JOLLAR

Laboratorio Sistemi Operativi A.A 2017/2018

Una realizzazione del gruppo **EdgarEboli**

Tommaso Pagni

Domenico Papandrea

Daniel Fabbrica

Mail di riferimento: domenco.papandrea@studio.unibo.it

Jollar implementa un sistema di scambio elettronico decentralizzato, dove gli utenti effettuano transazioni certificate dalla rete senza il bisogno di un organo centrale garante.
Sfruttando una struttura di rete *Peer-To-Peer* (P2P), la garanzia dell'operazione è affidata ad un algoritmo di creazione di *Proof-of-Work* (POW), ovvero una certificazione unica e digitale della registrazione delle transazioni. Gli utilizzatori dunque dovranno elaborare una serie di valori "complessi" da trovare di cui sia facile controllare la correttezza. Le transazioni vengono registrate insieme alla POW nell'archivio, pubblico e noto a tutti gli utilizzatori, detto Blockchain.

Come già anticipato nel paragrafo precedente, il sistema si basa su un'architettura di rete *Peer-To-Peer*, con un numero arbitrario di nodi, comprendente anche un server dedicato ad il solo coordinamento temporale, che fornisce l'ora esatta ai vari nodi in rete, e ed un *Network Visualiser*, il quale consente di monitorare lo stato dei nodi, visualizzare le transazioni eseguite da ciascuno di esso, sia in entrata che in uscita, e la blockchain del sistema.

CONCETTI DEL JOLLAR

LA BLOCKCHAIN

La *blockchain* è una struttura dati costituita da un insieme di **blocchi** concatenati contenenti le transazioni validate dalla rete. Ogni **blocco** possiede le informazioni relative alla sua posizione nella blockchain e alla *POW* che è stata elaborata per validare lo stesso.
Abbiamo sfruttato la possibilità di creare variabili ad albero presente in *Jolie* per organizzare al meglio i dati salvati.

La BlockChain viene salvata all'interno di '*global.blockchain*' da ogni nodo.

Block è composto da:

.hash, il nome del blocco;
.**prevHash**, il nome del blocco nella posizione precedente al blocco in esame;
.requester, la *location* del nodo che richiede l'inserimento del blocco nella blockchain;
.indexOf, la posizione in indice nella blockchain del blocco;
.chain*, è l'array che contiene la catena di numeri primi necessaria per elaborare la *POW;*
.time, l'orario in *ms* cui viene creato il blocco;
.difficulty, valore di controllo per elaborare la *POW;*
.transaction*, insieme di transazioni inserite nel blocco.

```
type Block: any{
    .hash:string
    .prevHash?: any
    .requester?:string
    .indexOf:int
    .chain*: any
    .time:RichiestaMs
    .difficulty: any
    .transaction[0,10]: TransType
}
```

FIG. 1 – STRUTTURA DI UN *BLOCCO*

Il carattere *?* in figura(1) indica un dato facoltativo, il carattere * indica un array di dimensione [0,n].

Ogni blocco che viene iscritto al registro prevede una ricompensa di 6 *Jollar* per chi elabora la POW che certifica l'autenticità dello stesso. Il *Reward* viene iscritto nel blocco come una speciale transazione tra il sistema, identificato come "REWARD", e il nodo incaricato.

LE TRANSAZIONI

Una transazione è la parte di blocco in cui si specifica lo scambio dei jollar tra due entità della rete. Andiamo ad analizzare il prototipo di una di esse:

.publicKeySeller*, un array che contiene la chiave pubblica del venditore;
.publicKeyBuyer*, un array che contiene la chiave pubblica del compratore;
.jollar, il numero di jollar dell'operazione;
.peerID?, la *location* del richiedente l'operazione;
.transID?, l'identificativo della transazione;
.signature?, la firma che il venditore applica per confermare la cessione dei jollar.

```
type TransType: void{
    .publicKeySeller*:undefined
    .publicKeyBuyer*:undefined
    .jollar:any
    .peerID?:string
    .transID?:string
    .signature?: string
}
```

FIG. 2 – STRUTTURA DI UNA TRANSAZIONE

I partecipanti allo scambio si identificano con la chiave pubblica in loro possesso, mentre la chiave privata viene utilizzata per firmare l'***hash*** della transazione in modo da certificare il possesso della somma inserita. Infatti, per poter inviare dei *Jollar* è necessario che la chiave pubblica che vogliamo utilizzare sia già presente come beneficiaria di un'altra transazione.

