Abbiamo analizzato il problema di definire dei meccanismi che permettessero di programmare applicazioni su infrastrutture distribuite con servizi stateless quindi con servizi che sono funzioni che non si ricordano dello stato della loro esecuzione. Ci interessa andare a osservare come si possono progettare linguaggi di programmazione che tengono conto delle caratteristiche del problema, quindi programmare in termini di funzioni, quindi con linguaggi di natura funzionale, ci interessa definire un meccanismo di sicurezza in termini di quelli che abbiamo chiamato i policy framing, ovvero che definiscono delle politiche di sicurezza che dipendono dall intera storia dell esecuzione e ci interessa definire un meccanismo di orchestrazione che astrae ulteriormente il livello dell'applicazione dai dettagli delle scelte specifiche, ovvero di una mappatura specifica di un servizio a un opportuno fornitore del servizio. Adesso andiamo a vedere quali sono le caratteristiche dal punto di vista dell' implementazione. Andiamo a vedere quali sono le caratteristiche della struttura di implementazione e andando a vedere concettualmente quali sono gli aspetti che devono essere presenti nella macchina astratta dell esecuzione del linguaggio, ovviamente ci concentreremo sugli aspetti nuovi, ovvero sulle caratteristiche specifiche delle richieste che stiamo considerando, non ci soffermeremo sul discorso di, per esempio, cosa ci resta nel runtime del linguaggio funzionale questo già lo sapete e quindi su questo riportiamo tutto quello che già conosciamo rispetto all'implementazione dei linguaggi di programmazione. Se noi andiamo a vedere concettualmente, cosa ci sta nella macchina astratta dell'esecuzione del linguaggio, quando stiamo eseguendo localmente un programma, localmente perché noi abbiamo in mente di avere una struttura distribuita, quindi la nostra applicazione è composta di varie parti e ogni parte sta su un luogo fisico, quindi sta su un luogo locale. Allora concettualmente possiamo vedere lo stato della configurazione della macchina astratta del linguaggio che sta eseguendo localmente un programma come composto da due parti. La prima parte è facile ed è al programma che deve essere eseguito, quindi è quello che lì è rappresentata dall' espressione e dato che siamo in un contesto funzionale, normalmente il programma come se fosse un espressione, quindi ci saranno tutte le informazioni aspettate: il program counter, lo stack dei record di attivazione, eccetera. Però dobbiamo tener conto del fatto che c'è una componente in più, la componente in più è la componente della storia dell esecuzione perché il meccanismo di sicurezza è pensato per avere delle strutture a run time che tengono conto e memorizzano tutti gli eventi relativi alla sicurezza ovviamente che sono stati eseguiti localmente dall espressione che può essere un servizio, può essere il programma cliente e che sono stati eseguiti dal programma localmente, interagendo col suo ambiente operazionale quindi a seguito di richieste o a seguito di quello che succede nel contesto del programma. Sottolineo la parola local history, nel senso che localmente viene e deve essere memorizzata solamente quel fragmento dell'esecuzione che fa riferimento all'esecuzione locale, non deve essere memorizzata una storia globale.

Configuration of the abstract machine

$$\eta, e$$

Execution history

Program to be executed

Local histories only record the events generated by a service locally on its operational environment

ABSTRACT MACHINE: DEFINITION

Configuration of the abstract machine

$$\eta, e$$

Operational rules

$$\eta\, e \rightarrow \eta'\, e'$$

su questo torneremo tra un attimo e cercheremo di capire quali sono i vincoli di questa scelta allora se noi andiamo a vedere la configurazione della macchina astratta locale, sono quindi queste coppie n,e e le regole operazionali ci dicono come cambia lo stato della configurazione localmente quando viene eseguita una certa operazione quindi la coppia n,e in realtà va in una coppia n',e', che definisce il nuovo stato della configurazione e ci si muove a seguito di un operazione che può essere una richiesta del servizio, può essere un'operazione locale, può essere quello che sia quindi localmente abbiamo quindi la storia locale.

NETWORK: SERVICES

Node $\ell\{e{:}\tau\}{:}\ \eta, e'$ 

Stateless service: services do not preserve the state across distinct invocations.

No Trust Relations: a service only trusts its own history

=> it cannot constrain the past history of its callers, to prevent that its client has visited a malicious site

Le storie locali, i servizi fanno affidamento soltanto sul loro comportamento locale, sulle informazioni di stato locale che possono ispezionare direttamente se non avessimo un sistema distribuito con la possibilità di avere una storia globale, allora, a questo punto dovremmo avere delle mutue relazioni di trust tra l'entità che sono coinvolte nell applicazione distribuita, l'esempio per cui viene messo è: se un servizio ha una relazione di trust con b allora la storia di a può comprendere anche quella di b e quindi può controllare delle politiche che hanno a che vedere col comportamento di b ovviamente e ribadisco questo è molto più difficile da affrontare, per la natura altamente dinamica dell'applicazione, faremo vedere più avanti alcuni esempi, dove si può definire un modello di trust probabilistico, quindi avere non una nozione di trust vero falso ma dire io ho un modello probabilistico che mi permette di inferire l'affidabilità di un sito sopra una certa soglia, allora quelse siamo sopra la certa soglia, ci possiamo affidare.

