SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Antonio Martinović

PRIMJENA HYPERLEDGER PLATFORME

DIPLOMSKI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Martinović

Matični broj: 45299/16–R

Studij: Informacijsko i programsko inženjerstvo

PRIMJENA HYPERLEDGER PLATFORME

DIPLOMSKI RAD

Mentor :

Izv. prof. dr. sc. Sandro Gerić

Varaždin, Rujan 2019.

Antonio Martinović

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom radu opisati ću *Hyperledger* ekosustav *blockchain* programskih okruženja i popratnih alata pod okriljem *Linux Foundation* organizacije *Hyperledger*. Nakon teoretskog opisa *blockchain* mreže i prednosti, te nedostataka istih, preći ću na opširni opis čitavog Hyperledger ekosustava, zatim ću se fokusirati na detaljni tehnički opis *Hyperledger Fabric*-a kao nejvećeg i najrazvijenijeg programskog okruženja. Te ću rad zaključiti sa opisom praktičnog rješenja u *Hyperledger Fabric* okolini. Cilj mi je na taj način čitatelja dobro upoznati sa mogućnostima, te prednostima i nedostacima dotične platforme, te kakvu ulogu ispunjava u kompletnom današnjem svijetu.

Ključne riječi: Permissioned Blockchain Hyperledger Distributed Ledger

Sadržaj

1. Uvod

U zadnje vrijeme *blockhain* je postao ključna riječ ne rijetko korištena na krivi način i u krivom kontekstu ili čak izjednačena sa izrazom "kripto valuta" što govori o tome koliko opća javnost, ali i profesionalni svijet, loše upoznat sa ovom novom tehnologijom. Zbog toga se sama nameće sličnost između trenutnog stanja *blockchain* tehnologije i stanja Interneta u svojim mladim danima kakav je bio krajem prošlog stoljeća, posebice vidljivim u svijetu kripto valuta ako se prisjetimo *dot-com* uspona i pada. Danas je Internet sveprisutan, kao civilizacija upoznali smo se sa prednostima i nedostacima opće povezanosti, pa se postavlja pitanje je li i *blockchain* tehnologija sadrži jednaki potencijal da promijeni način na koje naše društvo funkcionira.

Prednosti *blockchain* tehnologije svakako postoje, neke od njih ću pokazati u ovom radu, te ću pokušati na najbolji način predočiti potencijal koji se nalazi u ovoj tehnologiji, kao i pokazati na koji način se može ispravno iskoristiti.

1.0.1. Kada je blockchain dobro riješenje

Preveliki broj tehnoloških entuzijasta ili profesionalaca i poslovnih igrača priča o tome kako implementirati *blockchain* riješenje u svoje poslovanje, ali pri tome nužno ne pričaju o tome da li im je takvo riješenje uopće potrebno u njihovim slučajevima korištenja.

Ako bi htjeli postaviti neka osnovna pitanja prema kojima bi mogli odrediti da li nam uopće treba *blockchain*, onda bi oni mogli zvučati ovako[11]:

- Je li se rad mreže zasniva na međusobnoj interakciji između njenih korisnika koji pripadaju više od jednoj organizaciji?
- Postoji li problem povjerenja u mreži, odnosno je li individualni korisnici u mreži imaju suprotne ciljeve ili imaju razloga za lažno djelovanje unutar mreže?
- Zahtjeva li sustav dugoročno spremanje podataka o kojima ovise buduće radnje na sustavu?

Ukoliko je odgovor na sva 3 prethodna pitanja afirmativan onda možemo razmatrati mogućnost implementacije sustava koja koristi *blockchain*. No preostaje još pitanje kakav oblik *blockchain* tehnologije trebamo koristiti. *Blockchain* se primarno dijeli na dvije različite vrste: bez dopuštenja i sa dopuštenjima.

Jedan dobar primjer korištenja *blockchain* mreže je lanac opskrbe dobara. U takvom sustavu proizvođači i potrošači mogli bi dijeliti jedno spremište podataka na kojem je u svakom trenutku vidljivo trenutno stanje što znači da se taj lanac opskrbe može pretvoriti u lanac potražnje gdje proizvođači mogu temeljiti svoje poslovne odluke o proizvodnji prema stvarnim potrebama potrošača u dotičnoj mreži. Proizvođači u ovom slučaju mogu dobiti daleko precizniju sliku o potrebama tržišta, te se ne moraju oslanjati na predviđanja budućih kretnji tržište

temeljem prijašnjih. Potrošači bi također imali koristˌi od ovakve tehnologije na način da uvijek mogu provjeriti izvor dobara koje konzumiraju, te je li dobro proizvedeno na ispravan, etički i/ili siguran način, te dobivaju trenutni uvid u put koje je dobro prošlo do njih, što uključuje kad, gdje i kakve provjere su provedene nad dobrom. Primjer ovakvog proizvoda su *TE-FOOD* (te-food.com) i *Circulor* (https://www.circulor.com/).

Jedan od zanimljivijih primjera prijedloga uporabe je i globalna kontrola zračnog prostora koja bi integrirala civilne, istraživačke i vojne potrebe u jednom sustavu koji je nedavno predstavila NASA[19].

Problem koji se predstavlja ovdje je da uvijek postoji mogućnost da se u digitalni sustav unese krivi podatak o stvarnom svijetu, bilo greškom ili namjerno sa ciljem da se iznudi ilegalni dobitak, međutim ovo je problem koji postoji u svakom sustavu sa ovisnošću o ljudskom faktoru. Riješenje tog problema može se naći u internetu stvari, digitalnim senzorima koji bi mogli, primjerice, samostalno brojati proizvode koji su dostavljeni u skladištu, što se može koristiti uz trenutne metode koje uključuju redovne inspekcije dobavljača i proizvođača od strane regulativnih tijela.

2. O *Blockchain* tehnologiji

Blockchain je tehnologija koja se poprilično popularizirala u zadnjih nekoliko godina zahvaljujući rastu kripto valuta poput Bitcoin-a, Ether-a i naizgled beskonačno mnogo drugih kripto valuta različitih primjena. Primjeri raznoraznih načina korištenja kripto valuta su IOTA i Streamr koji služe za trgovanje podacima između računala, Walton Chain i VeChain koji žele digitalizirati praćenje stvarnih objekata, Oyster i Sia za spremište podataka i razni ostali specijalizirani lanci.

> Kripto valuta je bilo koji isključivo digitalni oblik valute koje uobičajeno nema centralnog izdavača ili regulativno tijelo, već koristi decentralizirani sustav za spremanje transakcija i upravljanje izdavanjem novih jedinica, te korištenjem kriptografije sprječava krivotvorenje i lažne transakcije[4].

Međutim kripto valuta je samo jedna od mogućih primjena *blockchain* tehnologije, koju jednostavan, ali i neprecizan način, možemo opisati kao baza podataka u koju možemo samo upisivati nove podatke koji utječu na njeno stanje. Dakle svaka podatkovna operacija poput stvaranja novog entiteta, izmjene ili brisanja postojećeg entiteta ostvaruje se novom transakcijom, a trenutno stanje je suma svih dotadašnjih transakcija. Opcionalno, operacije čitanja mogu biti izvedene u obliku transakcije, ukoliko je potrebno voditi evidenciju i o pristupu podataka. Velika prednost ovog načina rada je mogućnost pregleda povijesti promijene podataka, ali cijena je i puno veća količina prostora potrebnog za spremanje podataka.

No format zapisa nije jedini atribut *blockchain*-a koji ga dijeli od konvencionalnih metoda. Svaki ovakav sustav je realiziran kao mreža ravnopravnih članova (eng. *peer-to-peer*), te kao takav, temelji se na višestrukoj provjeri različitih članova mreže pomoću kojeg se donosi odluka o ispravnosti novih podataka, te da li je transakcija ispravna. Pametni ugovori su unaprijed dogovoreni programi koje korisnici koriste za izradu transakcija, odnosno, to je transakcijski protokol koji provjerava i potvrđuje uvijete transakcije, te, ukoliko su uvjeti zadovoljeni, rezultat se označava kao ispravan te se dodaje na trenutni niz transakcija. Ovisno o implementaciji, nevažeće transakcije se isto mogu zapisivati u nizu uz naznaku da su neispravne, za potrebe evidencije. Da bi se ispravne transakcije mogle dodati na *blockchain*, potrebno ih je zapisati u novi blok, koji se eventualno pripaja postojećem lancu. Za dodavanje novih blokova u lanac potreban je konsenzus između svih pripadnika u *blockchain* mreži o tome koji blok se smatra ispravnim. Jedan od poznatih konsenzus protokola je Dokaz o Radu (dalje u tekstu eng. *Proof of Work* (PoW)) kojeg je popularizirao Satoshi Nakamoto[17] koristeći ga u najpoznatijoj implementaciji *blockchain*-a, *Bitcoin*. PoW se temelji na izračunu sažetka svih transakcija u bloku, računska operacija koja zahtjeva konstantno rastuću količinu računalnih resursa, te se oslanja na činjenicu da se potencijalnim napadačima jednostavno više isplati koristiti svoje procesorske cikluse na fer način, nego pokušavajući lažirati sljedeće blokove i samim time transakcije od kojih su sačinjeni u svrhu vlastitog dobitka.

Konsenzus algoritmi i pametni ugovori zajedno za sobom povlače i uklanjanje potrebe za trećom stranom koja posreduje u izvođenju transakcija i pružanja vjerodostojnog izvora istine o trenutnom stanju. Primjeri treće strane u slučaju valute su centralne banke, ili banke u slučaju

financijskih prijenosa, ili država, u slučaju sudskih sporova ili javnih registara i slično. Kao što je i kod bilo kojeg drugog sustava baze podataka, postoje različite uloge u sustavu, npr. oni koji imaju prava za čitanje i pisanje ili oni koji imaju prava samo za čitanje. U slučaju *blockchain*-a, oni sa pravima pisanja mogu dodavati nove blokove u *blockchiain*, a oni sa pravima čitanja bi samo mogli čitati podatke iz dotičnih blokova. Navedeno je temelj za jednu od bitnijih razlika u *blockchain* sustavima danas, oni koji podržavaju dopuštenja i oni koji ih ne podržavaju, kao što je navedeno prije. *Bitcoin* i *Ethereum* su vjerojatno najpoznatiji predstavnici *blockchain*-a bez dopuštenja. Ovo znači da bilo tko može pisati i čitati podatke, te ne postoji nekakvo tijelo koje kontrolira prava pristupa podacima, pa samim time ne postoji nikakva eksplicitna zaštita od ilegalnog ponašanja raznih članova. Naravno, pametnih ugovora moguće je uvesti takve kontrole na aplikacijskoj razini. Ovakva otvorenost funkcionira u nekim slučajevima korištenja, međutim u nekim slučajevima ipak je potrebna ugrađena kontrola pristupa i uloga u sustavu kakvu pruža druga vrsta *blockchaina.*

U *blockchain* sustavima sa dopuštenjima postoji jedno ili više tijela koji izdaju identifikacije i prava članovima na mreži, odnosno, reguliraju tko ima pravo čitanja podataka, granulaciju pristupačnosti podacima i tko može te podatke izmijeniti. Najpoznatiji primjer ovog principa rada je najvjerojatnije *Hyperledger Fabric* o kojemu će mo u detalje u poglavlju 6.

U usporedbi sa centraliziranim bazama podataka, *blockchain* nudi neke atribute koji ga odvajaju od konvencionalnih baza[11]:

- **Javna provjerljivost** - bilo tko, ili u *blockchain*-u sa dopuštenjima, bilo tko sa adekvatnim dopuštenjima, može provjeriti trenutno stanje, te se uvjeriti u ispravnost istog tako da provjeri prema prijašnjem stanju i transakcijama da li su ispoštovana sva pravila i protokol koji je doveo do trenutnog stanja. U centraliziranim bazama podataka je potrebno vjerovati tijelu koji njom upravlja da je promjene donijelo na ispravan način i da nije bilo lažnih izmjena stanja.

- **Transparentnost** - podaci mogu biti dostupni vanjskim korisnicima, no količina dostupnih podataka može se razlikovati prema razini prava koje korisnik posjeduje. Moguće je imati i potpuno privatni *blockchain* u kojem ovaj atribut nije toliko bitan, međutim implementacije poput *Hyperledger Fabrica* omogućuju jednostavno ubacivanje vanjskih korisnika koji mogu provoditi bilo kakve provjere autentičnosti koje zahtjeva zakon ili neka druga vlast.

- **Privatnost** - povjerljivi podaci mogu se osigurati korištenjem provjerenih metoda šifriranja, ili korištenjem većeg broja *blockchain* mreža koje međusobno dijele podatke. Dobar primjer za ovo vidjet će mo u poglavlju 5 Između transparentnosti i privatnosti postoji prirodni sukob, potpuno transparentni sustav ne pruža nikakvu privatnost i obrnuto, no postizanje adekvatne ravnoteže između ove dvije stavke je pitanje domene aplikacije.

- **Integritet** - podatke spremljene na *blockchainu* izuzetno je teško naknadno izmijeniti, bez usuglašavanja većine mreže. U centraliziranom sustavu ne postoji nikakav u potpunosti pouzdan način osiguranja od izmjene podataka od strance centralnog tijela koji ima apsolutni pristup.

- **Redundancija** - zbog samo principa rada *blockchain* sustava, prirodno svojstvo je redundancija pri čemu se sustav osigurava od gubitka podataka u slučaju nesreće ili isključivanja dijela mreže.
- **Povjerljivost** - tko ima najveći autoritet u sustavu da dodaje nove članove ili briše postojeće. U sustavima bez dopuštenja, takav član ne postoji, zbog nedostatka kontrole pristupa. U sustavima sa dopuštenjem to je problem koji ovisi o aplikaciji, moguće je imati jedno ili više tijela koji upravljaju pristupom i imaju mogućnost dodavanja ili izbacivanja članova sa mreže.

