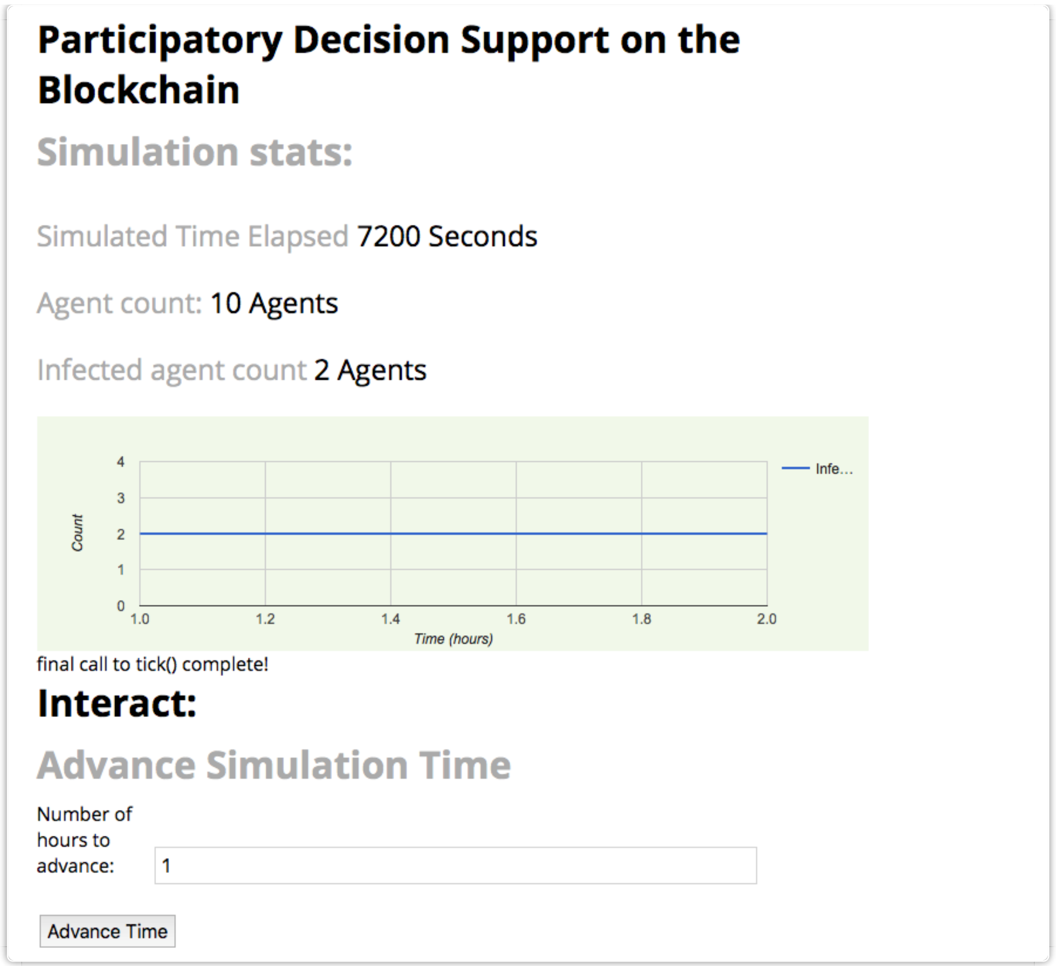
DOCUMENT INFO

TAGS

RELATED

COMMENTS

HISTORY



Participatory Decision Support on the Blockchain

Simulation stats:

Simulated Time Elapsed 7200 Seconds

Agent count: 10 Agents

Infected agent count 2 Agents

final call to tick() complete!

Interact:

Advance Simulation Time

Number of hours to advance: 1

Advance Time

What

So far, when interacting with Geth, we have done so from its console. However, when creating DApps, this is not enough; we need to create proper interfaces and remotely interact with Geth, or any other node.

This is where Web3 comes in. It is a Javascript library that simplifies RPC communication with an Ethereum node. In the context of it, the node is called a *provider*. It exposes a large number of methods, as long as they have been enabled over RPC by Geth.

Important groups of methods are related to:

- the underlying blockchain infrastructure. Examples are:
  - web3.eth.blockNumber
  - web3.eth.gasPrice
  - web3.eth.getBalance(...)
- transactions and contracts. Examples are:
  - web3.eth.sendTransaction(...)
  - web3.eth.contract(...)
  - web3.eth.getCode(...)
- formatting and calculations. Examples are:
  - web3.fromWei(...)

web3.eth is the same eth we had in Geth.