Local Histories: **services do not trust other services**

Global histories require some trust relation among services: if a service A trusts B, then the history of A may comprise that of B, and so A may check policies on the behaviour of B.

a questo punto possiamo definire come sono le regole di comunicazione di rete, quindi com'e che si evolve una rete. Qui abbiamo fatto vedere come un servizio si evolve, quindi abbiamo un servizio, una locazione con un codice "e" e un interfaccia tau supponendo di avere eta la sua storia di esecuzione, e1 l'entità attualmente in esecuzione, diventa n', e2 ovviamente il servizio pubblicato rimane inalterato si modifica soltanto la storia e il programma che è attualmente in esecuzione la cosa importante è che l'evoluzione del nodo è guidata dal piano di esecuzione vi ricordate il piano di esecuzione tiene traccia delle relazioni tra le richieste del servizio, che sono caratterizzate da un etichetta. Se a un certo punto voglio richiedere un servizio che ha certe caratteristiche caratterizzate da una struttura di tipo che è il contratto della richiesta e a questo punto, il piano, che è il meccanismo che l' orchestrazione mette a disposizione a livello della stazione del programma, per individuare l'endpoint che risolverà la richiesta del servizio. Vuol dire che il piano è il meccanismo che l'orchestratore usa per guidare i passi dell esecuzione dell'applicazione distribuite e quindi il fatto di avere una macchina distribuita in questo caso di esecuzione guidata dal piano rende chiaro il ruolo giocato dall orchestrazione, che è quello di fare in modo che le richieste del servizio vengano esaudite in opportuni endpoint quando abbiamo una richiesta di esecuzione.

$$\ell\{e:\tau\}:\ \eta, e1 \rightarrow_{\pi} \eta', e2$$

The evolution of a node is driven by the plan π

VALUES

allora questi sono gli ingredienti, quindi abbiamo visto l' ingrandimento della macchina locale, il discorso di storia locale vs storie globali con la nozione di trust eventuale, adesso il concetto di macchina distribuita e come sono le configurazioni di una macchina distribuita adesso, per poter definire l'esecuzione della macchina astratta dobbiamo capire quali sono i valori, cosa sono i valori in un linguaggio funzionale, sono l'entità che non devono essere più valutate, quindi quando uno definisce un linguaggio funzionale deve dire quali sono i valori che non cessano di essere valutati. In termini di implementazione vuol dire che sono quelle entità che siamo sicuri che rappresenteranno dei dati a run time che non richiedono un ulteriore passo di valutazione. Nel nucleo di linguaggio che stiamo considerando, i valori cosa sono le variabili? perché non devono essere più valutate? perché basta fare un operazione di lookup dell'ambiente, andare a vedere che cosa c'è associato alla variabile, nell ambiente associa i nomi delle variabili a valori e quindi siamo sicuri che non dobbiamo valutarlo ulteriormente. Le astrazioni siano esse lambda normali, quindi funzioni anonime che non sono ricorsive o funzioni ricorsive sono valori nel linguaggio funzionale, le funzioni sono cittadini di ordine superiore e hanno tutti i diritti di tutti gli altri cittadini in realtà e quindi sono i valori del linguaggio e infine anche le richieste sono valori proprio perché ci muoviamo in un contesto che la stessa richiesta produce un valore che è il valore a cui sarà compito all'orchestratore di dare una risposta. Assumiamo di avere un valore distinto che ci serve in tutte quelle operazioni di cui abbiamo bisogno per attivare qualcosa. Dunque abbiamo un valore distinto da tutti gli altri, quindi diverso dai valori delle variabili, diverso dalle astrazioni, diverso dalle richieste che ci permette di caratterizzare il fatto che abbiamo un operazione che è disattivata, un'operazione che in quel momento è un' entità che non è operativa.

Values v are the variables, the abstractions, and the requests.

We write $*$ for a distinguished value

[Event]

$$\eta, \alpha \rightarrow \eta\alpha, ()$$

stiamo memorizzando l'azione alfa nello stato di esecuzione

[Framing In]

$$\frac{\eta, e \rightarrow \eta', e' \qquad \eta' \models \varphi}{\eta, \varphi[e] \rightarrow \eta', \varphi[e']}$$

[Framing Out]

$$\frac{\eta \models \varphi}{\eta, \varphi[v] \rightarrow \eta, v}$$

L'iftheelse è una cosa standard, cioè eseguo true nello stato eta se la guardia mi da true, eseguo il ramo else se la guardia mi va a false, allora l'If viene semplicemente rappresentato da questa regola che dipende dal valore di verità della guardia booleana, quindi è uno schema di regole che sostanzialmente corrispondono a due regole, la prima dice che se sono in uno stato eta, e la guardia di verità è vera, allora vado in uno stato eta e eseguo e_true, se la guardia verità invece è false vado in uno stato eta e devo continuare a eseguire l'espressione false e tutto questo è rappresentato dal fatto che vado sempre in uno stato eta e_B(b) che mi darà true o false, quindi vuol dire che caratterizzerà il ramo then o il ramo else

[If] η, **if b then** e_{true} **else** $e_{false} \rightarrow \eta, e_{\mathcal{B}(b)}$

Adesso andiamo a vedere le due regole dell applicazione funzionale. Supponiamo di avere un'applicazione di e1 e e2, notate che siamo sempre in un contesto funzionale dove abbiamo delle espressioni, le espressioni possono produrre dei valori e questi valori possono essere delle funzioni, possono essere delle funzioni definite in precedenza o possono essere delle funzioni che io ottengo come risultato della valutazione dell espressione, quindi vuol dire che ho un meccanismo dinamico di gestione delle funzioni che non necessariamente posso avere delle funzioni tutte dichiarate potrei ottenere una funzione anche come risultato della valutazione di un espressione.