HASH

Gli ***hash*** presenti nei blocchi e nelle transazioni sono un ulteriore strumento di sicurezza fondamentale per garantire le informazioni contenute. Essi traducono una qualsiasi stringa di lunghezza arbitraria in una stringa di lunghezza fissa, nel nostro caso traducono le informazioni relative al blocco o alla transazione in una stringa di 16 caratteri. In Jolie la funzione di Hash MD5 (da 128 bit) si trova già implementata nell'interfaccia MessageDigest.

Ad esempio, per calcolare l'hash di un blocco concateniamo in una variabile tutte le informazioni presenti nel blocco precedute da un valore di controllo "a". Confrontare l'hash scritto nel blocco con uno calcolato *adhoc* può indicare la possibilità di manomissione del blocco e quindi invalidarlo.

PROOF OF WORK

La proof of work è composta da due fattori: la *catena* e la *difficoltà*.
La catena è una successione di Cunningham di primo tipo, ovvero una serie, con almeno due valori, di numeri primi correlati da una funzione.
La difficoltà è il quoziente tra il resto del quoziente tra il primo numero non primo e il primo numero primo della catena, pk invece è il nostro ultimo valore.
Essendo i numeri primi degli elementi difficili da individuare ma facili da convalidare rappresentano una soluzione elegante al problema della sicurezza dei blocchi.
Ogni blocco deve avere una catena con difficoltà maggiore rispetto al precedente, così da rendere sempre più difficile individuare la pow blocco dopo blocco.

COMPONENTI DEL JOLLAR

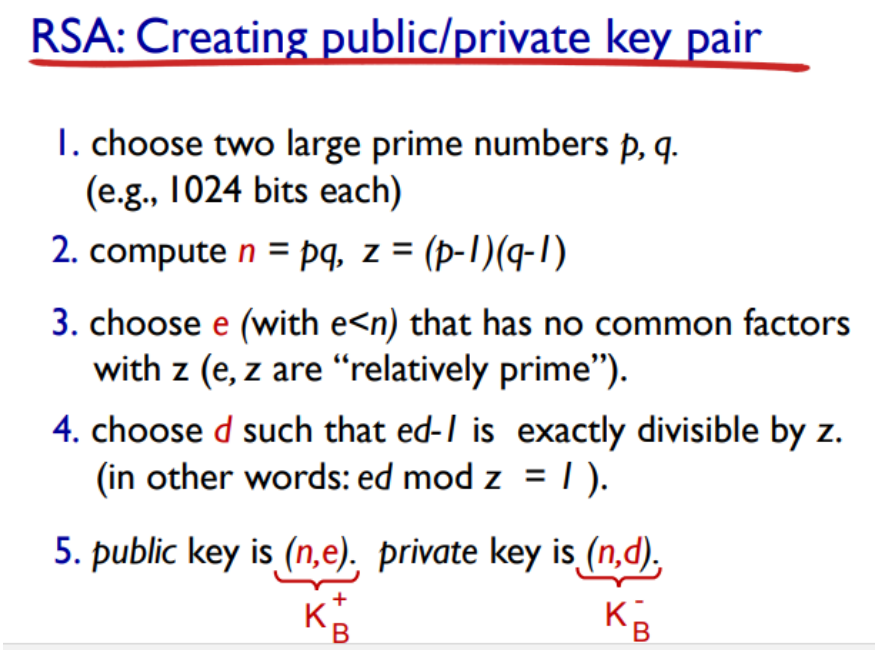
Il componente principale della rete è il *Peer*, ovvero il nodo, che compie tutte le elaborazioni sulla blockchain, gestisce le transazioni, crea i blocchi, li valida e li inserisce nella struttura dati.