Blockchain sustav, dakle, predstavlja spremište podataka nad kojima su čvrsto definirana pravila interakcije, uvijek je moguće pregledati prijašnje transakcije, pratiti stanje sustava kroz vrijeme i samim time potvrditi autentičnost trenutnog stanja podataka, a korištenjem adekvatnog algoritma za konsenzus otpornog je na pokušaje falsificiranja podataka i krivotvorenja novih transakcija. Navedena svojstva predstavljaju potencijal da se korištenjem *blockchain* tehnologije uvelike poboljša efikasnost i transparentnost poslovanja i vladavine.

3. Hyperledger projekt

Hyperledger je započeo kao ideja da se napravi platforma koja će omogućiti razvoj novih distribuiranih knjižnih tehnologija (dalje u tekstu eng. *Distributed Ledger Technology* (DLT)) , tako da ponudi temeljnu podatkovnu strukturu, *blockchain*, i podršku za pametne ugovore u jednostavnom i proširivom paketu. Što znači da je *Hyperledger* zapravo skup različitih tehnologija, organizacija i ljudi kojima je zajednički cilj izgraditi i promovirati tehnologiju koja će služiti kao temelj daljnjeg razvoja. Unutra *Hyperledger* projekta uključene su mnoge poznate korporacije kao što su IBM, Intel, J.P. Morgan, Fujitsu, Hitachi, Cisco, Samsung i mnoge druge[8].

Ciljevi projekta su[12]:

- Izgraditi zajednicu tehnički i poslovno sposobnih ljudi koji će se fokusirati na razvoj i unaprjeđenje *Hyperledger* ekosustava
- Izgraditi i održavati alate, platforme i programe sa zajedničkim ciljem da olakša rad sa *Hyperledger* sustavima
- Okupiti programere, korporacije, organizacije i krajne korisnike, te im pomoći u razvoju i promociji svojeg softvera
- Ponuditi zajednici mogućnost zajedničkog rada i usmjeravati razvoj *Hyperledger* tehnologija

3.1. Programski okviri i alati

Hyperledger projekt trenutno sačinjava 6 programskih okvira za razvoj *blockchain* aplikacija i 8 pomoćnih alata. Trenutno je svaki od *Hyperledger* projekata pod jedinstvenom *Apache* autorskom licencom. Voditelj projekta, Brian Behlendorf, bivši glavni programer Apache HTTP servera i utemeljitelja *Apache Software Foundation*, ističe ovo svojstvo kao iznimno bitno, jer omogućuje nesmetan prijenos ideja i koda iz projekta u projekt čime je osigurana velika prilagodljivost budućim izmjenama i načinima korištenja *blockchain* tehnologija. U ovom dijelu rada okvirno ću opisati individualne programske okvire i alate, vidljive na slici 1, zatim ću detaljnije opisati svakih od njih u daljnjem tekstu.

3.1.1. Programski okviri

Od 5 navedenih programskih okvira u *Hyperledger* projektu, svi pružaju podršku za prilagođene pametne ugovore osim *Indy* okvira koji je specijaliziran za pružanje digitalnog distribuiranog identiteta. Ostalih pet programskih okvira imaju poprilično razinu preklapanja funkcionalnosti, s obzirom da svi oni nude osovinu za izradu *blockchain* aplikacija, pa tako svi oni nude jedan ili više načina izvršavanja pametnih ugovora, održavanja konsenzusa na mreži i spremišta podataka u lančanoj strukturi. Programska okruženja u *Hyperledger* projektu su

Slika 1: Pregled Hyperledger obitelji [8]

dizajnirani za opću namjenu, odnosno, nisu specijaliziran za određenu primjenu kao što je to slučaj sa velikim brojem već postojećih i aktivnih *blockchain* rješenja.

3.1.1.1. Burrow

Burrow je sustav koji omogućuje izvršavanje pametnih ugovora prema *Ethereum* specifikaciji uz kontrolu dopuštenja[15]. Ovakvo riješenje olakšava prelazak sa javnog *blockchaina* temeljenog na *Etherium* tehnologiji zbog nativne podrške za *Solidity* jezik koji se koristi za rad sa Etherium Virtualnom Mašinom (engl. *Etherium Virtual Machine*). Ovo riješenje originalno je postojalo pod imenom *eris-db* i prvi put obavljene 2014. godine, a u *Hyperledger* lepezu došao je 2017. godine uz pokroviteljstvo Intel-a.

Hyperledger Burrow koristi *Tendermint* konsenzus protokol, strani projekt pod Apache licencom, preko vlastitog ABCI (engl. *Application Blockchain Interface*) sučelja kako bi osigurao neovisnost između konsenzus protokola i aplikacijskog koda[1].

3.1.1.2. Fabric

Fabric je čelni proizvod *Hyperledger* projekta, originalno kontribuiran od strane IBM korporacije, trenutno slovi kao najrazvijeniji i najzreliji od ponuđenih programskih okvira. Pametni ugovori unutar *Fabric* okvira čvrsto su povezani za imovinom sa kojom upravljaju, te se skupno zove *chaincode*. Dakle, *chaincode* definira način rada sa određenom imovinom, tako da sadržava sve operacije koje je moguće izvesti sa tom imovinom, primjerice pregled stanja, ili prijenos imovine. Svaki novi *chaincodea* izvodi se u novom *Docker* kontejneru, te rezultira novim unosom u transakcijskom dnevniku, te potencijalnom izmjenom u aplikacijskom stanju u obliku ključ-vrijednost parova.[5]

3.1.1.3. Grid

Grid je okruženje koje se sastoji od više postojećih *Hyperledger* tehnologija referentnoj implementaciji za rad sa lancom nabave. Rješava česte probleme sa kojime se susreću aplikacije u ovoj domeni, odnosno nudi već testirane i funkcionalne komponente za obavljanje funkcija potrebne aplikacijama ovog tipa na način koji je testiran i prati pravila struke, ovi problemi uključuju rad sa pravima i ulogama, upravljanje identitetima i praćenje robe.

3.1.1.4. Indy

Indy je *blockchain* riješenje za distribuirani identitet, *The Sovrin Foundation* ga je prenio u *Hyperledger* projekt 2016. godine. Jedini je od programskih okruženja koji ne podržava proizvoljno dodavanje pametnih ugovora, jer funkcionira na način da omogućuje korisnicima da sami upravljaju svojim podacima na mreži, a uz pomoć vjerovnika, ostali korisnici mogu biti sigurni u autentičnost podataka korisnika[6].

3.1.1.5. Iroha

Iroha slovi kao jednostavniji od ostalih programskih okvira na način da pruža naredbe sa kojima se upravlja sa stanjem na *blockchainu*, tj. stanjem imovine, popisom dopuštenja i uloga na mreži. Nudi biblioteke za razne programske jezike, uključujući i nativne Android i iOS biblioteke, Python, C++, te NodeJS biblioteke za web aplikacije. Koristi konsenzus protokol otporan na bizantinske greške jasnog imena YAC (engl. "Yet Another Consensus").

3.1.1.6. Sawtooth

Sawtooth je *Intelova* varijanta implementacije *blockchaina*. Za konsenzus koristi unikatni protokol po svojoj naravi u Hyperledger obitelji, Dokaz o proteklom vremenu (dalje u tekstu eng. *Proof of Elapsed Time* (PoET)) . Ovaj protokol iskorištava funkciju Intel procesora odnosno Okoline za Sigurno Izvršavanje (dalje u tekstu eng. *Trusted Execution Environment* (TEE)) , kako bi garantirao da je proteklo određeno vrijeme prije potvrde transakcije, tako da napadaču oteža lažno potvrđivanje na način da mu postane vremenski neisplativo, po čemu je sličan PoW protokolu, ali ne zahtjeva aktivni rad procesora i time nepotrebno trošenje resursa.

3.1.2. Alati

3.1.2.1. Aries

Aries je implementacija decentraliziranog sustava za upravljanje ključevima (dalje u tekstu eng. *Decentralized Key Management System* (DKMS)) koji omogućuje povezivanje unutra mreže ravnopravnih članova i dijeljenje isprava, poruka i ključeva. Primarni cilj ovog alata je omogućiti inteoperabilnost između svakog sustava i identiteta na mreži koristeći decentralizirani identitet (dalje u tekstu eng. *Decentralized Identity* (DID)) . *Aries* je sastavni dio

Indy projekta iz kojeg je i nastao. Trenutno je najveći projekt sa ovakvim ciljevima u svijetu[9], potreba za generičkim rješenjem digitalnog novčanika koja bi se mogla koristiti kao način identifikacije praktički za svaku svakodnevnu potrebu individualne ili pravne osobe je rezultirala da ovaj projekt postane zaseban i neovisan o implementaciji *Indy* projekta sa svrhom da se što prije i što više krene koristiti u stvarnom svijetu.

3.1.2.2. Caliper

Caliper je alat za mjerenje performansi *blockchain* aplikacije, što uključuje brzinu izvođenja pametnih ugovora, vrijeme potvrde transakcije, vrijeme upita u aplikacijsko stanje, korištenje računalnih resursa i ostale značajne metrike.

Trenutno *Caliper* podržava *Fabric*, *Sawtooth* i *Iroha* sustave. Podršku za različite *blockchain* sustave ostvaruje preko svojeg adaptivnog sloja, koji prevodi uobičajne *blockchain* operacije, kao što su pozivanje funkcije pametnog ugovora, upit u stanje i slično, u operacije koje *Caliper* zna izmjeriti i zabilježiti[8].

3.1.2.3. Cello

Cello alat pruža apstrakciju nad upravljanjem *blockchain* mreže tako da omogućuje brzo instaliranje i pokretanje novih čvorova u mreži, te uklanjanje starih, održavanje i pregled trenutno instaliranih čvorova, statusa instalacija i stvarnih ili virtualnih računala unutar kojih se izvode operacije. Jednostavnije rečeno, *Cello* pruža *blockchain* kao servis[2].

3.1.2.4. Composer

Composer je alat koji znatno ubrzava razvoj *blockchain* aplikacija. Koristeći *Composer* korisnik može puno više vremena uložiti u poslovnu logiku aplikacije umjesto da se zamara sa postavljanjem i administriranjem čitave Imovina se definira pomoću jednostavnog domenski specifičnog jezika za modeliranje, a koristeći *EcmaScript* programski jezik definiraju se pametni ugovori. Trenutno *Composer* podržava izradu aplikacija za *Fabric* okruženje, ali krajnji cilj je izgradnja jedinstvenog jednostavnog sučelja za rad sa svim programskim okruženjima u *Hyperledgeru*.

3.1.2.5. Explorer

Explorer je preglednik za stanje na *blockchainu* što uključuje pregled podataka u blokovima, pozivanje funkcija pametnih ugovora, pregled informacija o mreži, transakcijama i greškama. Trenutno podržava samo *Fabric* programski okvir, međutim postoji inicijativa da se podrška proširi i na ostale projekte.

3.1.2.6. Quilt

Hyperledger zajednica smatra da budućnost leži u velikom broju različitih *blockchaina*, što stvara potrebu za jednostavnijom komunikacijom između njih, gdje *Quilt* nalazi svoje mjesto. Alat omogućuje lakšu komunikaciju između *text* digitalnih novčanika, institucija, organizacija, poduzeća i lanaca opskrbe koji koriste *blockchain* i omogućuje im izvođenje zajedničkih transakcija.

Ideja iza *Quilt* projekta ima visoku vrijednost jer pruža standardizirani i kvalitetni protokol za preusmjeravanje transakcija i interoperabilnost, ali i sučelje za jednostavno povezivanje većeg broja *blockchaina* u jednu aplikaciju na većoj razini apstrakcije[8].

3.1.2.7. Transact

Transact je biblioteka koja upravlja izvršavanjem pametnih ugovora, preciznije, ovo znači da regulira planiranje izvođenja transakcija i spremanje rezultata u svjetsko stanje, ali ostavlja konsenzus protokol, prava pristupa, upravljanje blokovima i ostale detalje konkretnoj implementaciji *blockchain* tehnologije. Drugim riječima, *Transact* pruža proširivu implementaciju za rad sa virtualnim mašinama koje izvršavaju program pametnog ugovora.

3.1.2.8. Ursa

Ursa je drugi projekt koji je proizašao iz *Indy* okruženja a radi se o samostalnoj kriptografskoj biblioteci. Biblioteka se trenutno sastoji od dva dijela[7]:

- *Base Crypto* biblioteka nudi više osnovnih kriptografskih operacija kao što je potpisivanje koristeći kriptografiju eliptičnih krivulja (*ed25519* i *secp256k1*) i sažimanje (*blake2*)
- *z-mix* je biblioteka za izradu takozvanih dokaza bez znanja (eng. *Zero-Knowledge proof*) koji omogućuju da stranka A dokaže stranci B istinitost tvrdnje bez da B dozna detalje o tvrdnji. Trenutno *Fabric* i *Indy* trebaju izvoditi ovakve dokaze u svom radu, pa je zadatak *z-mix*-a ponuditi fleksibilnu i sigurnu implementaciju za izradu dotičnih pravila.

Cilj biblioteke je ponuditi riješenje za sve kriptografske potrebe *Hyperledger* obitelji i ostalih[9].

4. Composer alat

Uz pomoć *Composer* alata moguće je brzo kreirati *blockchain* mrežu temeljene na *Fabricu*, te generirati početnu korisničku aplikaciju u *Angular* programskom okviru, testove i REST servise za interakciju sa *blockchainom* temeljem vlastitog modela podataka. Pri generiranju projekta *Composer* će postaviti okolinu sastavljenu od vrlo zrelog i modernog okruženja za razvoj cjelokupne web aplikacije i pozadinskih servisa. Ovo programeru uvelike olakšava posao i ubrzava vrijeme između ideje i njene realizacije bez puno razmišljanja. U ovom poglavlju objasnit ću funkcionalnosti alata, i pokazati ih na praktičnom primjeru.