[App1]

$$\frac{\eta, e_1 \rightarrow \eta', e_1'}{\eta, e_1\ e_2 \rightarrow \eta', e_1'\ e_2}$$

[App2]

$$\frac{\eta, e_2 \rightarrow \eta', e_2'}{\eta, v\ e_2 \rightarrow \eta', v\ e_2'}$$

Siamo nello stato locale eta, dobbiamo valutare e1 applicata ad e2, e1 rappresenta l'astrazione funzionale e2 rappresenta il parametro attuale su cui andrà operata l'astrazione. Allora anche in questo caso si procede passo passo nella valutazione di e1, e questo è quello che dice la regola App1. Quindi la regola dice se io ho un applicazione, devo cominciare a valutare la parte sinistra la regola ci dice esattamente come la macchina deve operare in termine di regole di esecuzione del calcolo, in un caso dell'applicazione, prima si valuta la parte sinistra dell'applicazione, poi solo dopo si valuta la parte destra e procedo nella valutazione della parte sinistra fintanto che non produco un valore, quello è la regola App2, la regola App1 continua a operare fin tanto che e1 è diverso da un valore, quand'è che scatta la regola App2, quindi quand'è che finisco di operare quella parte sinistra in un applicazione? Quando la parte sinistra di un'applicazione è diventata un valore, allora a quel punto comincio a operare e a valutare il secondo argomento dell'applicazione. In soldoni vuol dire che queste due regole ci rappresentano la regola di valutazione di un applicazione funzionale. L'applicazione funzionale è fatta da due argomenti, quindi nell'albero di sintassi astratta avrò un apply che prende un primo argomento e1 e un argomento e2, nell'albero di sintassi astratta l'interprete del linguaggio deve cominciare a valutare e1 e continua a valutare e1 fintanto che e1 non è arrivato un valore a questo punto sostituisce il valore e comincio a valutare e2 quindi vuol dire che comincio a valutare il parametro, l'espressione che corrisponde al parametro attuale dell'applicazione e comincia a valutare il parametro attuale, l'espressione parametro attuale, quindi l'applicazione viene valutata da sinistra verso destra, prima si valuta l'espressione che corrisponde all'astrazione funzionale e poi si va a valutare l'espressione che corrisponde al parametro attuale.

A questo punto abbiamo i due assiomi che corrispondono al fatto che abbiamo una lambda, quindi abbiamo una funzione non ricorsiva, siamo nello stato eta, l'astrazione funzionale ha dato come risultato un valore che è una lambda, quindi è un entità della forma lambda x e, la valutazione dell' espressione parametro attuale ha dato adito alla valutazione di un valore qualunque cosa sia e quindi qui vedete chiaramente espresso che le astrazioni di lambda siano esse ricorsivi o no sono dei valori, perché vedete questi sono esattamente i casi in cui la regola di esecuzione è corretta, allora se abbiamo valutato un astrazione funzionale con una lambda, abbiamo fatto la valutazione del parametro nell ambiente del chiamante ancora una volta, dobbiamo fare il passaggio del parametro, ed eseguire il corpo della funzione. Nel passaggio dei parametri usiamo un meccanismo Call-by-value associamo a tutte le occorrenze della x all'interno del corpo del programma, associamo alla x il valore, quindi questo vuol dire che concettualmente stiamo mettendo il record di attivazione della chiamata alla funzione e nel record di attivazione della chiamata della funzione mettiamo il legame tra il nome del parametro formale x che abbiamo trovato nella definizione dell astrazione e il valore del parametro attuale v che abbiamo ottenuto andando a valutare l'espressione parametro attuale e a questo punto eseguiamo in quell ambiente il corpo della funzione. La stessa cosa, passaggio dei parametri intendo, vale nel caso in cui abbiamo una funzione ricorsiva.

[AbsApp1] $\eta, (\lambda x.e)\ v \rightarrow \eta, e\{v/x\}$

[AbsApp2] $\eta, (\lambda_z x.e)\ v \rightarrow \eta, e\{v/x, \lambda_z x.e/z\}$

Una funzione ricorsiva è un lambda che si ricorda il nome della funzione stessa e quello che è nella meta notazione è lambda zeta nome della funzione x e parametro. Avviene la stessa cosa relativamente al passaggio del parametro, ovvero eseguiamo nello stato locale il corpo della funzione e facendo il passaggio dei parametri, quindi facendo il binding tra il nome del parametro formale x e il valore del parametro attuale v, però è una funziona ricorsiva quindi ci dobbiamo ricordare nell ambiente di esecuzione, ci dobbiamo ricordare che Z è definita e Z è definita come la funzione lambda z x, questo perché se noi non mettessimo nell'ambiente il legame tra il nome della funzione e la sua definizione, essendo la funzione ricorsiva nel corpo della e c'è un invocazione di zeta, vi ricordate abbiamo fatto la volta scorsa l'esempio classico del fattoriale, bene, se non mettessimo questa informazione nell'ambiente, ricordando le relazioni tra il nome della funzione e la sua definizione, quando andremo ad eseguire il corpo, non avremo un legame di ambiente per il nome della funzione ricorsiva e quindi daremmo un risultato non corretto perché andremo ad avere un fallimento a run time perché non avremo un legame per il nome della funzione ricorsiva. Questo sostanzialmente definisce localmente quello che occorre per andare a eseguire programmi locali.