For more info on web3 interface (gory details) see:

https://github.com/ethereum/wiki/wiki/JavaScript-API

Useful code snippets that you can use similar to the getTransactions.js example demonstrated:

https://ethereum.stackexchange.com/questions/2531/common-useful-javascript-snippets-for-geth/3478#3478

interacting with contracts:

https://ethereum.stackexchange.com/questions/8736/how-to-call-my-contracts-function-using-sendtransaction

Exercises

Run these in geth ... console for the sake of simplicity. It has an auto-complete facility:

- press TAB key once, it completes what it can
- press TAB key twice, you get the list of auto-complete options

Look around

List your accounts

```
> web3.eth.accounts
["0xd2e2f145b706f4da24b38d048094497c90a7b35c"]
```

Get your balance

```
> web3.eth.getBalance("0xd2e2f145b706f4da24b38d048094497c90a7b35c")
3800000000000000
```

Convert values

Numerals

Value amounts are always passed as wei, but when it comes to showing the user, it is better to use other units, such as ether or finney.

```
> web3.toWei(1, "ether")
"1000000000000000000" // String
> web3.toWei(web3.toBigNumber(1), "ether")
1000000000000000000 // BigNumber
> web3.toWei(web3.toBigNumber(1), "ether").toString(16)
"de0b6b3a7640000" // BigNumber converted to hex
> web3.toWei(web3.toBigNumber(1), "ether").toString(10)
"1000000000000000000" // BigNumber converted to decimal
> web3.fromWei("1000000000000000000", "finney")
"1000" // String
> web3.fromWei(web3.toBigNumber("1000000000000000000"), "finney")
1000 // BigNumber
> web3.toBigNumber("0x400")
1024
```

The number returned is a String or a BigNumber that lets you make calculations on extremely large numbers. In Ethereum, wei values are always very large.

```
> web3.toWei(web3.toBigNumber(1), "ether").plus(web3.toWei(web3.toBigNumber(1), "finney"))
1001000000000000000 // BigNumber
```

Strings

When writing contracts, you will come across bytes8 to bytes32 and bytes. When these types are returned to Web3 from a contract method, and you know they represent a string, you may use:

```
> web3.fromAscii("Hello World")
"0x48656c6c6f20576f726c64"
> web3.toAscii("0x48656c6c6f20576f726c640000000000000000000000000000000000000000000") // Typical
bytes32
"Hello Worldx00x00x00x00x00x00x00x00x00x00x00x00x00x00x00x00x00x00x00x00x00"
> web3.toUtf8("0x48656c6c6f20576f726c640000000000000000000000000000000000000000000")
"Hello World"
> web3.fromUtf8("élan")
"0xc3a96c616e"
```

Use it in the browser

Let's now look at how we use Web3 on the browser. We will reuse the deployed code and addresses of the *Introduction to Ethereum contracts*. Here is a summary of the actions we will have to perform:

- Instruct Geth to listen to RPC calls
- Create folders for the project
- Download required Web3 files
- Create contract files
- Prepare compiled contract in JS
- Create an HTML file of the rudimentary application

Prepare Geth

So far, we have launched Geth from the command-line, and interacted with it from within its own console. If we are now to interact with Geth from a browser, we need Geth to answer to calls coming from it. This takes place over an RPC, so to instruct Geth to open up add these parameters:

```
$ geth --rpc --rpcport 8545 --rpcaddr 0.0.0.0 --rpccorsdomain "*" --rpcapi "eth,web3" console
```

- 8545 is the default port.
- 0.0.0.0 tells it to serve all IP addresses, which is convenient when you run it from a virtual machine.
- When you have a public Geth and want to limit requests to it, this is where you may narrow the --rpccorsdomain "*" part.

Verification

If you did not get output similar to the lines below then did not correctly copy the RPC configuration.

```
# eth.coinbase
$ curl -X POST --data '{"jsonrpc":"2.0","method":"eth_coinbase","params":[],"id":1}' localhost:8
545
{"jsonrpc":"2.0","id":1,"result":"0x6c503786685f23abae76976fe0aa843f75ea9e71"}
# eth.accounts
$ curl -X POST --data '{"jsonrpc":"2.0","method":"eth_accounts","params":[],"id":2}' localhost:8
545
{"jsonrpc":"2.0","id":2,"result":["0x6c503786685f23abae76976fe0aa843f75ea9e71","0xaa2d08a053b8f0
07cc685e72975e31bcd336a028"]}
# Failed personal.listAccounts
$ curl -X POST --data '{"jsonrpc":"2.0","method":"personal_listAccounts","params":[],"id":3}' lo
calhost:8545
{"jsonrpc":"2.0","id":3,"error":{"code":-32601,"message":"The method personal_listAccounts does
not exist/is not available"}}
```

Let's temporarily restart Geth with --rpcapi "eth,web3,personal", and let's try the previously failed personal_listAccounts:

```
# personal.listAccounts
$ curl -X POST --data '{"jsonrpc":"2.0","method":"personal_listAccounts","params":[],"id":4}' lo
calhost:8545
{"jsonrpc":"2.0","id":4,"result":["0x6c503786685f23abae76976fe0aa843f75ea9e71","0xaa2d08a053b8f0
07cc685e72975e31bcd336a028"]}
```

We just confirmed that for personal.listAccounts to be exposed, Geth needs to enable personal over RPC.

You can use something similar to the following to run geth in such a way that it will accept RPC requests:

```
./geth --datadir "/Users/Marek/Documents/blockchain_course/" --networkid 20171104 --rpc --rpcpor
t 8545 --rpcaddr 0.0.0.0 --rpccorsdomain "*" --rpcapi "eth,web3,personal" console
```

Then you can display a crude browser interface using the following:

and the following in the same directory:

index.html

index.js

web3.js

This simple web page uses web3 to communicate with a running ethereum client (in this case geth) listening for commands on http port 8545. If working correctly it should display your coinable address. Make sure you look at your browser's logging console for details or errors.