Architettura PeerToPeer

Package Server

devices/

1. **LocalExecutionPeerDevice:** esattamente come la classe LocalExecutionDevice vista nella architettura Client/Server rappresenta la piattaforma di esecuzione in modalità locale, l'unica differenza che è conforme alla struttura della classe PeerDevice al posto di RemoteDevice

2. **PeerDevice**: esattamente come RemoteDevice, rappresenta la piattaforma di esecuzione nella quale è il Client remoto ad eseguire il programma aggregato, le differenze sono che utilizza l'interfaccia NetworkCommunication per comunicare e che il PeerDevice può rappresentare anche un Support remoto (che rappresenta un vicino) alla quale il Support può inviare messaggi come SendToNeighbour. Il PeerDevice conosce l'informazione se rappresenta un Client oppure no grazie alla collaborazione dell'interfaccia NetworkCommunication.

network/

3. **ServerSupport**: a differenza del Support dell'architettura Client/Server è modellato come un PeerDevice, possiede le stesse responsabilità di Supporto con la differenza è che deve avere l'informazione se una Join e una SendToNeighbour sono state inviate da PeerDevice che rappresentano Client oppure che rappresentano altri ServerSupport remoti, questa informazione è contenuta nel campo content della Join e nella classe PeerDevice. In base a questa informazione cambia il comportamento della SendToNeighbour e della Join

4. **PeerDeviceManager**: La differenza con il DeviceManager è che mantiene l'informazione di quali Device sono Support e quali sono Client (cioè connessi direttamente al ServerSupport del dispositivo fisico). La grossa differenza è che non conosce più la topologia delle rete, perché ogni ServerSupport possiede il proprio PeerDeviceManager, questa informazione è distribuita, ogni PeerDeviceManager possiede una informazione locale sulla rete, ovvero tutti i client locali che sono direttamente connessi al ServerSupport in esame e i ServerSupport direttamente connessi al ServerSupport che rappresentano il vicinato.
   Quindi per esempio se esistesse solo un ServerSupport nel quale più client si connettono vengono compiute le stesse operazioni dell'architettura Client/Server, la topologia in questo caso sarebbe FullyConnected.
   Per le relazione di vicinanza tra Client possono essere considerate le seguenti tre regole:
   -se due client sono connessi allo stesso ServerSupport allora la relazione di vicinanza tra questi due client è vera
   -se due client sono connessi a due ServerSupport distinti e questi due ServerSupport sono connessi allora la relazione di vicinanza tra questi due client è vera
   -in tutti gli altri casi la relazione di vicinanza tra i due client è falsa.

5. **NetworkController**: offre servizi applicativi come il ServerSupport (lo cerca se non è supportato), possiede una lista di CommunicationControllerInterface che sono Interfacce Controller di vari Livelli Network che possono essere inseriti per aumentare in maniera indipendente funzionalità (come BLE)

6. **CommunicationControllerInterface**: interfaccia che specifica le responsabilità per ogni livello Network che si decida di aggiungere

7. **NetworkInformation**: interfaccia che viene utilizzata quando si ricevono messaggi Join, incapsula informazioni utilizzate per assegnare il physicalDevice al PeerDevice e viene restituita al ServerSupport per ricevere il contenuto del messaggio (che è stato incapsulato) e per assegnare il physicalDevice con il giusto NetworkCommunication). Questa interfaccia nasce dal bisogno di non cambiare le responsabilità già assegnate nell'architettura Client/Server, ovvero il DeviceManager continua ad essere il Creator del Device ed il Support non ha dipendenze sulle implementazioni di Communication, tuttavia finché si usavano socket asincroni dove bastava un indirizzo per avere tutte le informazioni necessarie, questo non vale per altri parametri T(come i socket classici) e quindi il Server Support non possiede le informazioni necessarie per assegnare la giusta implementazione della comunicazione al Device, il device è creato dopo che il socket è già stato aperto, quindi si utilizza questa interfaccia che continua a nascondere le implementazioni dal Server Support lasciando libertà a chi la implementa di avere le informazioni necessarie e la loro gestione.
   Ho utilizzato Socket come parametro T perché la maggior parte dei protocolli di rete li utilizza e lasciare l'implementazione più generale possibile (ad esempio se si decidesse di utilizzare WiFi Aware)

network/communication/

8. **NetworkCommunication**: questa classe astratta implementa l'interfaccia Communication nella quale il parametro T è un Socket, le responsabilità oltre quelle di communication sono quella di offrire una funzione connect, offrire una funzione che indica se si è connessi ad un client e la possibilità di aggiungere un'altra network communication oltre questa, per ora esiste un campo nextCommunication che specifica un pattern Chain of Responsability per lasciare in futuro la possibilità ad un PeerDevice di avere più device fisici (ovvero è possibile raggiungere il Peer con più livelli Network), ma può essere tolta in qualsiasi momento senza cambiare il comportamento delle altre classi.

network/communication/local

9. Qui ho implementato le classi sopracitate, vengono utilizzati indirizzi locali utilizzati per raggiungere Server Support e Client, con la differenza che l'indirizzo a cui si vuole connettere deve essere dato esplicitamente (non c'è un protocollo di ricerca attivo che autonomamente ricerca se esistono server nella rete, ma può essere aggiunto in qualsiasi momento)