Adesso dobbiamo vedere come funzionano le reti, allora vi ricordate abbiamo detto che le reti funzionano assumendo che l' orchestratore abbia le informazioni di planning e questo è il motivo in cui abbiamo il pigreco, che ci permettono di associare a le richieste del servizio l'endpoint dove questa richiesta di servizio viene esaudita da un servizio che è stato pubblicato. La regola di esecuzione ci dice come interagiamo tra il livello locale dell esecuzione del programma e la sua interfaccia di rete. Questa è la regola che qui abbiamo chiamato inject, quindi vuol dire che abbiamo la macchina che ha fatto un passo locale, quindi un passo locale vuol dire che la coppia eta,e è diventata eta',e' adesso abbiamo una visione locale che deve essere trasformata nella visione globale del nodo della rete beh, questo qui è banale quello che diciamo che la macchina con in esecuzione l'endpoint l, ha fatto un passo guidata e notate che qui ha fatto un passo senza bisogno di usare l' orchestratore perché l' orchestratore si vede soltanto quando abbiamo delle richieste di servizio, lo vedremo tra un attimo. A questo punto, quello che succede, è che la macchina all' endpoint l è diventata eta' dovuto al fatto che internamente la macchina ha fatto un passo di calcolo e quindi l'injection è semplice. Poi abbiamo la regola del parallelo, ci dice se una sotto componente della rete ha fatto un passo bene allora una parte più grande della rete fa lo stesso passo, vuol dire che in un'applicazione distribuita le varie componenti si muovono in modo asincrono, quindi quando una fa un passo, se il resto della rete non è coinvolto nel passo, si muove analogamente, quindi questo vuol dire che non abbiamo un clock globale perché per fare un passo tutti i componenti della rete non si devono sincronizzare sullo stesso clock ma possono muoversi in modo asincrono e questo è il minimo sindacale che uno si aspetta da un'applicazione distribuita in un infrastruttura che va dal cloud alla iot, cioè non avere un meccanismo di sincronizzazione unica.

[Inject]

$$\frac{\eta, e \rightarrow \eta', e'}{\ell: \eta, e \rightarrow_{\pi} \ell: \eta', e'}$$

La regola dell'injection mi dice qual'è la relazione tra l'evoluzione di una macchina locale, quindi il fatto che ho una macchina locale che si muove e allora a questo punto il nodo della rete si muove, la regola par mi dice, supponiamo di avere un'applicazione che è composta da 800 micro servizi, ogni micro servizio sarà un nodo della rete, un nodo della rete vuol dire caratterizzato da un endpoint che un indirizzo http con anche con la porta quindi non necessariamente corrisponde a una macchina fisica sono degli endpoint poi però sono l'astrazione di un container di un'esecuzione allora a questo punto, supponendo di avere un certo numero, supponiamo che un endpoint si muove, allora quelli che stanno in parallelo con lui si muovono allo stesso tempo, anche se non sono coinvolti nell esecuzione, non c'è una sincronizzazione bene, immaginiamo di farlo passo dopo passo, ne abbiamo due, a questo punto tre ,anche il terzo si muove, supponendo di averne 25 dal secondo al 25° si muovono tutti, questa regola dice esattamente che una sotto componente di una rete si muove allora se la sotto componente della rete che era N1 si muove e diventa N1', la rete che è composta da N1 parallelo N2 si muove ma la seconda sotto componente, quella che è N2 vedete, rimane inalterata, si muove soltanto N1, vuol dire che abbiamo un evoluzione complessiva di una rete di applicazione che si muove in modo asincrono soltanto parti locali sono coinvolte non tutto però la applicazione distribuita si muove.

[Par]

$$\frac{N_1 \rightarrow_{\pi} N_1'}{N_1 \parallel N_2 \rightarrow_{\pi} N_1' \parallel N_2}$$

