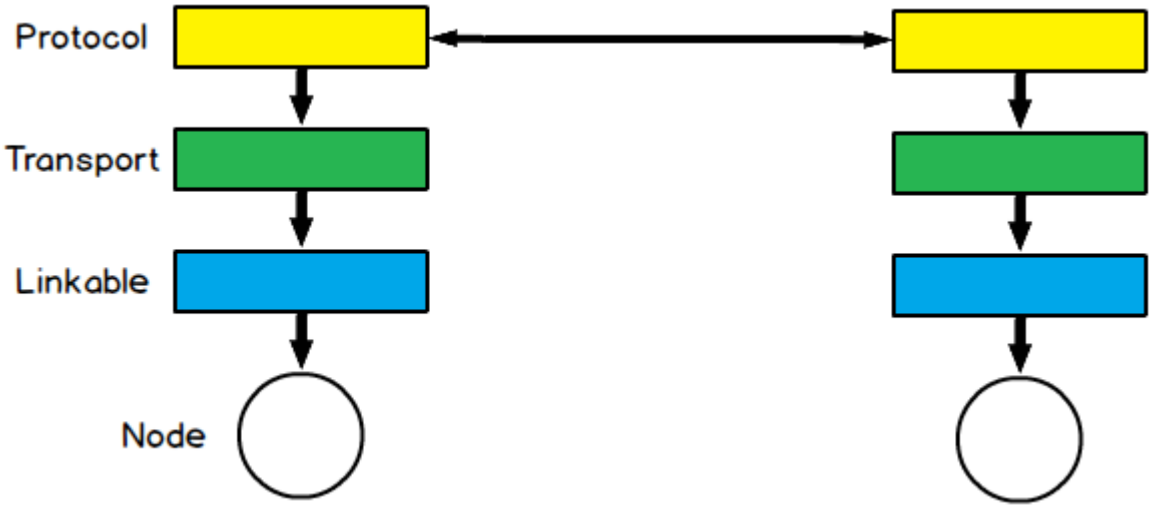
Peersim - How to use it

I sistemi peer-to-peer (P2P) possono essere estremamente grandi (milioni di nodi). I nodi si uniscono e lasciano la rete continuamente. Sperimentare con un protocollo in un tale contesto non è affatto un compito facile.
PeerSim è stato sviluppato per far fronte a queste problematiche e quindi raggiungere un'estrema scalabilità e supportare il dinamismo. Inoltre, la struttura del simulatore si basa su componenti e semplifica la rapida prototipazione di un protocollo, combinando diversi blocchi predefiniti, che sono appunto oggetti Java.

L'ovvio vantaggio dell'utilizzo di Peersim è rappresentato dal dover imparare a programmare due sole tipologie di componenti (di classi Java) e per tutto il resto potersi appoggiare su un sistema già funzionante, di cui serve capire le meccaniche di base, piuttosto che dover scrivere tutto da zero.

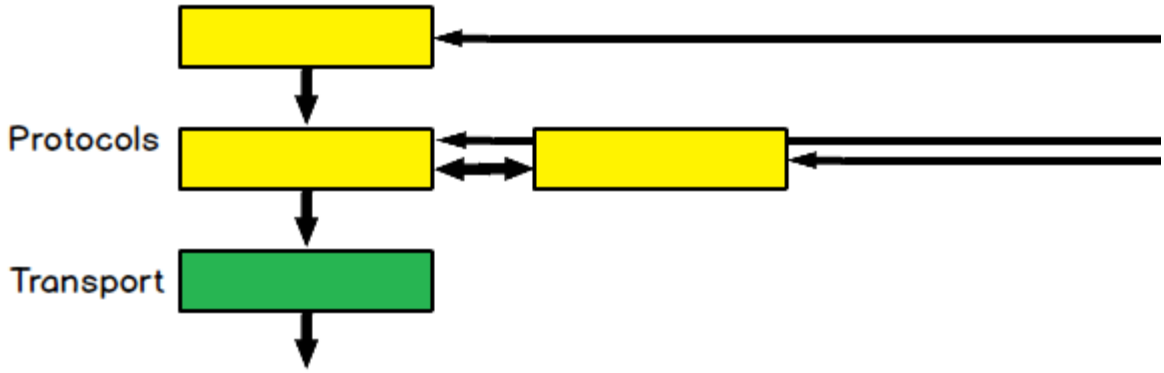
Questo tutorial si pone lo scopo di fornire informazioni utili per progettare e realizzare simulazioni personalizzate, in base alle esigenze dell'utente, risparmiando così la lettura del codice sorgente del programma altrimenti necessaria.
In particolare sarà necessario capire come scrivere le classi utili per la creazione di un proprio protocollo personalizzato, senza avere profonda comprensione di tutti i componenti.
Verrà usato un approccio descrittivo senza mostrare esempi specifici, ma piuttosto illustrando in generale le dinamiche di funzionamento del simulatore nel suo complesso.