INIZIALIZZAZIONE DEL NODO:

L'inizializzazione del nodo consiste nella creazione della chiave pubblica e privata dello stesso. Ogni nodo all'avvio crea la propria sfruttando il *define* ***createKeys.***

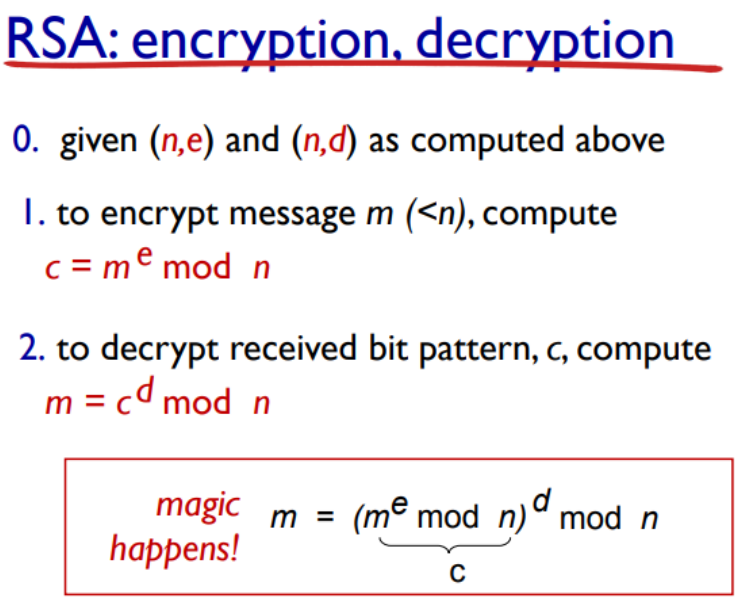
Con il *define* ***isPrime*** calcolo un numero primo random da poter utilizzare come p0 (primo numero della nostra catena).

Per ottenere e calcolare le chiavi pubbliche e private, abbiamo utilizzato l'algoritmo di crittografia assimetrica, chiamato RSA.



L'inizializzazione del nodo consiste nella creazione della chiave pubblica e privata necessarie per eseguire transazioni e quindi poter interagire sulla rete. Ogni nodo all'avvio crea la propria sfruttando il metodo createKeys, questo sfrutta l'algoritmo di crittografia asimmetrica RSA per calcolare le chiavi e "firmare" le transazioni.
La firma delle transazioni consiste nel creare un metodo che con la funzione di Hash MD5 già implementata in Jolie, facciamo l'hash della chiave pubblica del mittente + chiave pubblica destinatario + chiave privata del mittente + il numero di jollar. Con il metodo isPrime ricerco due numeri primi(1.), . Per ottenere e calcolare le chiavi pubbliche e private, abbiamo utilizzato l'algoritmo di crittografia asimmetrica, chiamato RSA. Abbiamo implementato questo algoritmo all'interno del metodo createKeys, con il quale calcoliamo i parametri delle chiavi pubblica (composta dai valori n , e) e privata (composta da n , d). Effettuata la creazione delle chiavi, il nodo entra nella rete contattando per la prima volta il servizio Broadcaster, richiedendo il download della blockchain e comunicando la propria

Location e la chiave pubblica.
Se non esiste una blockchain in rete il nodo si occuperà di creare il primo blocco senza transazioni, aggiudicandosi i primi 6 jollar previsti come premio per l'elaborazione.Con il download della blockchain o l'eventuale creazione del primo blocco si conclude 'avvio del nodo nodo.



Abbiamo implementato questo algoritmo all'interno del metodo ***createKeys***, con il quale calcoliamo i parametri delle public key(n,e) e private key(n,d).

Effettuata la creazione delle chiavi, il nodo entra nella rete contattando per la prima volta il servizio Broadcaster richiedendo il download della blockchain e comunicandogli la propria *Location* e la chiave pubblica.

Se non esiste una blockchain in rete il nodo si occuperà di creare il primo blocco senza transazioni, aggiudicandosi i primi 6 *jollar* previsti come premio per l'elaborazione.
Con il download della blockchain o l'eventuale creazione del primo blocco si conclude l'avvio del nodo.

BROADCASTER

Broadcaster è l'insieme dei servizi utilizzati per connettere tra loro i vari *Peer*. Esso viene embeddato nel nodo per poter dare al sistema la possibilità di amministrare autonomamente le comunicazioni tra le parti; inoltre raccoglie le validazioni dei vari passaggi in modo da evitare l'invio di informazioni non garantite dalla rete.

I suoi servizi sono:

- **peerOnline(myInfo)(bk)** , è il servizio che "avvia" il broadcaster. Vengono qui recuperate e salvate nel log di accesso le informazioni inerenti alla

location dei peer. Inoltre, richiede la blockchain agli altri nodi in rete per restituirla come risposta di questo servizio, se presente.