adesso ci rimangono sostanzialmente l'interazione e la comunicazione via rete, che cosa succede quando io faccio una richiesta di servizio e cosa succede quando la richiesta di servizio è stata eseguita. Allora andiamo a vedere come si muove, quindi supponiamo di essere in una configurazione di una sotto componente della rete e in questa configurazione abbiamo un endpoint l, abbiamo uno stato eta e a questo punto il programma in esecuzione locale ha fatto una richiesta di un servizio r con un valore v che potete vedere come il parametro della richiesta di servizio. A questo punto dobbiamo chiamare l' orchestratore per fare la mossa il quale ci dice, guardate nella rete esiste un endpoint l' e noi sappiamo che la richiesta di servizio r esaudita dall'endpoint l' quindi questo vuol dire che l'orchestratore è stato in grado di fare il matching tra la richiesta del servizio che ha fatto il cliente e le informazioni del servizio che sono state pubblicate e quindi da in pasto al servizio l' la richiesta. Il fatto che ci sia un servizio con un endpoint l' è caratterizzato dal fatto che questo vuol dire che la richiesta di servizio deve andare in parallelo con un nodo che ha esattamente come endpoint l' ha come codice il codice del servizio e' e questo servizio non è attivo. Questo corrisponde al fatto che la storia locale è una storia vuota e il valore locale quel servizio è il valore distinto stellina che abbiamo detto prima. Se questa è la situazione, quello che succede è che possiamo avere una mossa dei due nodi, quindi i due nodi si sono sincronizzati sulla richiesta del servizio perché l' orchestratore ha indicato come endpoint l', a questo punto, quello che succede è che sul nodo delle richieste del servizio rimane una operazione di wait, ovvero sono in attesa della risposta dell'endpoint l'. Notate che qui stiamo operando anche se in modo leggero, un meccanismo di sincronizzazione tra la richiesta e l'esecuzione, quindi abbiamo un meccanismo che ci permette di fare l'esecuzione remota di una funzione e rimanere in attesa del risultato della funzione. Questo sul lato cliente, sul lato del servizio succede che viene mandato in esecuzione il codice di l' e gli viene dato come valore esattamente il valore della richiesta allora questo punto poi localmente nel nodo l' verrà mandata avanti l'esecuzione dell'applicazione e un v qualunque cosa sia l'esecuzione dell'applicazione, andando avanti, vedete, abbiamo la replay. La replica avviene in una situazione in cui nella rete abbiamo un nodo che sta attendendo il risultato dall'endpoint l', su l' ho eseguito il codice del servizio e sono arrivato esattamente alla produzione di un valore v con lo stato locale eta', a questo punto mando indietro, faccio il marshalling del risultato sul nodo all'endpoint l, quindi a questo punto sul nodo all'endpoint l avrò il valore eta di v, e sul nodo del servizio ripristino la situazione iniziale, ovvero cancello la storia dell esecuzione infatti diventa y e produco il valore stellina che sta a indicare che a questo punto il servizio può essere utilizzato per una nuova esecuzione. Allora questo caratterizza esattamente i due aspetti che avevamo detto all'inizio di questa lezione, ovvero quando io devo andare a controllare le proprietà di sicurezza perché e' al suo interno avrà dei framing, quando io vado a controllare nell esecuzione di e' le proprietà di sicurezza, controllo le proprietà di sicurezza relative allo stato dell esecuzione locale del servizio, ovvero quello che qui abbiamo indicato l', non vado a controllare la storia eta del cliente, e quindi questo ci rappresenta bene il fatto che le storie sono locali e la portata delle politiche di sicurezza ha una portata locale alla storia dell esecuzione. La seconda cosa che ci caratterizza bene questo reply e anche la request se lo andiamo a vedere in sequenza è che i servizi sono stateless, ovvero quando parte una richiesta di servizio viene attivato il servizio con il parametro che corrisponde alla richiesta del cliente, il parametro v e vado ad eseguire il servizio su quel parametro, quindi quella particolare instanziazione del servizio a partire dalla storia vuota. Nella reply si vede bene che quando il servizio ha prodotto un valore, viene fatta un'operazione che reinstalla il servizio ma lo reinstalla allo stato vuoto, infatti vedete, abbiamo epsilon quindi vuol dire che quando ripartirà il servizio ripartirà con lo stato locale vuoto, non ci sono informazioni di stato che rimangono memorizzate tra un' invocazione di un servizio e quella successiva, e quindi i servizi sono stateless e questo caratterizza bene la nozione di stateless.

[Request]

$\pi = r[\ell'] \mid \pi'$

$$\ell: \eta, \mathrm{req}_r\ v \ \|\ \ell'\{e'\}: \varepsilon, \star$$

$$\rightarrow_{\pi}$$

$$\ell: \eta, \mathrm{wait}\ \ell' \ \|\ \ell'\{e'\}: \varepsilon, e'v$$

[Reply]

$$\ell: \eta, \mathrm{wait}\ \ell' \ \|\ \ell'\{e'\}: \eta', v$$

$$\rightarrow_{\pi}$$

$$\ell: \eta, v \qquad \|\ \ell'\{e'\}: \varepsilon, \star$$

La reply cosa mi dice? Io sono l e ho fatto una richiesta di servizio e sto aspettando che l' mi restituisca un valore e questo è caratterizzato che nell endpoint l ho un wait di l', con uno suo stato. In l' ho mandato in esecuzione la richiesta del servizio, ho prodotto un valore v e una storia eta'. Allora a questo punto ho su l' endpoint l' ho prodotto il valore con la sua storia, sull endpoint l sto aspettando la risposta del servizio, devo fare la sincronizzazione tra queste due entità. Il servizio all endpoint l e il servizio all'endpoint l' . Come faccio a fare la sincronizzazione? La sincronizzazione è che il valore v che è stato prodotto da l' deve andare ad l, ma i servizi sono stateless e l'informazione di stato è locale al servizio, vuol dire che nella trasmissione del risultato trasmetto soltanto il valore v non la storia eta', che rimane locale all'esecuzione del servizio quindi trasmetto solo il valore questo è esattamente il motivo per cui prima avevo detto che le storie sono locali, cioè ogni entità memorizza solo la propria storia di esecuzione. Analogamente, sul lato l' devo continuare ad avere il servizio attivo e il servizio attivo lo ho perché l'interfaccia rimane ancora attiva l' ha sempre l'interfaccia, però devo azzerare la sua storia di esecuzione e quindi azzerando la sua storia di esecuzione ho che la storia sull'endpoint l' è diventata epsilon.