4.1. O alatu

Sa gledišta aplikacijskog programera svaka *blockchain* implementacija ne nudi puno više od spremišta podataka, što je samo jedan manji dio izrade cjelokupne aplikacije koje se uobičajeno sačinjavaju od korisničkog sučelja i pozadinske poslovne logike koja obavlja funkcije aplikacije, manipulira korisnicima i poslovnim entitetima, te ih dohvaća i pohranjuje u spremište podataka koje je obično u obliku baze podataka temeljene na relacijama, dokumentima ili nečem sličnom. *Composer* funkcionira kao sučelje za *Fabric* mrežu, što znači da se u pozadini uvijek pozivaju funkcije *Fabric* programskog okvira, a komponente koje ću navesti nastavku služe kako bi opisali sloj apstrakcije koji približava tehničku implementaciju nečemu što je bliže poslovnom procesu.

Glavne komponente *Composer* alata su: [3]

- **Jezik modeliranja** sa definiranim pravilima izrade modela podataka i podrškom za tipove podataka, relacije između entiteta, nasljeđivanje, imenske prostore i validacije podataka korištenjem regularnih izraza (eng. *Regular expression*). Na primjeru 4.1 je pokazana definicija modela entiteta imovine (eng. *asset*) `Fish` kojoj je primarni ključ polje alfanumeričkog tipa `fishId` takav da odgovara danom regularnom izrazu, sadrži obavezna polja `type` i `state`, obavezne reference na entitet tipa `Fisher` i `Business` i opcionalnu referencu na `Regulator` entitet.

```
asset Fish identified by fishId {
    o String fishId regex=/^(\w){3,60}$/
    o FishType type
    o FishState state
    --> Fisher fisher
    --> Regulator regulator optional
    --> Business owner
}
```

Listing 4.1: Primjer definiranja modela

- **Liste kontrole pristupa** (dalje u tekstu eng. *Access Control List* (ACL)) kojima definiramo pravila prema kojima učesnici sa određenim ulogama mogu raditi određene ope-

racije nad resursima u poslovnoj mreži. Moguće je definirati veći broj ACL pravila čime se gradi tablica odlučivanja o dozvoli odnosno zabrani pristupa. Pravila u ACL-u se provjeravaju slijedno kako su i definirani, pa čim *Composer* naiđe na pravilo koje odgovara transakciji primjenjuje se odluka prema njemu, a ostala pravila se ignoriraju. Ukoliko ACL nije definirao pravilo za određenu akciju tada je zadano ponašanje da se izvršavanje transakcije zabranjuje. Ovu provjeru *Composer* obavlja automatski prije izvršavanja programskog koda za transakciju koji će primijeniti izmjene, ukoliko se transakcija ocijeni kao dozvoljena, tada se transakcija sprema u *blockchain*. U primjeru 4.2 je definiran tkz. kompleksan ACL koji opisuje da se imovina tipa `hr.foi.fishynet.Fish` može kreirati samo tako da učesnik tipa `hr.foi.fishynet.Fisher` proizvede transakciju tipa `hr.foi.fishynet.CatchFish`, a pri tome mora biti ispunjen uvjet da vlasnik imovine bude isti kao i učesnik u transakciji. Način na koji se definira vlasnik imovine u ovom slučaju je prikazan u isječku koda 4.3.

```
rule FisherCanCatchTheirFish {
    description: "Fishers can catch their own fish"
    participant(p): "hr.foi.fishynet.Fisher"
    operation: CREATE
    resource(a): "hr.foi.fishynet.Fish"
    transaction: "hr.foi.fishynet.CatchFish"
    condition: (a.owner.getIdentifier() == p.getIdentifier())
    action: ALLOW
}
```

Listing 4.2: Primjer liste pravila pristupa

- **Poslovna logika** mreže opisuje se koristeći programski jezik *EcmaScript*. Kroz poslovnu logiku moguće je kreirati novu imovinu, transakcije, učesnike, te emitirati događaje. Kod poslovne logike poznat je još i pod nazivom pametni ugovor. Primjer 4.3 prikazuje definiciju funkcije koja se izvodi kada se proizvede transakcija `hr.foi.fishynet.CatchFish` kao što je definirano u dekoratoru unutar dokumentacije, prema čemu *Composer* zna koju funkciju pozvati za pojedinu transakciju. Ova funkcija kreira novu imovinu tipa `hr.foi.fishynet.Fish` definira joj početne atribute i emitira događaj na mrežu korištenjem *Composer* aplikacijskog sučelja.

```
/**
* When fish is caught this transaction shoud get executed
* @param {hr.foi.fishynet.CatchFish} fish caught transaction
* @transaction
*/
async function catchFishTransaction(tx) {
    const NS = 'hr.foi.fishynet';
    const factory = getFactory();

    // Create the new asset
    const fish = factory.newResource(NS, 'Fish', tx.fishId);
    fish.type = 'TUNA_WILD';
    fish.state = 'STORED';
```

```
    fish.fisher = factory.newRelationship(NS, 'Fisher', tx.fisher.
        getIdentifier());
    fish.owner = factory.newRelationship(NS, 'Fisher', tx.fisher.
        getIdentifier());

    // Add the asset to the blockchain
    const assetRegistry = await getAssetRegistry(`${NS}.Fish`);
    await assetRegistry.add(fish);

    // Emit an event for the creted asset.
    let event = getFactory().newEvent(NS, 'FishKilled');
    event.fish = fish;
    event.fisher = tx.fisher;
    emit(event);
}
```

Listing 4.3: Primjer poslovne logike

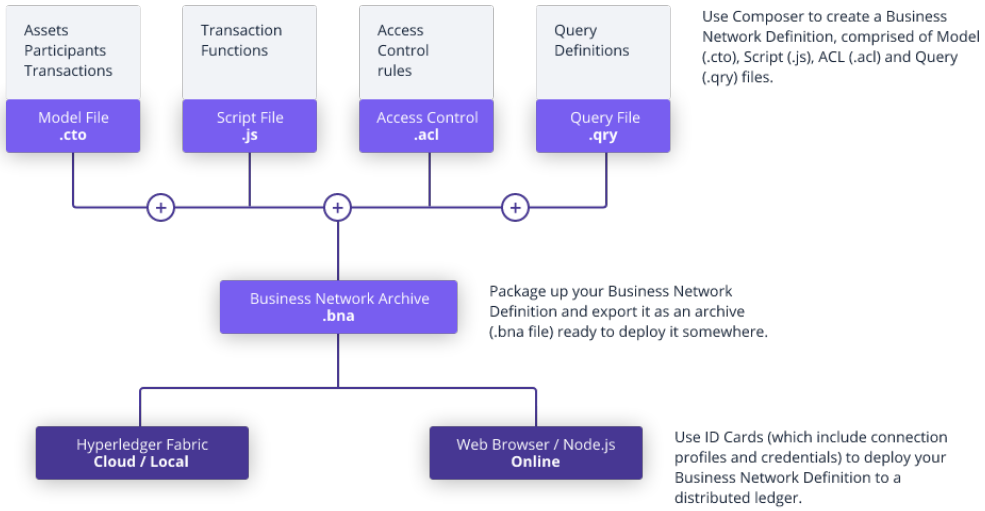
- **Jezik upita** sličan SQL-u kojim se generiraju HTTP GET metode ili pripremljeni upiti koji se mogu pozvati unutar transakcijskih funkcija za uvid u trenutno stanje mreže. U primjeru 4.4 odabiremo sve entitete tipa imovine `hr.foi.fishynet.Fish` tipa `typeParam` koji je proslijeđen u obliku GET parametra. Istovremeno se iz liste vraćaju samo oni rezultati koje određeni korisnik prema svojoj ulozi može i vidjeti prema definicijama iz liste kontrole pristupa.

```
query FishTypeQuery {
    description: "Select all Fish with type PARAM"
    statement: SELECT org.foi.fishynet.Fish WHERE (_$typeParam = type)
        ORDER BY [timestamp DESC, state DESC]
}
```

Listing 4.4: Primjer upita

- **Aplikacijsko sučelje** za upravljanje poslovnom mrežom za korisnike i administratore mreže.
- **REST** sučelje za interakciju za poslovnom mrežom uz pomoć *Loopback* biblioteke.
- **Pomoćni alati** za za razvoj poslovne mreže sastavljene od potpore za *Composer* sintaksu za *Atom* i *VS Code* uređivače teksta i *Yeoman* generator koda za izradu osnovne aplikacije u *Angular* okolini.

Composer se ovdje pojavljuje kao izrazito vrijedan alat iz više razloga. Olakšava rad sa *blockchainom* osobama bez prijašnjeg iskustva u dizajniranju pametnih ugovora zbog dodatne razine apstrakcije između poslovne logike i rada sa *blockchain* implementacijom, čime smanjuje vrijeme potrebno za razvoj i iteraciju funkcionalne mreže, ali i smanjuje rizik od mogućih grešaka u radu izazvanih nepoznavanjem specifičnosti *blockchain* sustava. Trenutno, *Composer* zna koristiti samo *Fabric* programsko okruženje, međutim postoje planovi za razvoj kompatibilnosti sa *Sawtooth* i *Iroha* okruženjima[8].



Slika 2: Datoteke Composer alata (Izvor:, 2018)

Na slici 2 prikazan je dijagram sastavnih dijelova *Composer* okoline. Datoteke `.cto` su sadrže modele entiteta, kao što je prikazano na primjeru 4.1. Svaka datoteka započinje sa deklaracijom svog imenskog prostora i sadrži sve modele koji pripadaju tom imenskom prostoru i niti jedne druge, no zato je moguće imati više datoteka ovog tipa. `.js` drže programski kod u kojem je definirana aplikacijska logika odnosno pametni ugovori, kao što je prije spomenuto na primjeru 4.3. Unutar ovih datoteka dostupne su *Composer* standardne biblioteke za kreiranje novih entiteta, relacija, upita, identiteta i ostale arbitrarne akcija inače dostupne u svakoj *NodeJS* (u trenutku pisanja ovog teksta, zadnja podržana verzija je *Node* 8.16), kao i administracijske akcije za kreiranje, instalaciju i ažuriranje novih ili postojećih pametnih ugovora.

`.acl` datoteka može biti samo jedna u strukturi i sadrži listu pristupa sličnu viđenom u primjeru 4.2. Ova pravila se proširuju i na REST zahtjeve na integriranom *Composer* REST serveru što uključuje i vlastite prilagođene upite koji se definiraju u `.qry` datoteci iz primjera 4.4.

Nakon razvoja aplikacije potrebno je sve izvorne datoteke zapakirati u Arhivu Poslovne Mreže (dalje u tekstu eng. *Business Network Archive* (BNA)) koja se može instalirati na postojeću *Fabric* mrežu za što nam trebaju podaci o administratoru *Fabric* mreže sa dopuštenjima za instaliranje i pokretanje *chaincodea* i izrade odgovarajućih kanala. Specifičnosti o izradi *Fabric* mreže detaljnije ću opisati u poglavlju 6.

4.2. Izrada aplikacije

Praktični primjer predstavljam u obliku skupa testova koji će proći kroz funkcionalnosti demo aplikacije i provjeriti njenu ispravnost. Aplikacija se zove *fishy-network* i dostupna je na javnom repozitoriju github.com/TopHatCroat/fishy-network. Ideja aplikacije je da se omogući praćenje izlova, uzgoja, regulacije i trgovanja krupnom ribom, prvenstveno tunom. Ovo omo-

gućuje regulativnim tijelima, trgovcima i krajnjim kupcima provjeru podrijetla ribe koja se nalazi na tržištu. Među prednostima koje ovo omogućuje je trenutna provjera izvora i lanca nabave ribe, provjeru da li je ilegalno ulovljena ili proizvedena, da li je skladištena prema pravilima i da li je prošla sve nužne regulativne povjere. Također je moguće, ukoliko se otkrije problem sa ribom, primjerice bolest, brzo ući u trag svim ostalim proizvodima koji su došli iz istog izvora u isto vrijeme.

Slučajevi korištenja koje ova demo aplikacija podržava su sljedeći:

- Ribar može uloviti ribu u divljem moru
- Uzgajivač može mrijest tunu pri čemu stvara nove jedinke za uzgajanje
- Uzgajivač može kupiti mladu ribu od ribara kako bi je udebljao u umjetnom uzgoju
- Ribar i uzgajivač može izmjeriti svoju svojstva svoje ribe kao što su težina, udio masnoće i tjelesna temperatura
- Ribar i uzgajivač mogu pripremiti svoju ribu za plasiranje na tržište
- Regulativno može provjeriti ispravnost uzgoja i spremanja ribe prilikom plasiranja na tržište
- Trgovac može kupiti ribu po dogovorenoj cijeni

Rezultat korištenja ovakvog sustava je pojačana sigurnost povećavanjem transparentnosti cijelog lanca nabave ribe, ali i povećanje povjerenja koji potrošači imaju u hranu koju konzumiraju. Aplikaciju je moguće pokrenuti i isprobati koristeći upute dostupne na repozitoriju. Najprije potrebno je namjestiti *Hyperledger Fabric* okolinu, za potrebe ove demonstracije to će biti minimalna funkcionalna verzija. Sastoji se od po jedne instance sljedećih kontejnera: *Fabric* član (*Fabric Peer*), servis za određivanje redoslijeda transakcija (eng-‘. *Fabric Orderer*), tijelo za izdavanje certifikata (eng. *Certificate Authority*) i *CouchDB*. Posljednji ustvari nije nužan za rad mreže, već je opcionalni dodatak koji *Composer* uključuje automatski, jer nudi više opcija za pretraživanje i dohvaćanje podataka, detalje o ovim kontejnerima detaljnije ću opisat u poglavlju 6.