- **newTrans(transType)** , è il servizio di inoltro delle transazioni che vengono effettuate. Sfrutta il log di accesso creato con *peerOnline* per trasmettere a tutti i nodi la transazione da inserire in un nuovo blocco.

```
[validAsk(newBlock)]{

    getSplitResult;
    acquire@SemaphoreUtils(semaforoC)(res);
    //inizializzo variabili
    undef( a );
    b=0;
    undef( valid );
    undef( notValid );
    valid=1;

    for ( j = 0, j <#SplitResult.result, j++ ) {

        Antenna.location =SplitResult.result[j];
        scope (ask_Scope){
            install(IOException => {println@Console("Richiesta validazione blocco fallita n."+a+" per: "+ Antenna.location)(); a++});

            if(Antenna.location != Server_location){

                validation@Antenna(newBlock)(isValid);

                println@Console( " Invio richiesta validazione a " + Antenna.location + " che risponde " + isValid )();
                if( isValid == true  ) {
                    valid++
                }else if( isValid==false ) {
                    notValid++
                }
            }else{
                println@Console( "NON CHIEDO LA VALIDAZIONE A ME STESSO!" )()
            }

        }

    };
```

- **ValidAsk (newBlock)**, è il servizio di validazione di un blocco(vedi figura sopra). Una volta creato, esso deve essere convalidato da almeno il 50%+1 della rete per essere inserito nella blockchain. Broadcaster si occupa di inoltrare ai peer il potenziale blocco, di recuperare i risultati della validazione e di comunicare la scrittura dello stesso se positiva.

```
    nOnline=#SplitResult.result - a;
    println@Console( a + ": Invii falliti , "+#SplitResult.result+ " nodi totali presenti nella lista" )();
    soglia=(nOnline/2)+(nOnline%2);
    undef( SplitResult );
    getSplitResult;

    if( valid>=soglia ) {

        for ( i=0, i<#SplitResult.result, i++ ) {
            //newBlock.newToken=csets.newToken=new;
            Antenna.location =SplitResult.result[i];
            println@Console( "Invio writeBlock n."+ i +" a: " + Antenna.location  )();
            install( IOException => println@Console( "ioex a "+ Antenna.location )() );
            acquire@SemaphoreUtils(semaforoE)(res);
            writeBlock@Antenna(newBlock);
            release@SemaphoreUtils(semaforoE)(res)

        }

    }else{
        println@Console(" Blocco non validato.")()
    };
release@SemaphoreUtils(semaforoC)(res)
}
```

Broadcaster, come Peer, ha anche il define *getSplitResult* che va a recuperare dal file del log degli accessi, le location dei vari peer.

FLOW DEL SISTEMA:

Abbiamo deciso per la demo di automatizzare le transazioni e quindi la creazione dei blocchi.
Ogni volta che un peer si connette alla rete, richiede ai nodi presenti nella rete la BlockChain. Nel caso in cui nessuno dovesse rispondere, il nodo procede alla creazione del primo blocco con conseguente reward di 6 jollar.
Successivamente innescherà un procedimento di controllo del portafoglio e, se disponibile un saldo qualsiasi, effettuerà una donazione di un numero random di *Jollar* ad un nodo della rete in modo da favorire più elaborazioni concorrenti.

```
[askChain(ask)(resChain){
    //semaforo in entrata
    global.pplOnline<<ask;
    println@Console( global.pplOnline.n + " join the room")();
    //release semaforo
    if( is_defined( global.blockchain.block.hash ) ) {
        resChain<<global.blockchain;
        resChain=true
    }else{
        resChain=false
    }
}]{
    checkMaCash;
    if( saldo>0 ) {
        charity
        }

    }
```

FIG. 3 – TRIGGER DELLE TRANSAZIONI

Come da figura 3, un peer che elabora la *Request-Response* di richiesta della blockchain, in seguito controlla le proprie finanze con il define *checkMaCash* che restituisce il saldo recuperato dalla blockchain. Se positivo viene innescato il metodo *charity* per la donazione dei *Jollar.*

In quest'ultimo (Fig. 4) viene calcolato in modo random il numero di *Jollar* e il destinatario della donazione, preparata la struttura dati e inviata al Broadcaster.

```
define charity{
        acquire@SemaphoreUtils(global.semaforoA)(res);

        checkMaCash;
        random@Math()(nRandom);
        elemosina= nRandom * saldo;
        if( elemosina>1 ) {
            transType.jollar=int(elemosina)
        }else{
            transType.jollar=1
        };
        with( transType ){
            .publicKeySeller[0]=global.publicKey[n];
            .publicKeySeller[1]=global.publicKey[e];
            .publicKeyBuyer[0]=global.pplOnline[#global.pplOnline-1].n;
            .publicKeyBuyer[1]=global.pplOnline[#global.pplOnline-1].e
        };
        release@SemaphoreUtils(global.semaforoA)(res);
        newTrans@BroadService(transType)
}
```