componenti interni ai nodi

Per la quasi totalità delle simulazioni personalizzate sarà sufficiente imparare a scrivere due sole tipologie di componenti: i *Protocol*, classi che implementano l'interfaccia *Protocol*, e i *Control*, classi che implementano l'interfaccia *Control*.
La prima tipologia chiaramente tratta di protocolli, quello/i su cui si vuole sperimentare e quelli ausiliari ad esso, mentre la seconda di controllori che osservano i protocolli o/e agiscono modificando lo stato di questi ultimi.

La differenza fondamentale sta nel modo in cui essi vedono la rete; i *Control* vedono tutti i nodi e la rete nella sua totalità, quindi hanno accesso a tutte le informazioni di quest'ultima; i *Protocol* sono contenuti in ogni nodo, possono agire sul nodo stesso e sui soli nodi vicini di cui hanno visione.



vari livelli di Protocol

I *Protocol* inoltre, dentro il singolo nodo, possono essere disposti su più livelli, formando una pila, in modo del tutto analogo a quella della rappresentazione ISO/OSI o TCP/IP per esempio.
Vi è anche la possibilità di mettere più protocolli, in parallelo, nello stesso livello.
Ogni istanza di un protocollo contenuta in un nodo avrà accesso ai dati e alle risorse delle istanze degli altri protocolli di livello minore o uguale, contenuti nel nodo stesso, e delle istanze dello stesso protocollo contenute negli altri nodi.

Molto utile è dare una lettura ai *Protocol* e ai *Control* già esistenti, da un lato per poter osservare le best practice usate nella scrittura di questi ultimi, dall'altro per evitare di dover scrivere da zero classi che in alternativa si potrebbero ottenere estendendo o prendendo spunto da quelle esistenti.

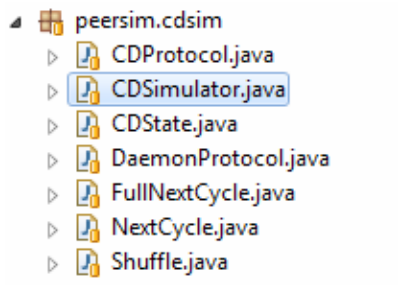
Entrambe le tipologie di componenti hanno due caratteristiche importanti.
La prima, fondamentale, è la periodicità con cui si attivano, ad ogni componente ne può essere assegnata una differente.
La seconda riguarda con quali componenti sono collegati (quindi per i *Protocol* la scelta del livello). Implica a quali dati e risorse di altri componenti hanno accesso, quindi cosa possono modificare e cosa possono sfruttare. Da ricordare che il collegamento è unidirezionale.

Il poter gestire velocemente e semplicemente queste due caratteristiche, a livello di codice, è uno dei punti di forza di Peersim, che ne evidenzia la sua versatilità.
Altra cosa importante da sapere è che le istanze di componenti non possono essere generate dall'utente nel suo codice altrimenti il simulatore non ne gestirà il funzionamento.

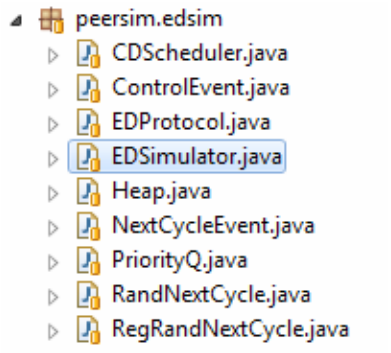
Per osservare l'andamento della simulazione si utilizzano *Control* specifici per questo scopo, che hanno quindi la caratteristica di observer. Essi possono essere usati per stampare dati anche su file.