5. Indy

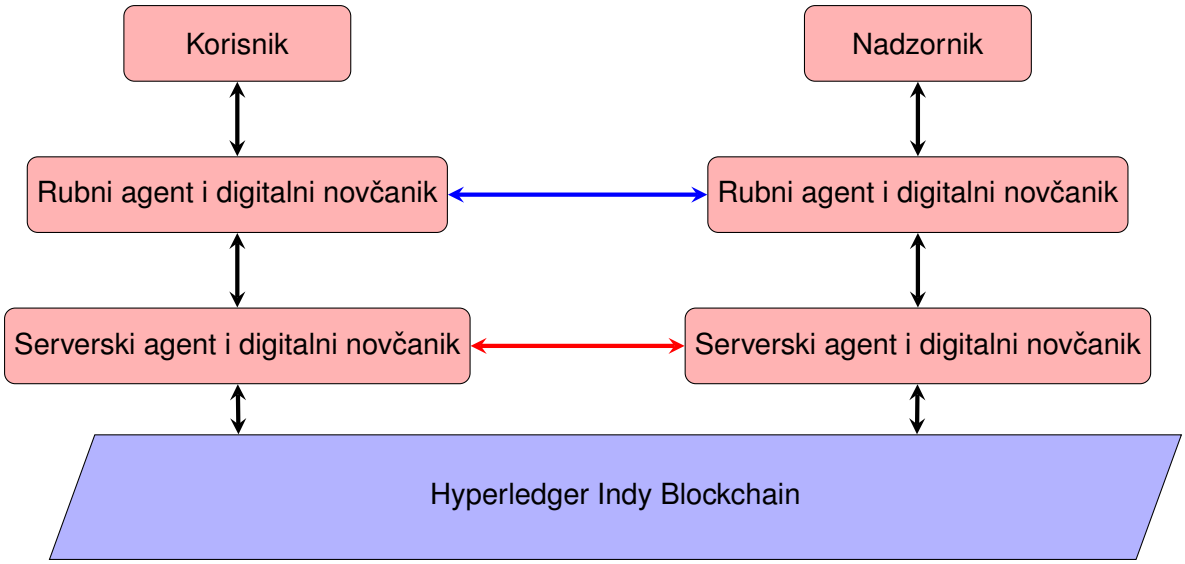
Indy je *blockchain* sustav koji za cilj ima ponuditi funkcionalan distribuirani identitet korisnicima. Projekt je započeo i aktivno razvija *Sovrin Foundation* pod nazivom Sovrin[20], te se u jednom trenutku sačinjavao od dijelova koji su danas samostojeći, *Hyperledger Aries* i *Hyperledger Ursa*, no sada, označava samo *blockchain* mrežu koja se koristi distribuirani rad dijeljenja identiteta i razmjene podataka. Model sustava se zove Samosuveren Identitet (dalje u tekstu eng. *Self-Sovereign Identity* (SSI)) i osmišljen je tako da postoji direktna veza između korisnika i onoga koji zahtjeva identifikaciju, i vrijedi za bilo dvostranu kombinaciju između osobe, organizacije i/ili stroja.

Sustav je sastavljen od 4 segmenta:

- **DID** je globalno jedinstven identifikator javnog ključa na nekom ledger-u, a označava vezu između dva učesnika u procesu identifikacije i autorizacije, te se koristi samo između ta dva učesnika. Nedavno je i službeno standardiziran od strane W3C-a, jedini identifikator koji je dobio tu čast nakon *http(s)*-a[9].
- **DKMS** je sloj za upravljanje i razmjenu ključevima, detaljnije ću ga opisati u sekciji 5.1
- ***DID Auth*** je sučelje prema vanjskim aplikacijama koje koriste sustav autentifikaciju i autorizaciju i vlastitu poslovnu logiku.
- **Potvrdljivi podaci o vlasniku identiteta**

Sustav svojim korisnicima ostavlja kontrolu nad vlastitim podacima tako da se niti u jednom trenutku na *Indy blockchain* ne upisuju privatni ili povjerljivi podaci, nego se oni drže isključivo na osobnom digitalnom novčaniku korisnika koji je kriptografski zaštićen. Ovaj novčanik se može nalaziti na specijaliziranom uređaju, ali najčešće će taj uređaj biti mobilni telefon zbog praktičnosti. Brisanje novčanika znači i brisanje identiteta, odnosno svih DID-ova koje mu pripadaju. Digitalni novčanik je moguće duplicirati na više uređaja, u slici 3 označeni kao rubni agenti, i držati ih sinkroniziranim preko serverskih agenata na kojima se spremaju kopije digitalnih novčanika više korisnika na siguran način. Ovakav način rada je nužan kako bi u svakom trenutku digitalni novčanik korisnika mogao biti dostupan ostalim učesnicima u mreži i da bi bilo moguće obaviti proces oporavka i sinkronizacije.

U slici 3 prikazana je procedura rada *Indy*-a u kojoj sudjeluju korisnik i nadzornik koristeći svoje agente. Odnos je u minimalnom slučaju moguć i samo pomoću rubnih agenata u kojem će se ostvariti direktna veza između dva rubna agenta i dva novčanika kojim upravljaju, ovaj odnos prikazan je plavom strelicom. Međutim, najčešći odnos će uključivati i serverske agente i novčanike, prikazan crvenom strelicom, kao i *Indy blockchain* koji omogućuje otkrivanje i razmjenu početnih podataka. Na sam *Indy blockchain* spremaju se (dalje u tekstu eng. *DID descriptor object* (DDO)) , JSON tip objekta koji sadrži javne ključeve koji su povezani sa nekim DID-em i skup poveznica koje vode na agenta serverski agent koji je zadužen za korisnika kojem DID pripada.



Slika 3: DID stog[10]

5.0.1. DID

DID se sastoji od tri dijela odvojenih sa dvotočkom, primjer DID-a je sljedeći:

`did:sov:21tDAKCERh95uGgKbJNHYp`

Prvi (plavi) dio je identifikator protokola, svaki DID počinje sa ova tri slova. Drugi dio (crveni) označava mrežu na kojoj se nalazi ovaj DID, konkretno u ovom slučaju označava Sovrign mrežu koja je i trenutno najveća. Treći dio (ljubičasti) je konkretna adresa DDO-a na koji vodi ovaj DID.

Nakon inicijalne izmjene podataka, dvije stranke koje žele potvrditi i provjeriti međusobne podatke mogu izgraditi povjerenje koristeći standardnu eng. *Public Key Infrastructure* (PKI) već dobro poznatu i provjerenu, ili će izmijeniti vlastite potvrdljive atribute identiteta koje su potpisane od treće strane čiji javni ključevi mogu biti dokazani na *Indy Blockchainu* a obje strane joj vjeruju[18].

DIgitalni novčanik sadrži:

- DID-ove
- Privatne i javne ključeve
- Veze prema serverskim agentima
- Tajne
- Podaci o identitetu
- Kriptografske tokene

Kako bi se omogućio ovaj način rada, sustav poznaje dvije različite uloge: korisnik i nadzornik. Korisnik je odgovoran za čuvanje svojih podataka, a u određenom trenutku može

poslati zahtjev nadzorniku koji potvrđuje navedene podatke čime se stvara vjerodostojna tvrdnja o podacima korisnika[10].

Ove tvrdnje nazivaju se dokazi bez znanja (dalje u tekstu eng. *Zero-knowladge proof* (ZKP)) , kriptografska tehnika koja omogućuje korisnicima dijeljenje informacija bez gubitka sigurnosti i privatnosti. ZKP ima tri atributa: kompletnost, što znači da se rezultatu dokaza može vjerovati, očuvanost, što znači da nije moguće kreirati rezultat koji je lažan, a da to nije očito i, na kraju, treba biti bez znanja, odnosno da druga strana ne otkrije ništa više o strani koja nešto dokazuje osim samog sadržaja dokaza.

5.0.2. Potvrdljivi podaci o vlasniku identiteta

U današnjem svijetu, korištenje zajedničkih identifikatora postaje veliki problem za privatnost na internetu zbog mogućnosti da se odvojeni podaci o korisniku na većem broju servisa povežu međusobno korištenjem, u najčešćem slučaju, adrese elektroničke pošte korisnika. Najveći krivci u ovim scenarijima su oglašivačke kuće koje prate ekstremnu većinu korisnika interneta, te pomoću različitih servisa treće strane dolaze u posjed više čestica podataka o svojim korisnicima poput: imena, slike, datuma rođenja, adrese, religije, interesa, bogatstva i ostalih. Ono što najviše omogućava ovakvo skupljanje podataka su identični identifikatori koji prosječan korisnik dijeli posvuda na internetu, kao što je prije spomenuta, adresa elektroničke pošte ili telefonski broj ili korisnička oznaka (eng. *username*), te činjenice da nije moguće ograničiti pristup podacima koje dijelimo, odnosno, u trenutku kada neke podatke pošaljemo nekom od servisa na internetu, tada se gubi i kontrola nad tim podatkom. Iako je, nedavni EU zakon, GDPR do neke mjere olakšao drugi navedeni problem, i dalje to ne garantira ispravno postupanje sa svim podacima, niti uzima u obzir mogućnost da servis izgubi kontrolu nad danim podacima, odnosno da oni budu napadnuti i podaci preuzeti.

Potvrdljivi podaci o vlasniku identiteta su različite čestice podataka koje je moguće digitalizirani i kriptografski osigurati i naknadno potvrditi. Ovi podaci mogu biti napisani na ispravama poput osobne iskaznice, vozačke dozvole ili na nekim od povjerenih servisa na internetu. Ovakvi podaci omogućuju korisniku da dokaže određene tvrdnje o sebi na trivijalan način, tako da ih pokaže drugoj strani koja te podatke želi potvrditi. No ovaj način nam nije zanimljiv, jer je to način na koji ovakve operacije radimo, a već smo zaključiti da je postalo bolno očito da ne zadovoljava.

Prilikom potvrde podataka onaj koji potvrđuje može biti siguran, jer je to kriptografski garantirano, u istinitost o tome da su podaci potvrđeni od strane koju potvrditelj očekuje, da su podaci izdani točno onom korisniku koji ih predstavlja, da podaci nisu mogli biti ikako promijenjeni, tj. da su autentični, te da podaci nisu opovrgnuti od strane njihovih izdavača. Prilikom potvrde podataka, jedine dvije strane koje su uključene u proces su onaj koji želi potvrditi svoje podatke i onaj koji zatražuje na provjeru, što znači da treća strana koja je izdala i potvrdila točnost podatka nije svjesna gdje, kada i koliko često se dotični podaci koriste, što je bitan aspekt ovog sustava, jer se tako održava privatnost korisnika.

Hyperledger Indy nudi čitavo rješenje navedenih problema zbog mogućnosti da drugoj

strani daje minimalna potrebna količina podataka, te da nužno druga strana ne treba znati potpunu informaciju. Recimo, pristup nekome servisu zahtjeva da korisnik bude punoljetan, ovo bi inače zahtjevalo da slanje datuma rođenja i neke potvrde da je taj datum točan, no korištenjem *Indy*-a i prije spomenutog ZKP, moguće je dotičnom servisu samo odgovoriti na pitanje: "Da li imate više od 18 godina?" sa binarnim odgovorom "Da" ili "Ne". Detaljniji opisi ZKP-a su izvan opsega ovog rada, ali o ovoj zanimljivoj kriptografkoj tehnici moguće je pročitati više na izvoru [13].

5.1. DKMS

DKMS je novi pristup upravljanju kriptografskim ključevima namijenjen korištenju sa DLT-om gdje ne postoje centralizirani autoriteti. Trenutni sustav koji se koristi u svijetu oslanja se na manji broj entita koji uživaju povjerenje većine, te imaju načina da na siguran način održavaju svoje privatne kriptografske podatke sigurnima, koje kasnije mogu koristiti kako bi to povjerenje proširilo na svoje klijente. Ovakav sustav je podložan problemima, jer jedna greška na ključnom mjestu može uzrokovati probleme sa radom velikom dijela sustava. Nemogućnost privatne i neovisne razmjene povjerenja individualnih osoba i organizacija ima više posljedica. Primjerice, obvezuje individualne osobe i organizacije da koriste poznate i centralizirane poslužitelje digitalnih certifikata, odnosno povjerenja, koji imaju mogućnost diktiranja sigurnosti, privatnosti i poslovnih odluka, dugi nedostatak je da ograničava mogućnosti dijeljenja danog povjerenja van kanala na kojima su poslužitelji dostupni, te također, ograničava mogućnosti otkrivanja i izgradnje novih veza i povjerenja van kontrole poslužitelja[18]. DKMS izokreće ovaj način rada, omogućuje da svatko na internetu može biti izvor povjerenja ukoliko mu druge stranke vjeruju. Ovo je puno sličnije onome kako funkcionira svijet i u fizičkoj domeni, primjerice, fakultet izdaje diplomu čime potvrđuje završetak studija osobe navedene u diplomi, zatim svatko tko ima povjerenje u fakultet u pitanju može prema dotičnoj diplomi se uvjeriti u određenu razinu spremnosti osobe koja je posjeduje, ukoliko netko nema povjerenja u određeni fakultet, tada neće tu diplomu uvažiti kao adekvatna potvrda o spremnosti osobe koja ju posjeduje.

