Politecnico di Milano
Polo Territoriale di Como

Prova finale di Ingegneria del Software
Documento di progetto – Diagrammi UML

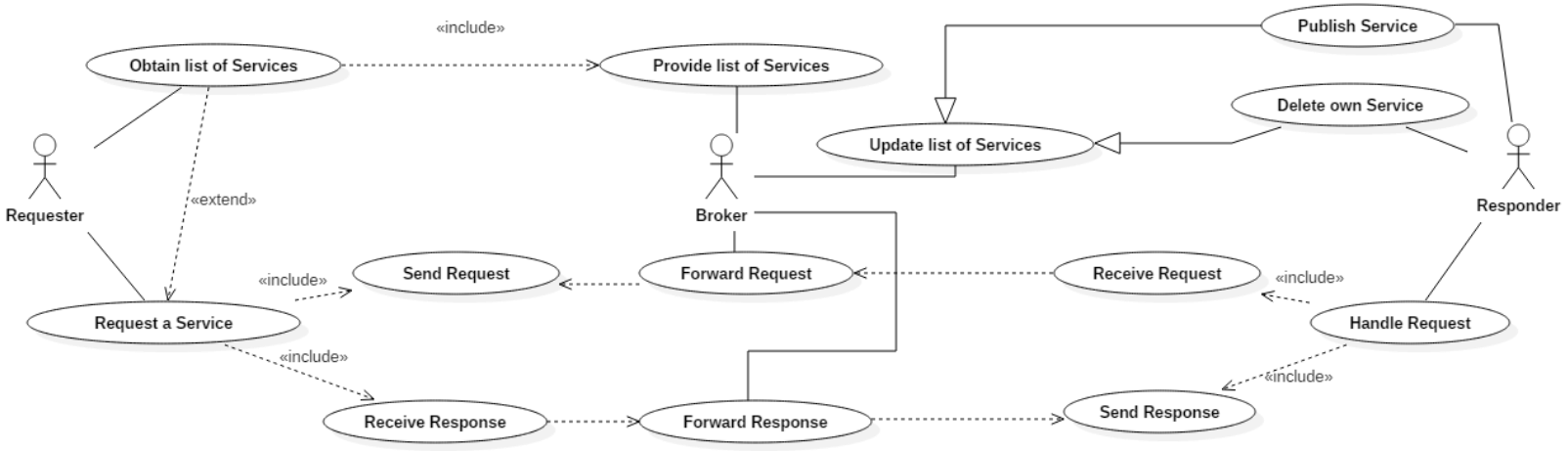
- Gruppo 1 -

Ghielmetti Nicolò
Quaglia Ennio
Zaffaroni Leonardo

Indice

Static Diagrams

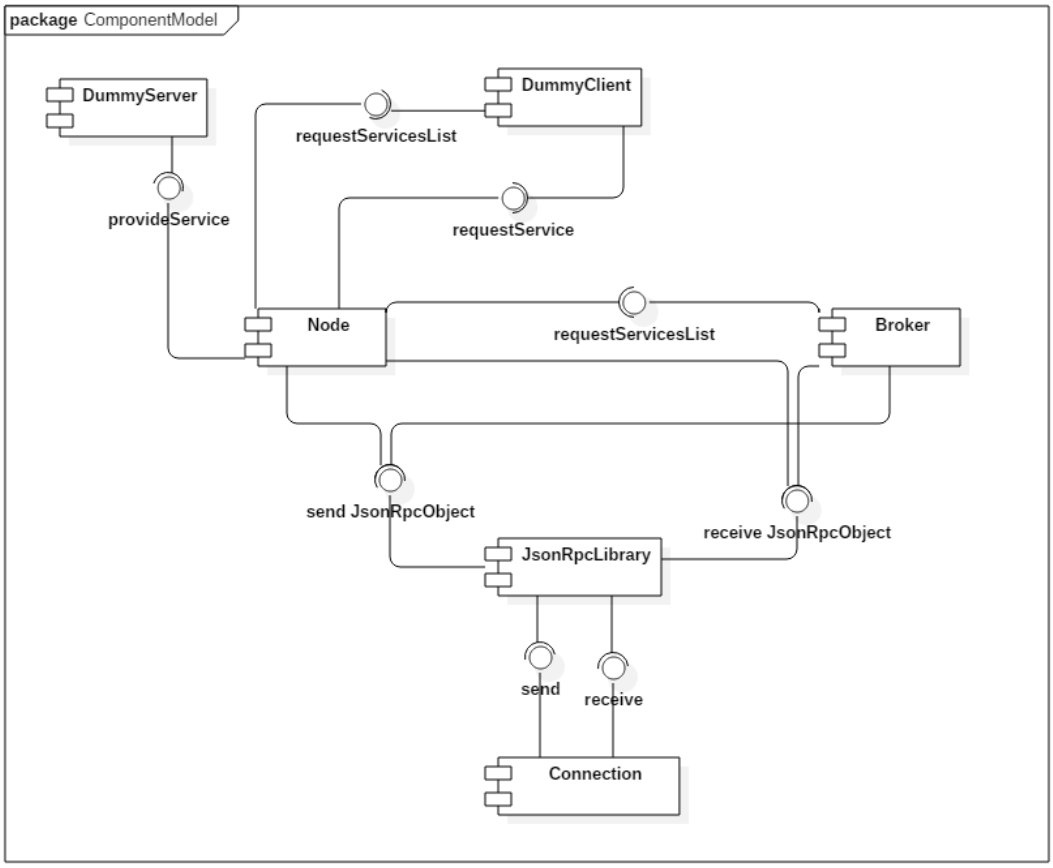
UseCase Diagram



Il sistema si può rappresentare attraverso tre attori:

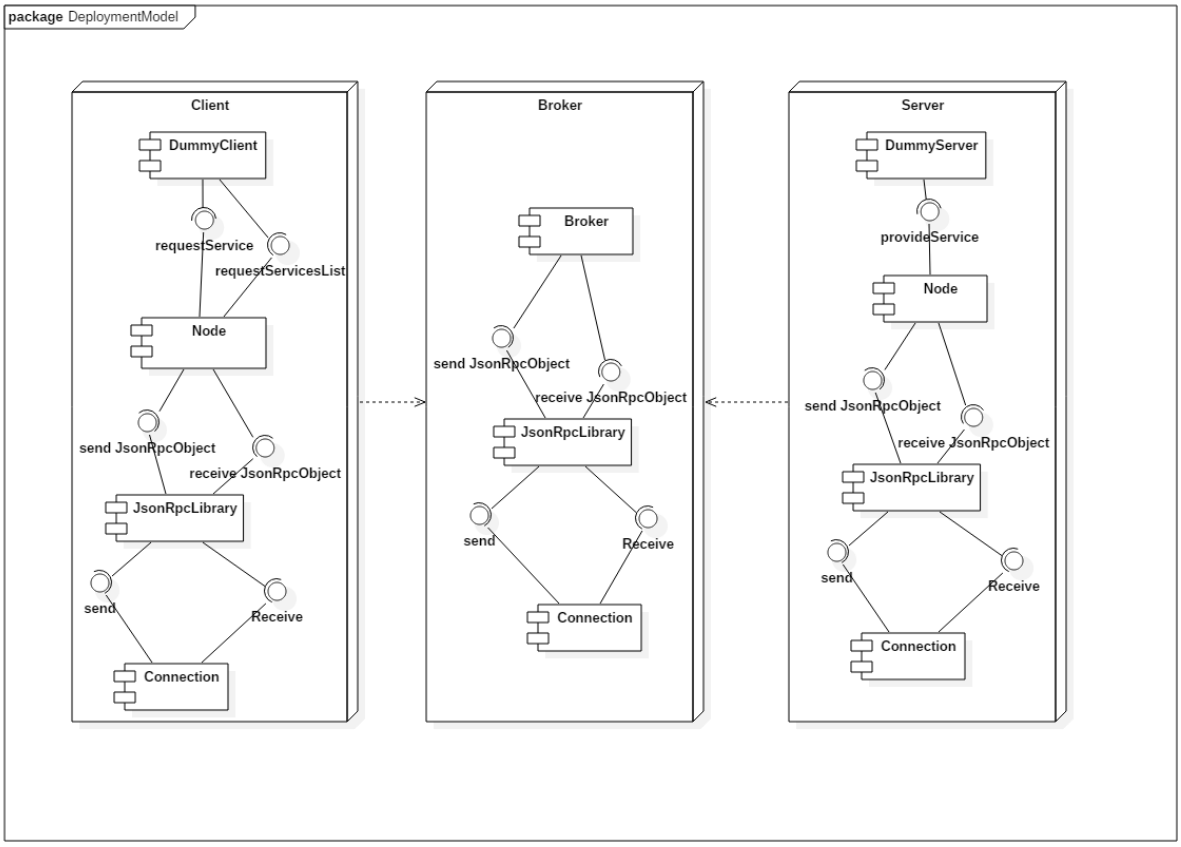
- **Requester**: è un utilizzatore di servizi. In particolare può:
  - richiedere una lista dei servizi disponibili al Broker.
  - richiedere l'invocazione di un servizio mediante la comunicazione trasparente realizzata dal Broker.
- **Responder**: è un fornitore di servizi. Ha il compito di:
  - gestire opportunamente le richieste, inviando risposte o errori adeguati.
  - gestire la sua presenza all'interno del sistema, aggiungendo e/o cancellando il proprio servizio.
- **Broker**: si occupa di gestire la comunicazione trasparente fra i vari **Requesters** e **Responders** presenti all'interno del sistema. In particolare si occupa di:
  - inoltrare richieste e risposte agli opportuni destinatari.
  - gestire la lista dei servizi disponibili mantenendola aggiornata.