FIG. 4 – TRANSAZIONE IN CHARITY

Tutti i peer riceveranno la nuova transazione con il servizio ***incomingTrans***, che si occupa di inserirla in coda e di avviare la creazione del blocco nel metodo ***createBlock***.

Il define (Fig. 5) *createBlock* va a comporre tutte le informazioni necessarie al blocco e alla sua validazione.
Inizialmente comincia la ricerca della catena di numeri primi necessaria alla creazione della POW, viene calcolata la difficoltà e aggiornato il valore di controllo per il prossimo blocco, così da non dover ricontrollare catene già spese.
Quindi verranno inserite varie informazioni:
- l'indice in cui dovrebbe posizionarsi il blocco
-le transazioni che sono in coda nella struttura ***global.w8trans***
-la ricompensa per l'elaborazione della POW
-l'orario di creazione del blocco (con una richiesta al server dedicato *SaaS*)
-l'***hash*** del blocco precedente
- l'***hash*** calcolato tramite la concatenazione di tutte le informazioni precedenti

```
define createBlock{
    acquire@SemaphoreUtils(global.semaforoC)(res);
    catena;
    for ( i=0, i<#result.chain, i++ ) {
        newBlock.chain[i]=result.chain[i] // Eseguo la POW e la inserisco
    };
    newBlock.difficulty=difficulty; //Inserisco la difficoltà

    global.check=result.chain[0] + 1; //aggiorno il valore di check per la prossima esecuzione del metodo catena

    undef(result.chain);//resetto catena per il prossimo blocco
              println@Console( "Creo il blocco numero : "+ #global.blockchain.block )();
    newBlock.indexOf=#global.blockchain.block; //Setto l'indice del blocco

    synchronized( tokenTrans ){
          //copio transazioni
        foreach ( child : global.w8trans ) {
            for ( l=0, l<#global.w8trans, l++) {
                  newBlock.transaction[indexTr].(child)=global.w8trans[l].(child);
                  if( child == "publicKeyBuyer" ) {
                    newBlock.transaction[indexTr].(child)[1]=global.w8trans[l].(child)[1]
                  };
                  if( child == "publicKeySeller" ) {
                    newBlock.transaction[indexTr].(child)[1]=global.w8trans[l].(child)[1]

                  }
                }
        };
        //undef del contenitore provvisiorio di transazioni
        undef( global.w8trans )
    };
```

FIG. 5 – METODO CREATEBLOCK

Completate queste operazioni il nodo invia la richiesta di validazione al Broadcaster che si occuperà di contattare tutti i nodi online.
La richiesta di approvazione arriva al peer con il servizio ***validation(aBlock)(isValid)*** (figura 6)***.***
Questo controlla se il blocco in arrivo concorda con la blockchain in suopossesso, controllando gli orari di creazione, le transazioni, e gli hash contenuti.
Il nodo risponderà al Broadcaster *true* o *false*, in base alle verifiche effettuate. Se più della metà della rete *valida* il blocco, esso viene inviato per la scrittura nella blockchain tramite il servizio **validAsk(newBlock)** (Fig. 7) dedicato all'aggiornamento della blockchain.

```
}
[validAsk(newBlock)]{

    println@Console( "Richiesta di validazione del blocco in posizione " + newBlock.indexOf  )();

    getSplitResult;
    acquire@SemaphoreUtils(semaforoC)(res);
    //inizializzo variabili
    undef( a );
    b=0;
    undef( valid );
    undef( notValid );

    valid=1;
    for ( j = 0, j <#SplitResult.result, j++ ) {


        Antenna.location =SplitResult.result[j];

        scope (ask_Scope){
            install(IOException => {println@Console("Richiesta validazione blocco fallita n."+a+" per: "+ Antenna.location)(); a++});

            if(Antenna.location != Server_Location){
                // println@Console( "BLOCCO RICHIESTO DA: " + newBlock.requester +  "    IO SONO: "+ Server_location)();
                acquire@SemaphoreUtils(semaforoE)(res);
                // println@Console( "VALID ASK
                //                 newBlock.rewStr " + newBlock.rewStr +
                //                " newBlock.hash " + newBlock.hash)();

                validation@Antenna(newBlock)(isValid);
                release@SemaphoreUtils(semaforoE)(res);
                println@Console( " Invio richiesta validazione a " + Antenna.location + " che risponde " + isValid )();
                if( isValid == true  ) {
                    valid++
                }else if( isValid==false ) {
                    notValid++
                }
            }else{
                println@Console( "NON CHIEDO LA VALIDAZIONE A ME STESSO!" )()
            }

        };
        release@SemaphoreUtils(semaforoC)(res)
```