Ovakav način rada dovodi do toga da ne postoji jedna točka koja može popustiti i time negativno utjecati na rad većeg dijela sustava, granulacija povjerenja pomaže u tome da ukoliko se i desi propust, on je ograničen samo na manji skup potvrdljivih podatka. Još jedna bitna stavka je i interoperabilnosti sustava, DKMS protokol omogućuje da bilo koja dva entiteta mogu razmijeniti svoje podatke, bez ovisnosti o trećoj strani koja ima potencijal unositi dodatne uvjete ili pravila rada. Vjerojatno najbitniji aspekt upravljanja ključevima je pitanje kako se oporaviti od njihova gubitka, odnosno gubitka pristupa novčaniku. U najgorem slučaju da subjekt izgubi pristup svim svojim uređajima koji sadrže njegove novčanike, onda je korisniku preostaju dvije opcije: van-mrežni oporavak sa fizičkim ključevima ili društveni oporavak. Van-mrežnih oporavak se ostvaruje putem ključa koji je zapisan na fizičkom mediju, na primjer, na papiru ili na USB ključu. Za društveni oporavak, korisnik najprije mora odabrati više korisnika kojima će povjeriti dijelove svojeg ključa za oporavak, zatim odabrani korisnici mogu odobriti zahtjev za oporavak ukoliko su svi složni oko tog postupka. Točan broj korisnika i uvjeti pod kojim se socijalni oporavak može izvesti nije još definiran u potpunosti[9].

5.2. Primjer

Na primjer, korisnici mogu biti građani RH, a jedan od nadzornika(eng. *steward*) može biti HZZO. Odnos između nadzornika i korisnika počinje tako da nadzornik pošalje zahtjev za spajanje korisniku koji sadrži novi DID nadzornika prema kojem želi da ga korisnik može prepoznati, mrežna lokacija (eng. *endpoint*) na rubnom agentu na koju korisnik može poslati zahtjev ukoliko prihvaća vezu i javni Ed25519 ključ prema kojem korisnik može potvrditi autentičnost poruka koju nadzornik šalje. Pretpostavimo, zbog jednostavnosti, da korisnik već vjeruje da je ovaj početni zahtjev autentičan, primjerice da je uz zahtjev primljen putem digitalno potpisane e-pošte, preko službenog portala sa odgovarajućom zaštitom, ili u najčešćem slučaju, zahtjev je primljen preko mobilne aplikacije koja podržava Indy mrežu, te djeluje kao korisnikov rubni agent. Ukoliko korisnik prihvatu ovu vezu, aplikacija će u njegovo ime odabrati novi DID koji će predstavljati identifikator korisnika u vezi sa nadzornikom, novi par javnog i privatnog ključa, novu tajnu za vezu (eng. *link secret*, detaljniji opis dalje u tekstu), te politiku potpisivanja za agenta u kojoj je u početku zapisano da agent u pitanju (uređaj koji korisnik koristi, primjerice, pametni telefon) je autoriziran predstavljati korisnika i obavljati akcije u njegovo ime kao što je serviranje mrežnih lokacija, predstavljanje korisnikovih podataka i dopuštenje za eventualnu izmjenu same politike potpisivanja.

U ovom primjeru, korisnik će zapisati podatke o statusu svog zdravstvenog osiguranja, te ih poslati u HZZO na potvrdu. HZZO tada stvara tvrdnju o točnosti predanih podataka kojom se korisnik može služiti u zdravstvenim ustanovama ili za neke druge potvrde koje traže ovakve podatke, kako bi potvrdio svoj status pomoću privatnog novčanika.

6. Fabric

Hyperledger Fabric (HLF) je vrlo zrela implementacija DLT-a modularnog dizajna. Korijenske funkcionalnosti koje nudi su integrirano upravljanje identitetima i pravima korisnika, privatnost i povjerljivost, paralelnu i učinkovitu obradu podataka i promjenjivu poslovnu logiku zahvaljujući pametnim ugovorima koji se u nomenklaturi HLF okruženja nazivaju pametni ugovori ili eng. *chaincode*. Posljednja stabilna verzija u trenutku pisanja ovog teksta je 1.4, stoga će se svi sljedeći opisi odnositi na ovu verziju.

U nastavku teksta bitno je razlikovati sljedeća dva izraza koje ću koristiti:

- **"Ravnopravni član"**, ili samo "član", (eng. *peer*) mreže odnosi se na digitalni entitet koji svojim računalnim resursima sudjeluje u radu čitave mreže, te izvršava transakcije i administrativne poslove u ime ili u korist korisnika ili organizacije.
- **"Korisnik"** se odnosi na fizičku osobu koja koristi HLF preko članova i u ime organizacije.
- **"Organizacija"** se odnosi na pravnu osobu sačinjenu od jednog ili više korisnika.

U kratko, članovi su osovina HLF mreže koji u sebi imaju spremljenu instancu *blockchaina* i pametnih ugovora, a u mreži pridružene organizacije sudjeluju kroz njihove korisnike koji pokreću i promatraju transakcije.

6.1. Transakcijski proces

Jedna od najbitnijih stvari koja razlikuje većinu *Hyperledger* programskih okvira, uključujući i *Fabric*, je činjenica da sadrži ugrađeni model autentifikacije i autorizacije. Ovo znači da su korisnici u transakcijama poznati sustavu što omogućuje i upravljanje pravima koje posjeduju različiti korisnici. Ova činjenica omogućuje korištenje ovih sustava u slučajevima kada su zakonski propisane obveze spremanja identiteta korisnika mreže u svrhu poznavanja tko ima pristup određenim podacima te autoritet izvršavanja određenih postupaka, primarni primjeri ovakvog slučaja korištenja možemo pronaći u financijskom sektoru ili javnim uslugama.

Modularna *Fabric* arhitektura dijeli transakcije na tri faze kako bi se povisile cjelokupne performanse sustava, proširio potencijal skaliranja i povećao kapacitet mreže. Ove tri faze su sljedeće:

- Simuliranje transakcije
- Redanje transakcija u blokove
- Potvrda i spremanje transakcije

Nakon pribavljanja identifikacije na mreži, koju će mo više opisati u sekciji 6.4, korisnik može pozvati funkciju pametnog ugovora tako da prijedlog transakcije pošalje na jedan ili više

članova kojim ima pravo pristupa. Kojim članovima i kojem broju njih ovisi o politici odobravanja koju ću opisati u poglavlju 6.5. Član ili članovi koji su primili prijedlog transakcije će ovjeriti transakciju na način da najprije provjere ispravnost formata prijedloga, zatim da transakcija nije već prije bila provedena i da je potpis korisnika koji je inicirao transakciju ispravan i da posjeduje odgovarajuća prava za izvršavanje transakcije na danom kanalu, o kojima će više priče biti u poglavlju 6.6.1. Nakon uspješne potvrde ispunjenosti uvjeta za transakciju, član izvršava simulaciju transakcije koristeći ulazne parametre koje je korisnik poslao i trenutno stanje na mreži. Ovim postupkom odobravajući član proizvodi digitalno potpisani skup podataka koji se sastoji od skupa povratne vrijednosti, skupa pročitanih podataka i skupa zapisanih podataka, ukoliko takvi postoje. Zadnja dva skupa su kolekcija ključ-vrijednost parova sačinjenih od podataka iz trenutnog globalnog stanja koji su u dotičnoj transakciji pročitani i zapisani. Detalji o formatu i načinu spremanja ukupnog globalnoga skupa podataka nalaze se u sekciji 6.6. U ovom trenutku, još se nikakve izmjene nisu zapisane u globalno stanje, te u slučaju kada je cilj samo čitanje stanje, ovdje se transakcija može zaustaviti. Međutim, ukoliko aplikacijska domena zahtjeva mogućnost praćenja i čitanja, primjerice, ako je bitna informacija o tome da li je korisnik bio svjestan određene činjenice o svjetskom stanju, tada se proces nastaviti kao i za ostale transakcije.

Ukoliko je transakciju potrebno zapisati u svjetsko stanje, tada će aplikacijski klijent u ime korisnika prikupiti sva odobrenja transakcije, te provjerava ispunjenje svih uvjeta prema politici odobravanja iz poglavlja 6.5. Zatim se odobrenja šalju servisu za određivanje redoslijeda transakcija (dalje u tekstu eng. *Fabric Ordering Service* (FOS)) . FOS će kronološki poredati transakcije, te ih grupirati u blokove. To je sve što će FOS napraviti, što znači da nikad neće čitati sadržaj transakcije, već ih samo organizirati u blokove koji će se, u sljedećem koraku ovog procesa, potvrditi i zapisati u svjetsko stanje.

Kako bi se izmjene iz transakcija mogle zapisati, FOS šalje blokove poredanih transakcija članovima kako bi ih ponovno potvrdili. Oni će još jednom provjeriti dali je politika odobravanja ispoštovana u svim transakcijama i da se podaci iz skupa pročitanih podataka nisu promijenili između prve i druge provjere. Ukoliko su oba uvjeta ispunjena, transakcija se označava kao uspješna, te će njen rezultat utjecati na svjetsko stanje, u suprotnom, transakcija se označava kao neispravna i svjetsko stanje ostaje netaknuto. Bez obzira na uspješnost transakcije, ona će biti evidentirana u *blockchainu*.

Transakcija započinje tako da korisnik mreže pozove određenu funkciju definiranu unutar pametnog ugovora sa odgovarajućim parametrima. Zatim transakcije na mreži izvode ravnopravni članovi izvršavajući instrukcije zapisane u funkciji pametnog ugovora, također znanog kao eng. *chaincode*, koji definiraju sve moguće operacije na mreži.

Prilikom izvršavanja pametnih ugovora stvara se set promjena s obzirom na trenutno stanje, što je zapravo prijedlog transakcije. U ovom trenutku, transakcija još nije evidentirana na mreži kao izvršena, već se samo stvara prijedlog koji se šalje natrag klijentu, a on ga prosljeđuje FOS, neovisnom entitetu unutar mreže čiji je zadatak poredati transakcije u niz kako bi se osigurala konzistentnost stanja. FOS zatim sastavlja jedan blok koji sadrži veći broj kronološki poredanih transakcija, te ih šalje svim članovima koji ih ponovno provjeravaju, te se temeljem

konsenzus protokola dogovaraju o valjanosti predloženog bloka, te ukoliko se slože, svaki član zapisuje novi blok u svoju instancu *blockchaina* čime su transakcije u bloku potvrđene u potpunosti.

Do sada smo zaključili da nam je potreban pametan ugovor da pročitamo ili izmijenimo stanje na *blockchainu*, ali pitanje koje ostaje ne odgovoreno je kako nastaju pametni ugovori na mreži. Pametne ugovore potrebno je najprije instalirati na člana, zatim se on instancira čime ugovor postaje dostupan za korištenje na mreži, te je moguće pozivati njegove funkcije. Svaka organizacija ima pravo instalirati svoje pametne ugovore na svoje članove, ali ovakav pristup može dovesti do fragmentacije funkcionalnosti preko cjelokupne mreže, odnosno nemogućnosti da članovi različitih organizacija budu međusobno kompatibilni, pa je bolje riješenje izrada pametnog ugovora koji može upravljati pametnim ugovorima u domeni aplikacije, odnosno, instalirati ih, instancirati, te potencijalno ukloniti sa mreže na način koji će biti brz i neprimjetan. Na ovakav način moguće je implementirati, na primjer, sistem glasovanja kroz mrežu prema kojem bi se odlučilo koji se ugovor instalira ili uklanja.

6.2. Pametni ugovori

Pametni ugovor je računalni program koji sadržava poslovnu logiku i instrukcije za izvođenje transakcija koje kreiraju, izmjenjuju ili brišu imovinu u *blockchainu*. HLF trenutno podržava pametne ugovore napisane u jezicima: Go, JavaScript (Node) i Java. Ovi programi se izvršavaju u *Docker* kontejnerima povezani sa članom za kojeg se izvode, tako da je svjetsko stanje proizvedeno nekim pametnim ugovorom dostupno samo unutar njega. /todoBit more info here Ovo znači da nije moguće pristupiti svjetskom stanju drugog pametnog ugovora direktno, međutim, moguće je unutar pametnog ugovora pozivati funkcije drugih pametnih ugovora, ukoliko su originalnom korisniku funkcije dostupna adekvatna prava, kako bi se dohvatili potrebni podaci za izvršavanje transakcije ili izvela neka dodatna transakcija. Dodatne transakcije mogu biti na istom ili na drugom kanalu. Ukoliko je transakcija na istom kanalu tada se skup pročitanih i zapisanih podataka jednostavno dodaje skupu izvorne transakcije koji prolazi kroz prolazi kroz cijeli transakcijski proces opisan u prethodnom poglavlju. Međutim, ukoliko se dodatna transakcija nalazi na drugom kanalu, tada ona može samo čitati podatke i ne može mijenjati svjetsko stanje drugog kanala.

Prije korištenja pametnog ugovora on mora biti instaliran na članu i pokrenut na kanalu. Sve organizacije koje žele izvoditi transakcije ili čitati podatke pametnog ugovora moraju ga instalirati na svog člana. Korisnik sa adekvatnim pravima može instalirati pametni ugovor na članove kojima ima pristup, zatim korisnik koji ima adekvatna pravila za pokretanje pametnog ugovora to radi na razini čitavog kanala što se propagira na sve članove koji su mu pridruženi. Prilikom pokretanja kanala, prenose se i inicijalne vrijednosti ili se izvodi migracijska logika sa starije verzije ugovora. U slučaju migracije sa stare inačice na novu, slična pravila vrijede, novi kod potrebno je instalirati na sve članove koji ga žele koristiti, jer u suprotnom ti članovi neće imati pristupa mreži nakon pokretanja nove verzije ugovora od strane administratora kanala. Zbog ovog detalja je izrazito bitno korigirati sa svim organizacijama i njihovim administratorima na mreži kada i kako će se slati nova verzija koda u produkciju. Na svakom novom članu koji se

pridruži određenom kanalu, pametni ugovor se automatski pokreće ukoliko ga dotični član ima instaliranom, ili se automatski pokreće u trenutku instalacije pametnog ugovora na tog člana. Koristeći istu instalaciju pametnog ugovora, član kože pristupiti većem broju kanala uz uvjet da su ime i verzija pametnog ugovora ista kao i aktualni na kanalu.