PeerSim supporta due modelli di simulazione: il modello event-based e il modello cycle-based. Quest'ultimo modello è semplificato, il che rende possibile raggiungere un'estrema scalabilità e prestazioni, al costo di alcune perdite di realismo. Tuttavia molti protocolli semplici possono tollerare questa perdita senza problemi.
Le ipotesi semplificative del modello cycle-based sono la mancanza di simulazione del livello di trasporto e la mancanza di concorrenza. In altre parole, i nodi comunicano direttamente tra loro, e ai nodi viene dato il controllo periodicamente, in ordine sequenziale, in modo che possano eseguire azioni arbitrarie. Da notare che è relativamente facile migrare qualsiasi simulazione cycle-based al motore event-based.

cycle-based



event-based



La differenza fondamentale tra i due modelli è la diversa visione del tempo; in cycle-based viene dato in “cicli di esecuzione” mentre in event-based in valori di tempo, quantità di “unità di tempo”. Una volta scelta l’unità di misura del tempo (quella di *simulation.endtime*) si dovrà rimanere coerenti con essa ogni volta che si avrà necessità di inserire o manipolare un valore di tempo.

PeerSim è stato progettato per incoraggiare la programmazione modulare basata su oggetti. Ogni blocco è facilmente sostituibile da un altro componente che implementa la stessa interfaccia (vale a dire, la stessa funzionalità). L'idea generale del modello di simulazione è:

- scegliere la dimensione della rete (numero di nodi)
- scegliere uno o più protocolli da sperimentare, e inizializzarli
- scegliere uno o più oggetti *Control* per monitorare le proprietà a cui si è interessati e modificare alcuni parametri durante la simulazione (ad esempio, la dimensione della rete, lo stato interno dei protocolli, ecc.)
- eseguire la simulazione invocando la classe Simulator con un file di configurazione, che contiene le informazioni di cui sopra

Il ciclo di vita di una simulazione è il seguente. Il primo passo è leggere il file di configurazione, dato come parametro da riga di comando o dall’Editor. La configurazione contiene tutti i parametri di simulazione relativi a tutti gli oggetti coinvolti nell'esperimento.

Quindi il simulatore imposta la rete inizializzando i nodi nella rete e i protocolli in essi contenuti. Ogni nodo ha gli stessi tipi di protocolli; cioè le istanze di un protocollo formano un array nella rete, con un'istanza in ciascun nodo. Le istanze dei nodi e dei protocolli sono create tramite clonazione. Viene generata una sola istanza usando il costruttore dell'oggetto, che funge da prototipo, e tutti i nodi della rete sono clonati da questo prototipo. Per questo motivo, è molto importante prestare attenzione durante l’implementazione del metodo di clonazione dei protocolli.
Se nei nodi e nei protocolli sono presenti variabili di tipi non primitivi va fatto con deep clone.

A questo punto, è necessario eseguire l'inizializzazione, che imposta gli stati iniziali di ciascun protocollo. La fase di inizializzazione viene eseguita dai *Control* pianificati per l'esecuzione solo all'inizio di ogni esperimento. Nel file di configurazione i componenti di inizializzazione sono facilmente riconoscibili dal prefisso init. Questi oggetti initializer sono semplicemente *Control*, configurati per l'esecuzione nella fase di inizializzazione.

Dopo l'inizializzazione il motore richiama tutti i componenti *Protocol* e *Control* una volta in ogni ciclo, fino a un determinato numero di cicli o fino a quando un componente decide di terminare la simulazione. In Peersim tutti gli oggetti *Protocol* e *Control* sono assegnati a un oggetto Scheduler che definisce quando verranno eseguiti esattamente. Per impostazione predefinita, tutti gli oggetti vengono eseguiti in ogni ciclo. Tuttavia è possibile configurare un protocollo o un controllo per l'esecuzione solo in determinati cicli ed è anche possibile scegliere l'ordine di esecuzione dei componenti all'interno di ciascun ciclo.

Nel modello event-based tutto funziona esattamente nello stesso modo del modello cycle-based, eccetto la gestione del tempo e il modo in cui ai protocolli viene passato il controllo. *Protocols* che non sono eseguibili (usati solo per memorizzare dati) possono essere applicati e inizializzati esattamente nello stesso modo. È possibile utilizzare anche i *Controls* di qualsiasi package esterno al package *peersim.cdsim*. Per impostazione predefinita, i *Controls* vengono chiamati in ogni ciclo nel modello cycle-based. Nel modello event-based devono essere schedulati esplicitamente, poiché non ci sono cicli.