FIG. 6 – VALIDAZIONE BLOCCO

La scrittura in blockchain conferma e garantisce la transazione scritta all'interno del blocco, essendo stata verificata dal 50%+1 della rete, ed essendo pubblica facilmente consultabile da chiunque.

NETWORK VISUALIZER

Il network visualizer è un'entità di osservazione della rete che serve a monitorare lo stato dei nodi e della blockchain, permettendomi di stampare a video le informazioni relative.

```
define versioneBK{
    getSplitResult;
    global.best = 0;

    for ( j = 0, j <#SplitResult.result, j++ ) {
        Antenna.location = SplitResult.result[j];
        scope (ask_Scope){
            install(IOException =>  {println@Console("CHECK VERSIONE -> " + Antenna.location + " non raggiungibile ")();
                                     global.version[j] = "NULL";
                                     global.version[j].owner = Antenna.location;
                                     global.app = 0
                                    });

            askVersion@Antenna()(versione);
            global.app = versione.num;
            global.version[j] = versione.num;
            global.version[j].owner= versione.owner;
            global.version[j].bk << versione.bk;

            if(global.app > global.best){
                global.best = global.app;
                global.best.owner = versione.owner;
                global.best.bk << versione.bk
```

Tramite input presi da tastiera si è in grado di effettuare delle scelte.
La prima scelta, *InfoNodi*, richiama alcuni per poi stampare a video tutti i Peer presenti in rete e la situazione delle transazioni ad essi legate.

La seconda scelta, *BlockChain*, a sua volta tramite l'utilizzo di alcuni define, stamperà a video la BlockChain con la versione più aggiornata presente nella rete.

Nella figura a destra possiamo, ad esempio, troviamo il define che permette al NetVisualizer di sapere quale nodo ha la BlockChain più aggiornata del sistema.

Esempio:

```
HURRY! Type something
ADMIN
  ----- WELCOME ADMIN -----

                                        1  -  Info Nodi
                                        2  -  BlockChain
1
infoNodi 13h 21m 4s

Nodo: socket://localhost:8001   chiave: (20777 , 413) versione: 1

<--- in entrata nel blocco 0 con hash 7b8abef36351dc40d2b268a4917db72a :6 jollar ricevuti da :REWARD,
Numero jollar in possesso: 6

----- Jollar totali sulla rete: 6 -----
```

TIMESTAMP

Il timestamp è un servizio, contenuto nel file Saas.ol, che riceve richieste per conoscere il tempo.

```
main
{
  [timeRequest()(tempo){

    getCurrentTimeMillis@Time()(ms);
    tempo = ms;
    println@Console( ms )()

  }]
```

Le varie componenti del sistema si rivolgono al Timestamp mediante delle RequestResponde che restituiscono il tempo in millisecondi.

L' operation implementata è:

. timeRequest()(tempo): che tramite la funzione getCurrentMillis()() dell'interfaccia Time.iol restituisce il tempo in millisecondi o in formato data-ora.

ISTRUZIONI PER LA DEMO

All'interno di una shell dopo essersi spostati nel path corretto della cartella del progetto, tramite le istruzioni a riga di comando:

1. Avviare Saas -> *jolie Saas.ol*

2. Avviare il NetVisualizer -> *jolie NetVisualizer.ol*

3. Avviare i 4 peer:
   jolie -C "Server_location=\"socket://localhost:8001\"" peer.ol
   jolie -C "Server_location=\"socket://localhost:8002\"" peer.ol
   jolie -C "Server_location=\"socket://localhost:8003\"" peer.ol
   jolie -C "Server_location=\"socket://localhost:8004\"" peer.ol

   P.s. Il parametro "-C" serve per cambiare le costanti all'interno del file