Za potrebe upravljanja vlastitim radom HLF sadrži i nekoliko sistemskih pametnih ugovora, koji prolaze kroz isti transakcijski proces kao i svi ostali. Izvorni kod dostupan na github.com/hyperledger/fabric/blob/master/core/scc je odličan primjer kako pametni ugori u svim aplikacijama trebaju izgledati, a uključuju sljedeće:

- **QSCC** (eng. *Query System Chaincode*) koji služe sa čitanje blokova, postojećih transakcija i meta podataka o *blockchainu*.
- **CSCC** (eng. *Configuration System Chaincode*) koji član poziva prilikom prvog pridruživanja u mrežu kako bi kreirao svoju početnu konfiguraciju ili dohvatio postojeću. Administrator člana ili kanala također može pozvati ovaj pametni ugovor kako bi izmijenio trenutnu konfiguraciju člana ili kanala.
- **LSCC** (eng. *Lifecycle System Chaincode*) je višestruko veći od prethodno navedena dva, a kao što mu ime govori, upravlja životnim ciklusom svih pametnih ugovora na mreži. Sa ovim pametnim ugovorom korisnik sa administrativnim pravima može instalirati, pokrenuti i nadograditi druge pametne ugovore, te izvoditi upite poput provjere postojanja određenog pametnog ugovora, dohvaćanja podatka o trenutnim pametnim ugovorima na članovima ili kanalu i upravljanjem i dohvaćanjem privatnih podataka 6.6.2.

6.3. Servis za redanje transakcija

Iz opisa životnog ciklusa transakcije u poglavlju 6.1 može se zaključiti da je jedna od najbitnijih stavki u HLF servis za redanje transakcija, odnosno FOS. Kao takav je i najosjetljivija točka sustava, pa je izuzetno bitno održati ovaj servis zdravim za rad čitave mreže. FOS će svaku transakciju na kanalu poredati po kronološkom redoslijedu i grupirati ih u blokove. Veličina i vremenski razmak ovih blokova ovisi o konfiguraciji samog kanala. S obzirom na važnost ovog aspekta mreže, bitno je da je ovaj sustav decentraliziran, optoran na mreže, te da nije pod kontrolom jedne organizacije već da je njegova uloga distribuirana među više pripadnika mreže.

HLF trenutno podržava tri implementacije FOS-a, a to su: *Solo*, *Kafka* i, nedavno dodani i najnapredniji sustav, *Raft*.

6.3.1. Solo

Solo sustav sastoji se od samo jedne instance jednostavnog servisa što znači da nije decentralizirani niti je otporan na greške, već je dizajniran da bude najjednostavniji mogući. Ova implementacija je namijenjena korištenju samo za razvojne svrhe i ne preporuča se koristiti u produkciji.

6.3.2. Kafka

Kafka je već poznati sustav koji nudi funkcionalnost slanja i primanja poruka u distribuiranom sustavu koji je već testiran u stvarnom svijetu i spreman za produkciju. Koristeći *ZooKeeper* sustav za upravljanjem većim brojem instanci *Kafka* servisa moguće je dobiti servis za redanje transakcija koji je otporan na kvarove (dalje u tekstu eng. *Crash fault tolerance* (CFT)) što znači da čitava grupa poslužitelja može nastaviti funkcionirati u slučaju kada polovina instanci - 1 od ukupnog broja poslužitelja prestane raditi zbog neke vrste kvara, poput odspajanja sa mreže ili slično. U CFT sustavima postiže se kvorum kada se N/2 + 1 instanci slaže oko nekog zaključka, jer je potrebna većina kako bi se neki zaključak prihvatio, ali ovo ne garantira ispravnost u slučaju kada je dio mreže pod kontrolom treće strane koja za to nema pravo, odnosno, napadnuto je. Međutim, *Kafka* i *ZooKeeper* su dizajnirani da budu pod kontrolom jedne organizacije, te nisu napravljeni da se pokreću na velikim mrežama sa većim brojem članova i organizacija, već u manjim grupama. Ovo znači, da u stvarnom svijetu, i dalje postoji problem da ovim servisom jer njime upravlja samo jedna organizacija, što nam ne garantira potpuno ravnopravnu mrežu već i dalje održava određenu razinu centraliziranosti.

6.3.3. Raft

Raft protokol za konsenzus je najnoviji dodatak u HLF ekosustavu, dostupan je od verzije 1.4.1 (Travanj, 2019). U budućnosti se očekuje da će *Raft* preuzeti poziciju kao standardni servis za redanje transakcija. Izvan dosega ovog rada je ulaziti u detalje rada *Raft* protokola, ali bitno je znati da je to već postojeći protokol za postizanje dogovora između više poslužitelja koji mogu dijeliti isto stanje sa kojim se svi slažu, te u slučaju kvara, reagirati i popraviti ga.

Za potrebe FOS-a, *Raft* bolje odgovara od *Kafke*, jer je moguće imati više članova sustava koji pripadaju različitim organizacijama, što znači da se uklanja taj aspekt centraliziranosti koji je postojao kod *Kafke*, no zadržavajući otpornost na kvarove. Međutim, preostaje još jedna prepreka do idealnog servisa za redanje transakcija, a to je (dalje u tekstu eng. *Byzantine fault tolerance* (BFT)) . Razlika između CFT i BFT servisa je to da CFT sustavi garantiraju ispravnost rada u slučaju N / 2 instanci koje nedostaju zbog neke vrste kvara, dok će BFT sustavi garantirati ispravnost rada u slučajevima kada N / 3 instanci prestane raditi zbog kvara ili radi sa lošim namjerama u korist treće strane.

Ovaj aspekt dobio je svoje ime po problemu dvaju bizantinskih generala[14] koji glasi ovako: dva generala napadaju su suparnički dvorac, odvojeno, nisu dovoljno snažni da osvoje dvorac, jedini način na koji imaju šansu je da napadnu zajedno, međutim, svaki od generala nalazi se na suprotnim brežuljcima, te kako bi se dogovorili o vremenu napada moraju slati pismonoše kroz dolinu u kojoj se nalazi dvorac, što ga izlaže tome da neprijatelj ubije pismonošu ili na njegovo mjesto stavi špijuna koji će prenijeti krivo vrijeme napada. Generali mogu osigurani određenu razinu povjerenja u odabrano vrijeme napada tako da prvi general pošalje prijedlog vremena, a drugi general pošalje potvrdu da je vrijeme prihvatio, međutim, koliko god pismonoša poslali, ne mogu osigurati da je zadnja poruka stigla i da je ispravna. Ovaj problem smatra se nerješivim.

Na ovaj način, neka od organizacija koja upravlja jednim ili više instanci servisa u grupi poslužitelja FOS-a može imati loše namjere ili biti pod kontrolom treće strane koja nema pravo za to bez znanja ostatka mreže. U ovom scenariju napadač može preferirati transakcije jedne od organizacija, ili odbijati transakcije nekim organizacijama. Ovo može negativno utjecati na rad mreže, međutim koristeći *Raft* protokol ovaj problem se pokušava ublažiti tako da za kontrolu FOS-a je potrebno preuzeti kontrolu nad većinskim dijelom mreže, što u dovoljno velikom sustavu može postati iznimno teško za izvesti. Također, ukoliko napadač uspije preuzeti kontrolu nad FOS-om, tada i dalje postoje ostali koraci odobravanja transakcija opisanih u poglavlju 6.1 koji će onemogućiti ovim transakcijama da se zapišu na kanalu. No bitno je napomenuti da BFT FOS nije potreban u svom slučajevima korištenja, ukoliko je mreža namjenjena internom korištenju unutar jedne organizacije ili manjeg broja organizacija koje su si međusobno poznate, tada nije nužno opravdana potreba za BFT, koja također za sobom povlači i određeni pad performansi zbog veće količine podataka koja se šalje kroz mrežu.

6.4. MSP

Poslužitelj servisa članarine (dalje u tekstu eng. *Membership Service Providers* (MSP)) je komponenta sustava koju *Fabric* koristi za apstrakciju svih kriptografskih mehanizama i protokola zaduženih za izdavanje i potvrdu korisničkih certifikata, određivanje pristupa i prava korisnika, te se temelji na Infrastrukturi Javnih Ključeva (dalje u tekstu PKI) . PKI se sastoji od programa i uređaja koje osoba može koristiti za potrebe potvrde identiteta i integriteta komunikacije sa drugom osobom, preko provjerenoga tijela za izdavanje certifikata (dalje u tekstu eng. *Certificate Authority* (CA)) koji veže određeni javni ključ sa određenim identitetom izdanim u digitalno potpisanom certifikatu koji je potpisan sa privatnim ključem CA[16].

Ideja iza MSP-a je da se omogući korištenje različitih implementacija upravljanja korisničkim pravima i identitetima koji su trenutno dobro znani i imaju široku uporabu poput: *LDAP*, *OAuth*, *Kerberos*, *Active Directory* i sličnih. MSP je obavezan na razini mreže, kanala, člana, FOS, i korisnika, koristeći njegove funkcije entitet u HLF mreži MSP prepoznaje kojim CA organizacijama, članovima ili korisnicima vjeruje, te koja prava imaju.

Prepoznajemo dvije vrste MSP-a: globalni (mreža i kanali) i lokalni (članovi, korisnici, FOS).

- **MSP mreže** definira koji organizacije pripadaju mreži tako da sadrži popis MSP-ova dotičnih organizacija što indirektno definira popis svih korisnika te mreže, jer svaki korisnik mora pripadati jednoj organizaciji. Također, definira i koji korisnici imaju ovlasti administriranja mreže, poput dodavanja novih organizacija ili kanala.
- **MSP kanala** definira koje organizacije pripadaju kanalu, te koji od njihovih korisnika imaju koja prava na kanalu, primjerice dodavanje organizacija u kanal i pokretanje novih pametnih ugovora.
- **MSP člana** je lokalan, što znači da je se nalazi samo na članu i vrijedi samo za člana na kojem se nalazi, to mu je jedina konceptualna razlika od tipa MSP za kanal. Dakle,

definira organizacije odnosno korisnike koji imaju pristup ovom članu i prava koja su vezana uz ovu razinu, kao na primjer, instaliranje novog pametnog ugovora, razliku između instaliranja i pokretanja sam opisao u sekciji 6.2

- **MSP FOS-a** je vrlo sličan po dometu i funkciji MSP-u člana, definira koje organizacije odnosno korisnici ili članovi mogu podnijeti nove transakcije i sa kojim pravima.
- **MSP korisnika** omogućuje da se korisnik predstavi na mreži kao član kanala ili kao nositelj određene uloge u mreži i/ili kanalu.

Kao što je već navedeno, lokalni MSP se fizički i logički nalazi samo na sustavu člana ili korisnika za kojeg vrijedi, dok su globalni MSP-ovi raspoloživi na svim članovima na kanalu koristeći konsenzus protokol, odnosno konfiguracijske transakcije, međutim logički se nalaze na kanalu ili mreži za koju su vezani.

6.5. Politika odobravanja transakcija

Politika odobravanja transakcija se koristi kako bi opisala članovima kako odlučiti o ispravnosti određene transakcije, odnosno, da li je ispravno odobrena. One se definiraju prilikom pokretanja i/ili nadogradnje pametnog ugovora. Kada član primi transakciju, on će pozvati svoj pametni ugovor zadužen za tu transakciju, pozitivan rezultat funkcije pametnog ugovora smatra se kao odobrenje transakcije. Zatim će HLF na temelju dobivenih odobrenja i svog internog pametnog ugovora spomenutog u poglavlju 6.2 odlučiti o valjanosti transakcije ukoliko ona na kraju ciklusa odobravanja zadovoljava sljedeće uvijete:

- Sva odobrenja koja su do sata obavljena u sustavu su ispravna, odnosno potpisani su sa ispravnim i valjanim digitalnim potpisima, te sadrže pozitivne rezultate.
- Postoji zadovoljavajući broj odobrenja
- Odobrenja su došla od očekivanih izvora

Politika odobravanja može biti prilagođena potrebama aplikacije, primjeri uključuju: većina članova organizacije "A" ili barem po N članova svih organizacija u kanalu. Ova pravila moguće je kombinirati kako bi se dobila politika koja odgovara domeni aplikacije, ali također je moguće napraviti i prilagođena pravila. Primjeri kada su potrebna posebno razvijena pravila su: situacija u kojima je potrebno provjeriti da tokeni unutar transakcije nisu već iskorišteni u nekoj drugoj transakciji, te kada odobrenje ne sadrži identitet člana, ali potpis i javi ključ su dani takvi da ih nije moguće povezati sa identitetom korisnika.

Prilagođena logika odobravanja sastoji se od binarne izvršne datoteke koja potrebno je instalirati na svakog člana zasebno, u slučaju da ne postoji dotična binarna datoteka na članu koji odobrava transakciju tada će njegovo odobrenje vratiti negativan rezultat. Ovo može, u najgorem slučaju, uzrokovati zaustavljanje rada mreže ukoliko izvršne datoteke nisu instalirane na dovoljnom broju članova, ili u drugom slučaju, može dovesti do neispravnog stanja na mrežu ukoliko je kriva logika potvrđivanja instalirana na većini članova. U nekoj od budućih verzija

HLF ovaj problem će se riješiti tako da se upravljanje životnim ciklusom prilagođene logike za odobravanje transakcije pomakne u infrastrukturu kanala, no do tada, odgovornost o ispravnosti odobravajuće logike leži isključivo na administratorima mreže.