Scelta la visione del tempo che è più congeniale, si deve stabilire quanti componenti servono per gestire il sistema che vogliamo realizzare.
Per ogni componente va decisa la periodicità e i collegamenti con gli altri componenti.

La periodicità, come detto in precedenza, nel modello cycle-based ha la granularità del ciclo, mente nel modello event-based è espressa in unità di tempo.
Questo secondo modello quindi ci permette di ottenere maggior realismo da questo punto di vista.
Per questo si consiglia allo sviluppatore di scrivere i tempi utilizzando una convenzione che renda palese l'unità di misura temporale scelta anche per gli altri lettori del codice.
Per esempio specificando per la durata della simulazione 60*60*1000, allora l'interpretazione naturale è che l'unità di misura sia in millisecondi. Si ottiene così una simulazione di durata di 1 ora, anche se si avrebbe lo stesso risultato scrivendo 1 e tenendo tutti gli altri tempi in ore. Sarebbe del tutto analogo, ma l'unità di misura risulterebbe sicuramente molto meno "visibile".
Il numero di unità di tempo indicanti la durata totale della simulazione va specificato tramite il file di configurazione, in simulation.endtime
Anche la periodicità di ogni componente è specificata nel file di configurazione
per i *Protocols* in protocol.NOME.step e per i *Controls* in control.NOME.step
e indica ogni quante unità di tempo il componente si attiverà.
Inoltre per ogni componente può essere indicato un tempo di inizio, differente dal tempo 0, e un tempo di fine esecuzione, differente da simulation.endtime indicato in
protocol(control).NOME.from e in protocol(control).NOME.until
In alternativa il componente può essere impostato per una sola esecuzione singola al tempo indicato in protocol(control).NOME.at
Da notare inoltre che è possibile indicare un valore di tempo anche per il delay di un evento (ad esempio un messaggio tra un nodo e un altro).

Per quanto riguarda il collegamento tra i vari componenti, è necessario valutare per ogni componente se questo avrà bisogno di accedere a dati o metodi di altre istanze di componenti.
Esempio classico se un *Protocol* avrà bisogno di modificare delle sue variabili interne o attivare dei suoi metodi con periodicità (protocol.NOME.step) diversa dalla propria sarà necessario creare un *Control* con la periodicità (control.NOME.step) che serve e collegarlo ad ogni istanza del *Protocol*.
Va ricordato che per ogni *Control* verrà creata una sola istanza (che vede la rete nella sua totalità e ha quindi accesso a tutti i nodi) mentre per ogni *Protocol* sarà creata una sua istanza per ogni nodo.

Per collegare un componente si utilizza questa convenzione:

```
7 public class    Observer implements Control {
8     // ------------------------------------------------------------------------
9     // Parameters
10    // ------------------------------------------------------------------------
11
12    /**
13     * The protocol to look at.
14     *
15     * @config
16     */
17    private static final String PAR_PROT = "protocol";
18
19    // ------------------------------------------------------------------------
20    // Fields
21    // ------------------------------------------------------------------------
22
23    /**
24     * Value obtained from config property
25     * {@link #PAR_PROT}.
26     */
27    private final int pid;
28
29    // ------------------------------------------------------------------------
30    // Constructor
31    // ------------------------------------------------------------------------
32    /**
33     * Standard constructor that reads the configuration parameters. Invoked by
34     * the simulation engine.
35     *
36     * @param prefix
37     *            the configuration prefix for this class.
38     */
39    public BLEObserver(String prefix) {
40        pid = Configuration.getPid(prefix + "." + PAR_PROT);
41    }
```

In questo modo si ha a disposizione la variabile pid (l'id del componente che si vuole collegare). Quello che si vede nell'ultima riga di codice è il modo con cui vengono lette le informazioni dal file di configurazione, è da quest'ultimo che dovrà essere indicato il nome del componente da collegare. Analogamente dal file di configurazione è possibile passare anche valori numerici o testuali, basterà chiamare il metodo adeguato (Configuration.getString per una stringa ad esempio).

