CONDIVISIONE DI SERVIZI CON JADE

1. Scopo del progetto

Il progetto ha come argomento la coordinazione tra agenti. Lo scopo è quello di permettere la **condivisione di servizi** presenti su uno o più dispositivi partecipanti ad una sessione. E' stata implementata una architettura che gestisce gli agenti (client/server) di modo che ogni host dell'architettura può usufruire di un servizio proposto (**client**), oppure offrire esso stesso un servizio (**server**).

L'architettura proposta usa **Jade** come framework per la comunicazione e la coordinazione tra agenti, quindi è necessario un layer di comunicazione: se gli agenti risiedono su dispositivi diversi, devono essere connessi da una rete.

Come caso di studio è stato implementato un servizio di **chat**. Esso serve soprattutto come esempio di utilizzo della piattaforma e può essere usato come pattern di sviluppo di eventuali servizi aggiuntivi: condivisione di stream multimediali, file sharing, file transfer, etc.

2. Struttura dell'architettura

Il progetto considera tre entità:

- l'architettura di supporto (middleware)
- il superserver
- il client

Le entità della piattaforma posso essere distribuite in modo eterogeneo sui nodi partecipanti. Il nodo sul quale risiede il middleware (in Jade è il nodo sul quale c'è il Main Container) viene considerato il nodo principale e deve essere il primo ad essere creato. I superserver e i client possono essere avviati su qualunque nodo e in qualunque quantità. Questo vuol dire che su un nodo è possibile avere avviati sia client che server in contemporanea (essendo agenti indipendenti).

L'architettura di supporto è composta da tutte le entità (framework / strumenti) che permettono la condivisione dei servizi. In questo momento il middleware comprende Jade come elemento di base, ma in sviluppi futuri potrebbe includere TuCSoN (come memoria dei dati di una sessione) o altre strumenti necessari alla coordinazione.

3. Interazione tra le entità

Sia il superserver che il client mantengono una lista dei servizi di cui sono a conoscenza. Il server ha una lista dei servizi che può offrire, il client una lista di servizi che può richiedere. Eventualmente il client può interrogare il server riguardo ai servizi disponibili (questa funzionalità non è ancora implementata nel progetto, ma all'occorrenza la si può introdurre). La lista è mantenuta in un file di testo, ma un miglioramento sarebbe utilizzare file xml che permette una organizzazione più facile.

La lista contiene oltre al nome dei servizi, anche alcune proprietà che li caratterizzano e le classi che li implementano. Nella versione corrente il numero di versione del servizio è una proprietà, ma possono essere aggiunte anche altre.

La negoziazione di un servizio si basa sulla shared knowledge (il client e il server conoscono a priori il protocollo di negoziazione). L'iniziatore è il client. Esso interroga il DF (Directory Facilitator) riguardo a superserver che condividono un servizio. Ottenuta la lista di quelli che offrono quel servizio, il client manda una CFP (Call For Proposal) a ciascuno di questi ultimi. I superserver rispondono con il numero di versione del servizio che offrono. Il client sceglie il superserver con la versione migliore per lui e richiede il servizio. Il superserver verifica se il server che gestisce il servizio è già avviato. In tal caso aggiunge il client alla sessione già attiva. Altrimenti, il superserver crea un nuovo server che andrà a gestire il servizio richiesto dal client. A questo punto l'interazione prosegue tra server e client, mentre il superserver torna ad aspettare altri clienti.

Il server di un servizio rimane attivo finché ci sono client che partecipano alla sessione di quel servizio. Quando non ci sono più client che utilizzano quel servizio, il server finisce l'esecuzione. Una richiesta successiva per lo stesso servizio sarà gestita da un nuovo server.

4. Testing

I test sono stati fatti su un'unica macchina per questioni di semplicità di sviluppo, ma questo non è un vincolo. Infatti, si può benissimo includere dispositivi mobili (fondamentalmente con Android) tra i nodi partecipanti come client. Non è stato testato il caso di un superserver su un dispositivo mobile (potrebbero esserci problemi legati alla dimensione della VM).

Come caso di studio è stato implementato un servizio di chat. Un client che lo richiede è abilitato a mandare messaggi di testo a tutti i nodi che stanno partecipando alla chat. Per ora la modalità di invio è solamente quella broadcast, ma è possibile implementare anche una modalità in cui il client può selezionare i nodi ai quali inviare un particolare messaggio.

5. Possibili sviluppi futuri

La struttura dell'architettura è tale da permettere l'aggiunta delle seguenti funzionalità:

- Caricamento di servizi (classi java) da client a server a run-time (facilitata dalla memorizzazione della corrispondenza servizio-classi che lo implementano)
- Scaricamento di servizi (classi java) da server a client per permettere al dispositivo del client di usufruire del servizio, oppure di avviare un server proprio (qui sarebbero da gestire le rispettive dipendenze)
- RBAC integration (per l'accesso a particolari servizi del superserver, come quello di aggiornamento dei servizi a runtime)

NOTA: Nel caso di sviluppi futuri è consigliato consultare anche la documentazione javadoc allegata al progetto.