6.6. Model podataka

6.6.1. Kanali

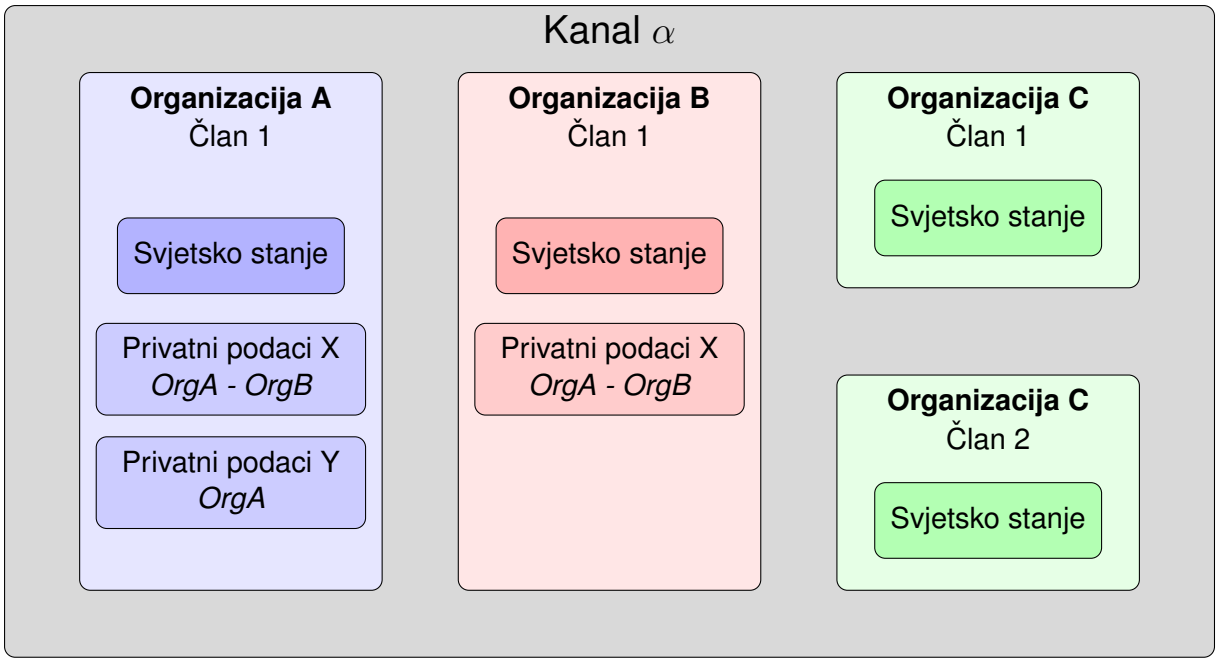
Kanali su jedan od načina podjele podataka unutar HLF koji se sastoji od dvoje ili više članova. Svaka transakcija u HLF izvodi se na jednom od kanala, te samo članovi određenog kanal mogu čitati podatke i izvoditi transakcije na tom kanalu. Svaki kanal započinje prvom konfiguracijskom transakcijom spremljenoj u prvom bloku na kanalu (eng. *genesis channel*). U konfiguracijskim transakcijama spremljeni su administracijski podaci o kanali koji uključuju popis pripadnika kanalu, te njihove ovlasti na kanalu. Postoji i poseban kanal koji se kreira prilikom samog početka HLF mreže koji se zove sistemski kanal. Ovaj kanal se koristi kako bi definirao početnu konfiguraciju čitave mreže, odnosno, tko su administratori mreže, razinu kompatibilnosti mreže, tko su članovi u mreži i slično. Svaku konfiguraciju kanala moguće je nadograditi, odnosno izmijeniti, slanjem izmjena u novoj konfiguracijskoj transakciji od strane korisnika koji imaju adekvatna prava. Prilikom pridruživanja novog člana određenom kanalu, njemu se konfiguracija kanala u obliku akumuliranih svih postojećih konfiguracijskih transakcija.

Primjer korištenja kanala bi bio primjerice da jedan trgovac trguje sa drugim trgovcima unutar granica Europske Unije unutar jednog kanal, a unutar drugog kanal trguje sa ostatkom svijeta. Na taj način je jednostavno organizirati prava pristupu transakcijama carinskoj službu unutar kanala za trgovanje sa ostatkom svijeta, bez da carinska služba uopće bude svjesna ostalih transakcija koje se odvijaju unutar drugog kanala.

6.6.2. Privatni podaci

Privatni podaci su još jedan način podjele podataka unutar HLF na način da se spremaju odvojeno od same transakcije u šifriranom obliku, te ih članovi koji im imaju pravo pristupa mogu pročitati pomoću ključa transakcije uz koju su vezani. Stoga, kanali su korisni kada je potrebno da čitave transakcije budu skrivene od određenih članova, a privatni podaci se koriste u slučaju da samo jedan dio podataka u transakciji bude dostupan određenim članovima na kanalu. Privatni podaci su i dobar način osiguravanja da podaci nikada neće doći do FOS, iako on nikad niti ne čita sam sadržaj transakcija [5]. Ovo je osigurano na način da se samo *hash* privatnih podataka sprema u samu transakciju, zbog potrebe potvrde ispravnosti podataka. Stvarni podaci spremaju se u sekundarnu bazu podataka (unutar HLF nomenklature to je eng. *SideDB*), koja se će se distribuirati svim autoriziranim članovima putem protokola ćaskanja koje je detaljnije opisan u poglavlju 6.6.4. *SideDB* je zapravo logički odvojeni dio *LevelDB* baze podataka koja se nalazi na svakom članu i služi za spremanje i ostalih podataka.

Primjer korištenja privatnih podataka je da trgovac želi držati cijenu po kojoj prodaje određena dobra ostalima tajnom, kako bi si omogućio različiti cjenik za određene klijente prema



Slika 4: Prikaz particije podataka na HLF mreži

kojima ima iz ovog ili onog razloga povoljniju ponudu. Doduše u ovom slučaju bi trebao svaku cijenu držati u privatnim podacima, jer bi neki klijenti postali sumnjičavi ako je njihova cijena javna, a nečija druga nije.

Na primjeru 4 vidimo jedan oblik particije podataka u kojem na "Kanalu α" djeluju tri organizacije, "Organizacija A", "Organizacija B" i "Organizacija C", svaka sa svojim članovima. Članovi "Organizacije A", u ovom primjeru, samo je jedan takav, imaju pristup svjetskom stanju, te "Privatnim podacima X" i "Privatnim podacima Y". Članovi "Organizacije B" imaju pristup "Privatnim podacima X" kao i "Organizacija A", ali nemaju pristup "Privatnim podacima Y", jer su oni čitljivi samo prvoj organizaciji, dok "Organizacija C" nema pristup niti jednim privatnim podacima već može čitati stanje samo sa svjetskom stanja sa svoja dva člana.

6.6.3. Dohvat podataka i bogati upiti

HLF trenutno stanje svijeta sprema u *LevelDB*, a, opcionalno, može se spremati i u *CouchDB* bazu podataka. Prijašnja je zadana odmah pri pokretanju HLF mreže, a potonju je moguće dodatno uključiti kako bi se mogli izvoditi bogati upiti (eng. *rich queries*). *LevelDB* pruža poprilično elementarne operacije nad podacima, sa njom je moguće pročitati dokument po ključu, pročitati dokumente po dometu dvaju ključeva, dakle svih dokumenata uključivo između dva ključa, te izmijeniti ili obrisati dokumente po ključu ili više njih, te unijeti nove dokumente. *CouchDB* pruža puno moćnije mogućnosti odabira podataka kao i njihovog indeksiranja, no u detalje ovoga neću ulaziti jer je to van područja ovog rada. Bitno je napomenuti da je prednosti *CounchDB*-a moguće iskoristiti samo ako pametni ugovor sprema podatke u obliku *JSON*-a, jer to je format s kojim *CouchDB* zna radit, inače HLF nije izbirljiv po pitanju formata podataka koji se sprema, jer pametni ugovori su ti koji te podatke obrađuju, tako da je moguće, primjerice, spremati podatke u binarnom zapisu, kako bi se ostvarile bolje performanse u smislu protoka

podataka ili veličine podatkovnog sloja, a u tom slučaju, naravno, moguće je dodatno implementirati unutar pametnog ugovora mogućnost zapisa podataka u neki drugi sustav koji se ne nalazi na *blockchain*-u i podržava format podataka koje aplikacija koristi. S obzirom da je *CouchDB* instanca kontejner odvojen od člana koji obrađuje transakcije; ne postoje garancija da su podaci u njoj u potpunosti ažurni, stoga se ne preporuča korištenje bogatih upita za vrijeme operacija koje rade izmjene u svjetskom stanju.

6.6.4. Protokol "ćaskanja"

Jedna od najbitnijih komponenata koja osigurava ispravan rad HLF-a kao distribuiranoga servisa je protokol "ćaskanja" (dalje u tekstu eng. *Fabric Gossip Protocol* (FGP)) . Ovaj protokol služi za razmjenu podataka o svjetskom stanju, što uključuje i privatne podatke, te podaci pripadnosti korisnika i njihovim pravima. Prva uloga ovog protokola je pružanje načina otkrivanja novih članova koji se pridruže mreži, ili otkrivanje koji od članova više nije dostupan. Ovo se odvija stalnim slanjem poruka o živosti između svih članova, s time da novi član, prilikom spajanja na mrežu, šalje poruku o živosti članu sidru (eng. *Anchor Peer*), koji mu odgovara sa popisom svih ostalih članova za koje član sidro zna. U ovom trenutku je član praktički pridružen mreži, a svi ostali članovi će doznati za njega u trenutku kada im novo-pridruženi član pošalje poruku o živosti ili kada se ostali članovi dobiju tu informaciju od člana sidra. S obzirom da je član sidro ključan u ispravnom funkcioniranju mreže, preporuča se da svaka organizacija definira više članova sidra kako bi se osigurao neprekinuti rad. Druga uloga FGP-a je propagacija novih podataka na mreži. Ovo služi tome da članovi koji nisu sudjelovali u potvrdi transakcije dobiju informaciju o tome što je transakcija promijenila u svrhu očuvanja konzistentnosti svjetskog stanja svih članova. Također, ovo služi i razmjeni privatnih podataka spremljenih u sekundarnoj bazi podataka između članova koji imaju pravo pristupa istim. Treća uloga protokola je podjela čitavog relevantnog svjetskog stanja novim članovima koji se pridruže mreži. Četvrta uloga je izbor vođe članova određene organizacije je opcionalna, u slučaju da organizacija odluči dinamički izabirati člana voditelja. Voditelja je moguće i ručno odabrati, no to nije preporučeno, jer član voditelj je odgovoran za komunikaciju sa FOS-om, te u slučaju da taj član postane nedostupan, organizacija više nije u mogućnosti slati nove transakcije.

7. Hyperledger Fabric Aplikacija

Za praktičan dio ovog rada odlučio sam napraviti, pomoću *Hyperledger Fabric* okruženja, aplikaciju za trgovanje električnom energijom za potrebe punionica električnih vozila.

U HLF aplikacijama, klijenti komuniciraju sa *blockchain*-om na dva načina: putem *Fabric SDK*-a ili putem *Fabric CLI* sučelja. Službeni i potpuno podržani *Fabric SDK* je dostupan u programskim je NodeJS i Java, a postoji i službeni, ali ne još u potpunosti podržani SDK u programskim jezicima Go i Python, a također je u razvoju i REST SDK koji ne ovisi o programskom jeziku.

Fabric SDK ima mogućnost kreiranja i upravljanja kanalima, pridruživanja novih članova na kanal, instaliranje i pokretanje pametnih ugovora sa adekvatnim pravima, te pozivanje funkcija pametnih ugovora, kao i upit u trenutno stanje. *Fabric CLI* je alat koji ima iste funkcionalnosti kao i *Fabric SDK* ali je namijenjen korištenju kroz komandnu liniju, te nije adekvatan za interakciju sa mrežom u stvarnom vremenu, ali je koristan za pokretanje sustava i odrađivanje kućanskih poslova održavanja mreže kao što je pridruživanje novih organizacija u mrežu, izradu novih kanala, pridruživanje novih korisnika ili članova i ostalo.