OTHER KINDS OF PLANS

noi in questo momento stiamo caratterizzando il comportamento dell' orchestratore e l'abbiamo già detto più di una volta, in termini di quello che abbiano chiamati i piani semplici, o vuoto o la composizione di due piani dove a un entità ho associato l' endpoint e questo è il piano che vedete qui, è l'astrazione che abbiamo fatto vedere in precedenza, dove abbiamo una sincronizzazione tra due entità in rete, sapendo che quello è l'endpoint, ma posso avere anche altri tipi di piani, ad esempio io potrei avere un orchestratore che a una richiesta di servizio mi può associare più di un endpoint e quindi può scegliere in modo non deterministico un endpoint dove posso soddisfare, oppure potrei avere dei piani nell orchestratore che sono dipendenti, ovvero potrei fare una scelta il fatto di fare una scelta poi mi vincolerà successivamente sulle altre scelte. Vuol dire che il meccanismo dell' orchestrazione è raffinato e dipende dalle scelte che uno fa quando vuole gestire un' astrazione all'interno del linguaggio di programmazione relativamente al modo di soddisfare dei vincoli di servizio.

- **Simple plans** $\pi ::= 0 \mid \pi \mid \pi \mid r[\ell]$

$$\ell : req_r \,||\, \ell' : \{P\} \rightarrow_{r[\ell']} \ell : wait\ \ell' \,||\, \ell' : P$$

- **Multi-choice plans** $\pi ::= 0 \mid \pi \mid \pi \mid r[\ell_1 \ldots \ell_k]$

$$\ell : req_r \,||\, \ell' : \{P\} \rightarrow_{r[\ell', \ell'']} \ell : wait\ \ell' \,||\, \ell' : P$$

- **Dependent plans** $\pi ::= 0 \mid \pi \mid \pi \mid r[\ell.\pi]$

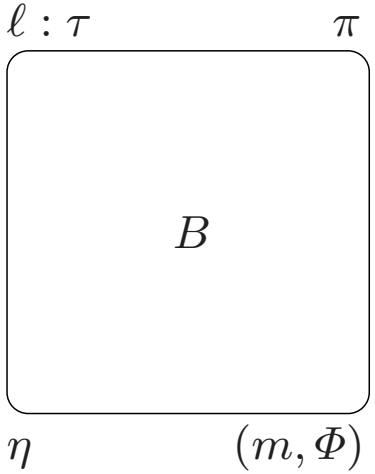
$$\ell : r[\ell'.\pi] \triangleright req_r \,||\, \ell' : \{P\} \rightarrow \ell : r[\ell'.\pi] \triangleright wait\ \ell' \,||\, \ell' : \pi \triangleright P$$

- …many others: multi+dependent, regular, dynamic,…

AN ALTERNATIVE FORMULATION

Andiamo a vederlo concettualmente, come è fatto a questo punto lo stato di esecuzione di un servizio? Lo vedete in termini della macchina di esecuzione, lo stato di esecuzione del servizio è caratterizzato dal fatto che in un certo endpoint, quindi, bisogna tener conto nelle informazioni nel fatto che sono all' endpoint l, ha pubblicato la sua interfaccia di comportamento che è il tau, esiste un orchestratore che ha il piano dell esecuzione, quindi l'orchestratore pigreco è l'entità che esiste, l' orchestratore è globale, quindi sa quali sono le varie entità, ha un codice B che è il codice del servizio, qualunque cosa sia. Ha uno stato eta che corrisponde alla storia degli eventi e poi ha una coppia che noi non abbiamo ancora considerato, perché prima volevo darvi un'idea intuitiva, anche se precisa dell'esecuzione, adesso entriamo a un discorso più dettagliato del container, ha un entità che è una coppia, (m,PHI) PHI è la sequenza delle politiche attive perché vi ricordate noi possiamo avere una politica all'interno di un'altra politica, così via come nel record di attivazione abbiamo lo stack dei record di attivazione per la politica abbiamo una struttura a stack che rappresenta l'ordine di esecuzione perché posso avere un frame di sicurezza che al suo interno ne prende un altro e così via. Ma perché mai vi mettete quell'altro flag m? Il flag m ci dice: vogliamo controllare la politica, cioè vogliamo dire che il monitor è attivo? Perché uno potrebbe anche avere uno stato in cui dice che non gliene fotte niente di avere una politica di sicurezza, io mi fido, allora questo punto non voglio avere overhead di esecuzione perché permettere politiche che siano verificate a runtime corrisponde ad avere un overhead. Se io assumo che io non sono mai attaccato perché mi fido dell'universo mondo accanto a me, non ho overhead a tempo di esecuzione quindi metto il flag ad off, ovviamente è rischioso fare questo. Uno potrebbe nella macchina di esecuzione potrebbe decidere che il runtime execution monitor non è attivato e quindi vuol dire concettualmente che tutte le politiche sono sempre verificate.

SERVICES: EXECUTION STATE



$\ell : \tau$	service location ℓ + interface τ
π	orchestration plan
η	event history
(m, Φ)	monitor flag m + sequence Φ of active policies
B	service code

$$\ell\langle e : \tau \rangle : \pi \rhd \eta, m, e'$$

Flag m representing the on/off status of the execution monitor on the active security policies.
When the flag is on, the monitor checks that the service history η adheres to the policies at hands.