Nei *Controls* si può quindi accedere alle istanze di tutti i nodi dei protocolli collegati.

```
// Control interface method.
public boolean execute() {


    for (int i = 0; i < Network.size(); i++){

        Myprotocol myprotocol = (Myprotocol) Network.get(i).getProtocol(pid);
        ...
    }

    return false;
}
```

Mentre nei *Protocols* si può accedere alle istanze dei componenti collegati (sempre tramite la variabile pid) ma dei soli nodi di cui si ha visione, cioè di cui si conosce l'id.

```
Linkable linkable = (Linkable) node.getProtocol( FastConfig.getLinkable(pid) );
Transport transport = (Transport) node.getProtocol( FastConfig.getTransport(pid) );

for (int i = 0; i < linkable.degree(); i++) {

    Node peern = linkable.getNeighbor(i);

    Myprotocol myprotocol = (Myprotocol) peern.getProtocol(pid);
```

Il pid di un *Protocol* sarà sempre lo stesso per tutti i nodi, questo deriva dal fatto che tutte le istanze di uno stesso *Protocol* sono collegate fra loro di default, come detto in precedenza.

```
Linkable linkable = (Linkable) node.getProtocol( FastConfig.getLinkable(pid) );
Transport transport = (Transport) node.getProtocol( FastConfig.getTransport(pid) );

for (int i = 0; i < linkable.degree(); i++) {

    Node peern = linkable.getNeighbor(i);

    transport.send(node, peern, msg, pid);
```

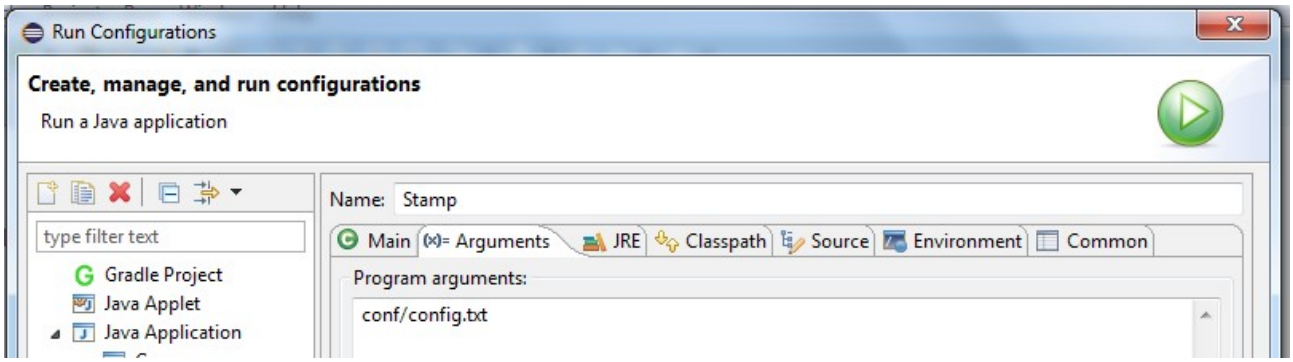
Per poter scambiare messaggi tra istanze di nodi diversi di uno stesso protocollo si può usare il livello di trasporto oppure direttamente EDSimulator.add

```
EDSimulator.add(
        delay,
        msg,
        peern,
        pid);
```

mentre per accedere ai nodi di cui si ha visione va utilizzato il livello linkabile, come mostrato.

Una volta presa confidenza con la gestione di queste due caratteristiche fondamentali si hanno tutti gli strumenti per progettare una propria simulazione e successivamente implementare un proprio protocollo, su cui si vuole sperimentare, e tutte le classi degli altri componenti necessari.

Importante infine ricordare che anche nel caso in cui si voglia avviare la simulazione da un Editor è necessario passare il file di configurazione come parametro:



Durante il run del programma, per monitorare l'andamento di determinate variabili (quelle utili da osservare) alla periodicità necessaria, vanno usati appositi *Control*, chiamati Observer.
Si consiglia tuttavia di usarli anche per salvare valori di interesse su file .dat, così da tenerne traccia. In questo modo li si avrà a disposizione anche per successive analisi. Saranno anche utilizzabili senza problemi su Gnuplot, oppure facilmente convertibili in formato Excel.