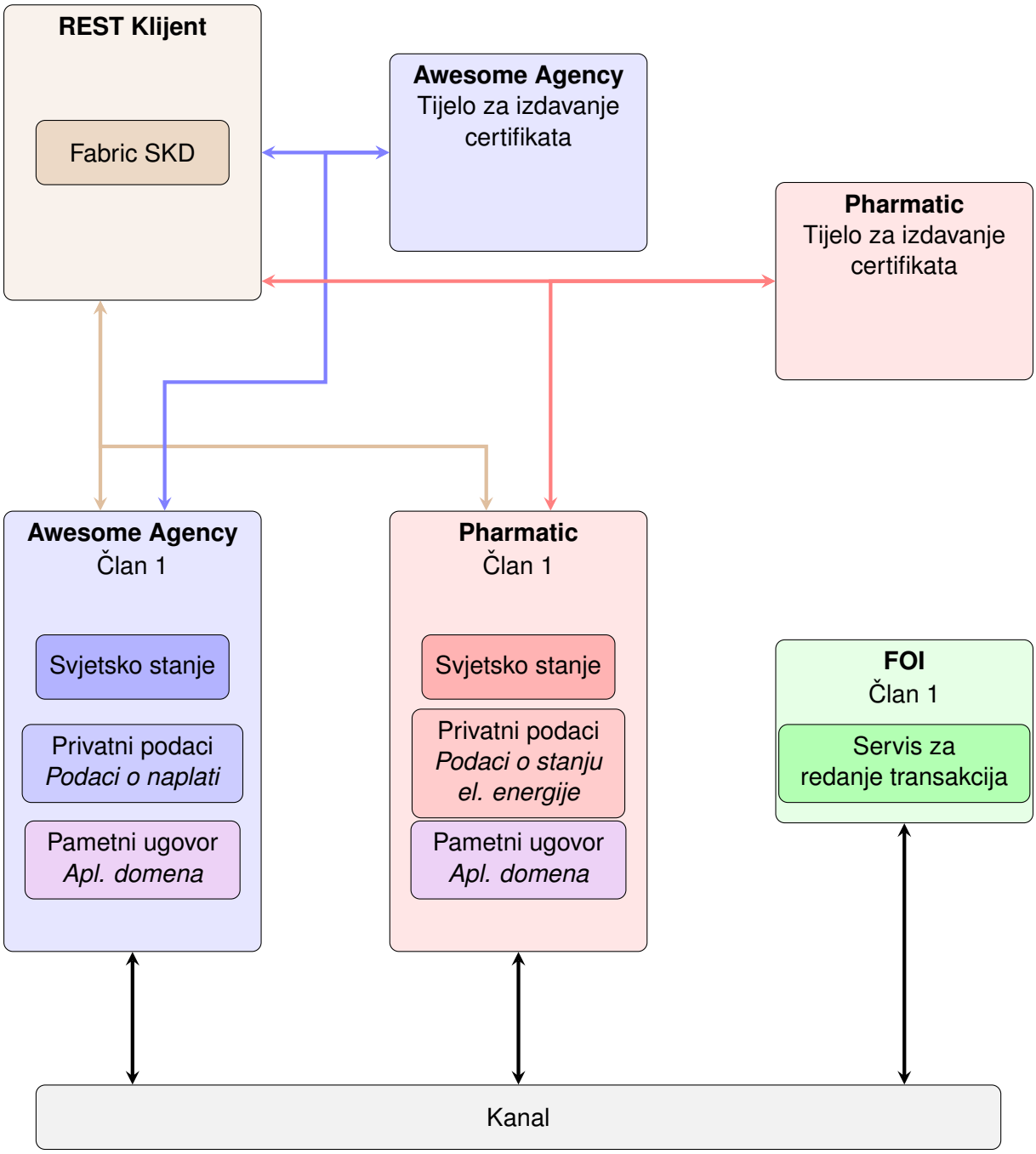
Aplikacija koristi HLF za pozadinsku logiku zaduženu za praćenje korisnika sustava, njihovu potrošnju električne energije, te evidenciju dugovanja sredstava između entiteta poslužitelja električne energije i korisnika, odnosno vozača električnih vozila. Korisničko sučelje izrađeno je u *ReactJS* okolini, koje omogućuje responzivnu interakciju sa sustavom, a pomoćna serverska aplikacija napisana je u programskom jeziku Go.

Sustav se sastoji od 3 organizacije:

- Neovisna organizacija koja upravlja servisom za redanje transakcija. Ova organizacija u primjeru se naziva FOI, a njena uloga je da na indiferentan način upravlja i održava sustavom za redanje transakcija u korist svih korisnika sustava.
- Organizacija za upravljanje korisnicima koja posjeduje povjerenje ostalih učesnika u sustavu da na ispravan način upravlja povjerljivim podacima klijenata koji im daju u uvid njihove podatke za naplatu novčanih sredstava u protuvrijednosti električne energije koje su potrošili.
- Organizacija koja upravlja punionicama kojih u sustavu može biti više, međutim u primjeru je ostavljena samo jedna. Zadaća ove organizacije je osiguravanje dostupnosti energije i ispravnog rada njihovih punionica. Klijent sustava će pomoću dotičnih uređaja u vlasništvu ove organizacije razmijeniti svoja novčana sredstva za električnu energiju.

7.1. Arhitektura

Na slici 5 prikazana je arhitektura aplikacije praktičnog primjera. Mreža je relativno jednostavnog formata, sastoji se od člana organizacije *Awesome Agency* (dalje u tekstu AA),



Slika 5: Prikaz arhitekture aplikacije

člana organizacije *Pharmatic* (dalje u tekstu P) i jednog člana koji je ujedno i FOS pod kontrolom organizacije *FOI*.

Obični korisnici pristupaju aplikaciji preko REST klijenta koji se onda spaja na poslužitelje certifikata i članove *blockchain* mreže ovisno o tome koji korisnik se spojio. S obzirom na distribuiranu prirodu aplikacije, REST klijenata može biti više i tako da budu skriveni iza raspodjeljivača opterećenja (eng. *load balancer*), i na taj način je moguće u svakom trenutku kreirati dodatne REST klijente ukoliko za to postoji potreba zbog povećane potražnje, i ako to pozadinska *blockchain* mreža može podnijeti. Proširenje pozadinske mreže, također, nije teško za napraviti, jer takva funkcionalnost je podržana u HLF, koji se brine o distribuciji podataka i osigurava njihovu konzistentnost i dostupnost diljem svih članova.

7.2. Pokretanje primjera

Prije pokretanja mreže potrebno je definirati određene informacije o tome tko pripada kojoj organizaciji i koja su njihova prava. Ovaj postupak odvija se u dva koraka. Prvi korak je izrada početnih kriptografskih materijala, odnosno, MSP-a koji se sastoji od javnih i privatnih ključeva za administratore i članove svake od prve tri organizacije. Implikacije i korištenja ovog kripto materijala detaljno su opisani u poglavlju 6.4. Kripto materijali koje želimo generirati nemaju ništa specifično za HLF osim dodatnih atributa koji se definiraju u certifikatima potrebnih za opise uloga koji člnaovi imaju na kanalu, ali zbog većeg broja ključeva koji potrebni na početku, postoji alat `cryptogen` koji će za nas napraviti većinu posla. Početna definicija se može pregledati na lokaciji `network/crypto-config.yml` u izvornom kodu primjera.

Ovo je potrebno za sastavljanje prvog bloka u prvom kanalu na mreži, koji će ujedno biti i jedini kanal u aplikaciji, osim administracijskog kanala. Početnu konfiguraciju ovog kanala generiramo sa alatom čija je ovo primarna uporaba, `configtxgen`, a taj se alat još može koristiti i za ispitivanje stanja na kanalu, organizacija na mreži i osnovnih podataka o transakcijama. Početna definicija se može pregledati na lokaciji `network/configtx.yml` u izvornom kodu primjera.

Kako bi pojednostavnio primjer izradio sam *Bash* skripte koje ovaj proces rade automatski, pa je za pokretanje mreže potrebno samo izvršiti skriptu `network/generate.sh` pod uvjetom da na računalu već postoje izvršne datoteke navedenih alata `configtxgen` i `cryptogen` u putanji (varijabla okoline PATH).

Pokretanje mreže radi se pomoću alata *Docker Compose* koji na lokalnom računalu pokreće instance kontejnera kakve su definirane unutar `network/docker-compose.yml` datoteke. Kao i za kreiranje početne konfiguracije i za ovo sam napravio skriptu koja preuzima potrebne datoteke, sastavlja mrežu te je pokreće na računalu: `network/start.sh`.

Nakon pokretanja, potrebno je još pokrenuti i skriptu `network/fixtures.sh` koja će inicirati početne podatke na mreži, kreirati korisnike i napraviti nekoliko transakcija za početak. Korisnička aplikacija, ukoliko je sve prošlo dobro, trebala bi biti dostupna na adresi *http://localhost:3000*.

Iz aplikacije moguće je, kao korisnik, započeti nove transakcije i završiti stare, te vidjeti svoje prijašnje transakcije. Kao administrator P organizacije moguće je vidjeti sve transakcije napravljene na punionicama organizacije P. Kao administrator organizacije AA moguće je dobiti sveobuhvatni prikaz sustava kao i podatke o naplati novčanih sredstava prema krajnjim korisnicima i prema ostalim organizacija koje nude svoje punionice.

7.2.1. Zaustavljanje i uklanjanje primjera

Dotični primjer može potrošiti ne trivijalnu količinu prostora na računalu zbog velikog broja slika virtualnih sustava (eng. *virtual images*), pa zato postoji i skripta koja će sve zajedno zaustaviti i ukloniti sa računala nakon izvođenja i testiranja primjera: `network/teardown.sh`.

8. Zaključak

U ovom radu sam opisao razloge postojanja *Hyperledger* ekosustava, te sam pokušao pokazati kako i zašto ova tehnologija i ova konkretna implementacija ima puno smisla za budućnost, što se posebice očituje time da već sada postoje sustavi koji koriste neke od okruženja ovdje prikazanih. Započeo sam temu sa kratkim uvodom u *blockchain*, koji je, kao tehnologija, i dalje u povojima i tek će se mora krenuti ozbiljnije koristiti, zatim sam opisao sve alate i okruženja pod *Hyperledger* krovom, te sam dao detaljnije opise *Composer* alata, te *Indy* i *Fabric* okruženja.

Potkrijepio sam svoje ideje sa dva praktična primjera, mrežom za praćenje lova, uzgoja i trgovine riba u *Composer* alatu i mrežom za razmjenu i trgovanjem električne energije za potrebe punionica električnih vozila. Smatram da oba primjera daju dobar uvid u razloge zašto bi bilo dobro koristiti *blockchain* tehnologiju u sustavima koji rješavaju ovakve probleme na razini koja se od njih očekuje.

Problem koji rješava *Indy* je jedan od najbitnijih za zdrav nastavak funkcioniranja interneta, ali i svih ostalih aspekata u današnjoj povezanoj ekonomiji, a ostale tehnologije kao što su *Burrow*, *Iroha*, *Sawtooth* i *Fabric* su, u mojim očima, jedne od budućih oslonaca načina rada svih igrača na dalje. U budućem sve više povezanom svijetu postoji iznimna potreba za rješavanjem problema privatnosti i povjerenja, a postalo je bolno očito kako individualna osoba ne može imati povjerenja u korporacije kojima daje svoje podatke na čuvanje, što kasnije uzrokuje i opći gubitak privatnosti. Riješenje još uvijek nije očito za sve probleme predstavljene u ovom radu, ali ja osobno čvrsto uvjeren da *Hyperledger* nudi put u dobrom smjeru.

Akronimi

ACL eng. *Access Control List*. 11, 12

BFT eng. *Byzantine fault tolerance*. 25, 26

BNA eng. *Business Network Archive*. 14

CA eng. *Certificate Authority*. 26

CFT eng. *Crash fault tolerance*. 25

DDO eng. *DID descriptor object*. 16, 17

DID eng. *Decentralized Identity*. 8, 16, 17, 20

DKMS eng. *Decentralized Key Management System*. 8, 16, 19

DLT eng. *Distributed Ledger Technology*. 6, 19, 21

FGP eng. *Fabric Gossip Protocol*. 30

FOS eng. *Fabric Ordering Service*. 22, 24–28, 30, 33

HLF *Hyperledger Fabric*. 21, 23–31, 33, 39

MSP eng. *Membership Service Providers*. 26, 27, 33

PKI eng. *Public Key Infrastructure*. 17, 26

PoET eng. *Proof of Elapsed Time*. 8

PoW eng. *Proof of Work*. 3

SSI eng. *Self-Sovereign Identity*. 16

TEE eng. *Trusted Execution Environment*. 8

ZKP eng. *Zero-knowladge proof*. 18, 19

Popis literature

[1] *Burrow Documentation*. Lipanj 2018. URL: https://wiki.hyperledger.org/display/burrow.

[2] *Cello Documentation*. Lipanj 2018. URL: https://hyperledger-cello.readthedocs.io/en/latest/.

[3] *Composer Documentation*. Lipanj 2018. URL: https://hyperledger.github.io/composer/latest/.

[4] *Cryptocurrency. (n.d.)* Lipanj 2018. URL: https://www.merriam-webster.com/dictionary/cryptocurrency.

[5] *Fabric Documentation*. Lipanj 2018. URL: https://hyperledger-fabric.readthedocs.io/.

[6] Linux Foundation. *Hyperledger Architecture, Volume 2*. Lipanj 2018. URL: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf.

[7] *Guided Tour of Hyperledger Fabric and Hyperledger Indy*. Srpanj 2019. URL: https://github.com/hyperledger/ursa.

[8] *Hyperledger*. Lipanj 2018. URL: https://www.hyperledger.org/.

[9] *Hyperledger Ursa repository*. Kolovoz 2019. URL: https://www.youtube.com/watch?v=qMZiMneuf1w.

[10] *Indy Documentation*. Kolovoz 2018. URL: https://wiki.hyperledger.org/projects/indy.

[11] A. Gervais K. Wüst. „Do you need a Blockchain?": *Procedia Technology* (2017).

[12] *Keynote: Introducing Hyperledger by Brian Behlendorf, Executive Director, Hyperledger Project*. Lipanj 2018. URL: https://www.youtube.com/watch?v=pr4Hb0jb0lo.

[13] Dimitry Khovratovich. „Anonymous credentials". (2016). URL: https://github.com/hyperledger-archives/indy-anoncreds/blob/master/docs/anoncred-usecase1.pdf.

[14] Georgios Konstantopoulos. „Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance". (2019). URL: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419.

[15] Intel Monax. „Burrow incubation proposal". (Ožujak 2017).

[16] *MSDN*. Srpanj 2019. URL: https://docs.microsoft.com/en-us/windows/win32/seccertenroll/public-key-infrastructure.

[17] Satoshi Nakamoto. „Bitcoin: A Peer-to-Peer Electronic Cash System”. (Listopad 2008).

[18] *Rebooting the Web of Trust IX: Prague*. Rujan 2019. URL: https://github.com/WebOfTrustInfo/rwot9-prague.

[19] Ronald J. Reisman. „Air Traffic Management Blockchain Infrastructure for Security, Authentication, and Privacy”. (2019). URL: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20190000022.pdf.

[20] *Sovrin Official Website*. Kolovoz 2018. URL: https://sovrin.org/.

Popis slika