STAND ALONE SERVICES

$$\eta, m, (\lambda_z x.\, e)v \to \eta, m, e\{v/x, \lambda_z x.\, e/z\}$$

$$\eta, m, \alpha \to \eta\alpha, m, *$$

$$\eta, m, \texttt{if}\ b\ \texttt{then}\ e_{tt}\ \texttt{else}\ e_{ff} \to \eta, m, e_{\mathcal{B}(b)}$$

$$\eta, m, \mathcal{C}(e) \to \eta', m', \mathcal{C}(e') \qquad \text{if } \eta, m, e \to \eta', m', e' \text{ and } m' = \mathit{off} \vee \eta' \models \Phi(\mathcal{C})$$

$$\eta, m, \mathcal{C}(\varphi[v]) \to \eta, m, \mathcal{C}(v) \qquad \text{if } m = \mathit{off} \vee \eta \models \varphi$$

where $\mathcal{C}$ is an *evaluation context*, of the following form:

$$\mathcal{C} \quad ::= \quad \bullet \mid \mathcal{C}\, e \mid v\, \mathcal{C} \mid \varphi[\mathcal{C}]$$

and $\Phi(\mathcal{C})$ is the set of *active policies* of $\mathcal{C}$, defined as follows:

$$\Phi(\mathcal{C}\, e) = \Phi(v\, \mathcal{C}) = \Phi(\mathcal{C}) \qquad \Phi(\varphi[\mathcal{C}]) = \{\varphi\} \cup \Phi(\mathcal{C})$$

NETWORK I

$$[\textsc{Sta}] \qquad \frac{\eta, m, e \to \eta', m', e'}{\ell : \pi \rhd \eta, m, e \to \ell : \pi \rhd \eta', m', e'}$$

$$[\textsc{Net}] \qquad \frac{N_1 \to N_1'}{N_1 \parallel N_2 \to N_1' \parallel N_2}$$

La rete cosa è caratterizzata? Un nodo in una rete è caratterizzata quindi da un endpoint e per questo vi ho introdotto questa formulazione fatta in questo modo, perché ora ve lo linearizzo, caratterizzata da un endpoint, dal piano dell' orchestratore, dallo stato dell'esecuzione, dal monitor e dall'espressione "e" che sta eseguendo. Allora vedete com'è fatta la regola? Se noi siamo in questo stato e la componente si muove ed è diventata eta' allora anche il nodo della rete si muove analogamente, che cosa vuol dire? vuol dire che un nodo della rete si muove se al suo interno ci sono delle azioni che lo fanno muovere, fanno modificare lo stato. Per quanto riguarda il comportamento asincrono, la regola rimane esattamente come prima.

NETWORK2

$$[\textsc{Pub}] \qquad N \to N \parallel \ell\langle e : \tau \rangle : 0 \rhd \varepsilon, \mathit{ff}, * \quad \text{if } \ell \text{ fresh and } \vdash_\ell e : \tau$$

$$[\textsc{Down}] \qquad \ell\langle e : \tau \rangle : 0 \rhd \varepsilon, m, * \to \ell\langle e : \tau \rangle : 0 \rhd \varepsilon, m, \mathsf{N/A}$$

$$[\textsc{Up}] \qquad \ell\langle e : \tau \rangle : 0 \rhd \varepsilon, m, \mathsf{N/A} \to \ell\langle e : \tau \rangle : 0 \rhd \varepsilon, m, *$$

adesso andiamo a vedere la pubblicazione il fatto che io possa avere un servizio down e il fatto che possa attivare un servizio. Vogliamo modellare e questo ci permette di descrivere contesti dove vogliamo modellare anche entità dinamiche e come faccio a modellare la pubblicazione? La pubblicazione è sostanzialmente un azione asincrona, io ho una rete a questo punto la rete si evolve in modo asincrono, diventa un nuovo endpoint dove ho pubblicato il servizio e, con la sua nuova interfaccia di comportamento delta e tau, non ho alcun piano per questo, perché non ha alcun piano? Perchè il servizio è stato pubblicato ora quindi nessuno la conosceva e notate che a questo punto, quando vado ad attivare questo nuovo servizio, quindi senza piano, lo vado ad attivare in uno stato che non ha storia, il monitor è false, non è attivato e lo stato locale è lo stato idle. Lo posso fare quando però so che l' endpoint è fresh, cioè non ci deve essere essere alcun endpoint che ha già occupato lo stesso servizio. Allora questa regola pub descrive nel dettaglio che cos'è la creazione dinamica di servizi in una rete di servizi, cioè fa vedere le condizioni che devono essere soddisfatti, io lo posso pubblicare quando gli pubblico l'interfaccia del servizio, una volta pubblicato, non ho alcun meccanismo di orchestrazione che lo gestisce perché nuovo, il flag di monitoraggio è a falsa e a questo punto non ha alcuna storia locale. Posso fare l'azione duale. Sono in un sistema che è stato pubblicato, non ho ancora piano di orchestrazione, qualunque sia il valore del flag, bene una cosa che posso fare è metterlo not available, quindi l'ho pubblicata e nessuno mi considera allora a questo punto vabbè, è inutile ho fatto della fatica inutile. L'altro invece è la regola che mi dice se io ho un servizio pubblicato ed era non attivo, lo posso rendere di nuovo attivo quindi questo mi va a monitorare lo stato dei servizi.

NETWORK3

$$[\textsc{Req}] \qquad \ell : (r[\ell'] \mid \pi) \rhd \eta, m, \mathcal{C}(\mathtt{req}_r \rho\, v) \parallel \ell' \langle e : \tau \rangle : 0 \rhd \varepsilon, m', * \longrightarrow \ell : (r[\ell'] \mid \pi) \rhd \eta, m, \mathcal{C}(\mathtt{wait}\, \ell') \parallel \ell' \langle e : \tau \rangle : (r[\ell'] \mid \pi) \rhd \sigma, m, e\, v$$

$$[\textsc{Ret}] \qquad \ell : \pi \rhd \eta, m, \mathtt{wait}\, \ell' \parallel \ell' : \pi' \rhd \eta', m', v \longrightarrow \ell : \pi \rhd \eta, m', v \parallel \ell' : 0 \rhd \varepsilon, m', *$$

The special event σ signals that the service has started.

uindi se sono in un contesto che ho fatto una richiesta di servizio e l' orchestratore mi dice che risolvo questa richiesta di servizio all' end point l' quello che faccio è la stessa operazione che abbiamo visto prima, attivo l'esecuzione del servizio col suo monitor, e ho una particolare azione sigma per dire l'ho attivato, quindi mi ricordo che l'ho attivato. Il return rimanda indietro il valore, non manda indietro la storia e riazzera il servizio sull endpoint l'.

NETWORK: ADDITIONAL RULES

$$\ell : (r[?] \mid \pi) \triangleright \eta, m, \mathcal{C}(\mathtt{req}_r \rho\ v) \rightarrow \ell : (r[?] \mid \pi) \triangleright \eta, m, \mathcal{C}(\{\mathtt{req}_r \rho\ v\})$$

$$\ell : \pi \triangleright \eta, m, \{e\} \rightarrow \ell : \pi' \triangleright \eta, m', e \quad \text{if } (\pi', m') = plan(\pi, m, e)$$

Abbiamo anche delle regole addizionali che prima non abbiamo visto. Ci permettono di tener conto dell'introduzione dinamica di servizi, ovvero vogliamo avere delle applicazioni che possono fare ad esempio delle operazioni di richiesta che non sono ancora risolte, succede che supponiamo di avere un orchestratore che non è in grado di risolvere la richiesta di servizio r, che non ha ancora avuto informazioni sufficienti, quindi vuol dire che sono in r, sono con il piano di esecuzione della prestazione che a r non mi ha dato ancora un endpoint, e ho una richiesta di servizio r. Quello che faccio è vado a reiterare la richiesta di servizio, vuol dire che faccio un passo dove non sono ancora in grado di risolverlo però continuo a fare la richiesta di servizio. Perchè uno può avere un meccanismo in cui la pianificazione può tener conto della dinamicità dell' attivazione dei servizi e la dinamicità la vedo in questo modo, con la regola seguente: sono in un endpoint, ho un piano ho uno stato e ho un esecuzione di una espressione e, notate che l'espressione e è all'interno di parentesi graffe. Questo corrisponde al fatto che io ho un codice e di questo codice voglio avere una pianificazione cioè non avevo la pianificazione precedente e allora a questo punto voglio avere il piano per questo codice che sto mandando in esecuzione. Allora a questo punto quello che succede è che io posso scongelare questo codice, infatti vedete nella parte destra dell'esecuzione del codice vengono tolte le parentesi angolate del congelamento, della pianificazione del codice viene mandato in esecuzione però ho modificato l' orchestrazione cioè l'orchestrazione e in realtà le informazioni di stato è una funzione plan che mi riattiva l' orchestrazione tenendo conto del piano dell esecuzione dello stato m e delle richieste dei servizi e, va a vedere una nuova orchestrazione. Concettualmente plan è l' orchestratore che va a vedere se mi è stato creata una nuova richiesta di servizio, devo dare un nuovo piano di esecuzione per lui, vado a vedere quali sono i servizi che ho in questo momento attivi e magari tra i servizi attivi posso considerare quelli che sono stati creati durante l'esecuzione che inizialmente non avevo considerato. Una volta che ho fatto questo, ho un momento di centralizzazione, faccio la discovery dei servizi e a seconda della discovery dei servizi individua un nuovo piano a questo punto il piano p' è quello che mi guida l' orchestrazione e l'esecuzione di e. Queste regole permettono di gestire in modo più dinamico la creazione e l'attivazione di nuovi servizi.

RECAP

- calculus: syntax and **operational semantics**
  - **Service as a function**
  - **call by properties**
  - **Operational semantics & orchestration**

Abbiamo definito servizi come funzioni, abbiamo fatto la call by property, abbiamo definito la semantica operazionale in vari stili e i meccanismi di orchestrazione e pianificazione, adesso quello che uno si domanda è, dato che costa fare il controllo a run time di politiche che possono essere innestate uno dentro l'altra, possiamo avere un meccanismo che ci permette di controllare staticamente le condizioni in cui possiamo tenere il flag abbassato? O detto in altri modi, possiamo definirci delle condizioni che permettono di vedere se un'applicazione che abbiamo scritto rispetta staticamente le politiche di sicurezza? Allora se siamo in grado di determinare quando con un meccanismo di analisi statica che la politica di sicurezza è rispettata, allora a questo punto possiamo abbassare il flag e quindi possiamo non avere un controllo a tempo di esecuzione delle politiche se sono attive perché l'abbiamo controllato staticamente La formulazione alternativa è solo per far vedere che si può gestire in modo dinamico la creazione, la disattivazione dei servizi e il meccanismo di orchestrazione.