Component Diagram



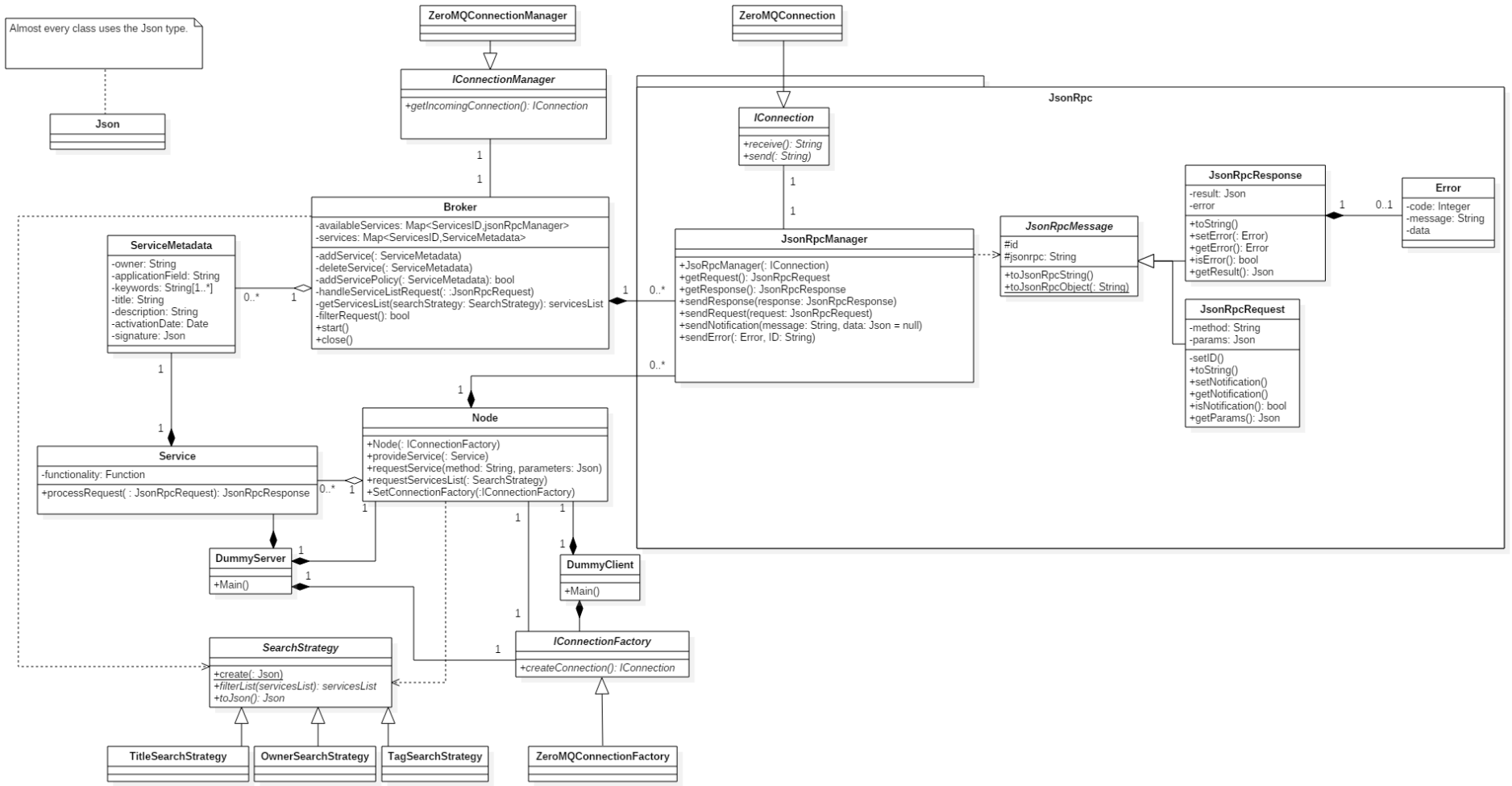
In questo diagramma vengono mostrati i moduli software in cui è diviso il sistema.
I componenti "*Dummy*" utilizzano le interfacce esposte dal componente **Node** per richiedere ed erogare servizi.
È importante evidenziare come i componenti **Node** e **Broker** comunichino utilizzando le stesse interfacce esposte dal componente **JsonRpcLibrary**: in questo modo è evidenziato che il protocollo di comunicazione per i moduli è *JsonRpc*.
Si può notare anche che sia la parte di erogazione sia la parte di richiesta di servizi sono fornite dal componente **Node**.
Il componente **JsonRpcLibrary** si basa sul componente **Connection** per inviare e ricevere i messaggi attraverso la rete.

Deployment Diagram



Il *deployment diagram* illustrato sopra rappresenta le macchine fisiche su cui possono essere eseguiti i vari moduli specificati nel *component diagram*. Come è possibile intuire dal *component diagram*, tutte e tre le macchine fisiche necessitano i componenti **JsonRpcLibrary** e **Connection**. Inoltre, come già evidenziato, **Server** e **Client** si basano entrambi su **Node**, utilizzando le opportune interfacce. Si noti che nonostante il deployment rappresenti tre macchine fisiche distinte, tutte e tre potrebbero essere messe in esecuzione su un unico calcolatore.

Class Diagram

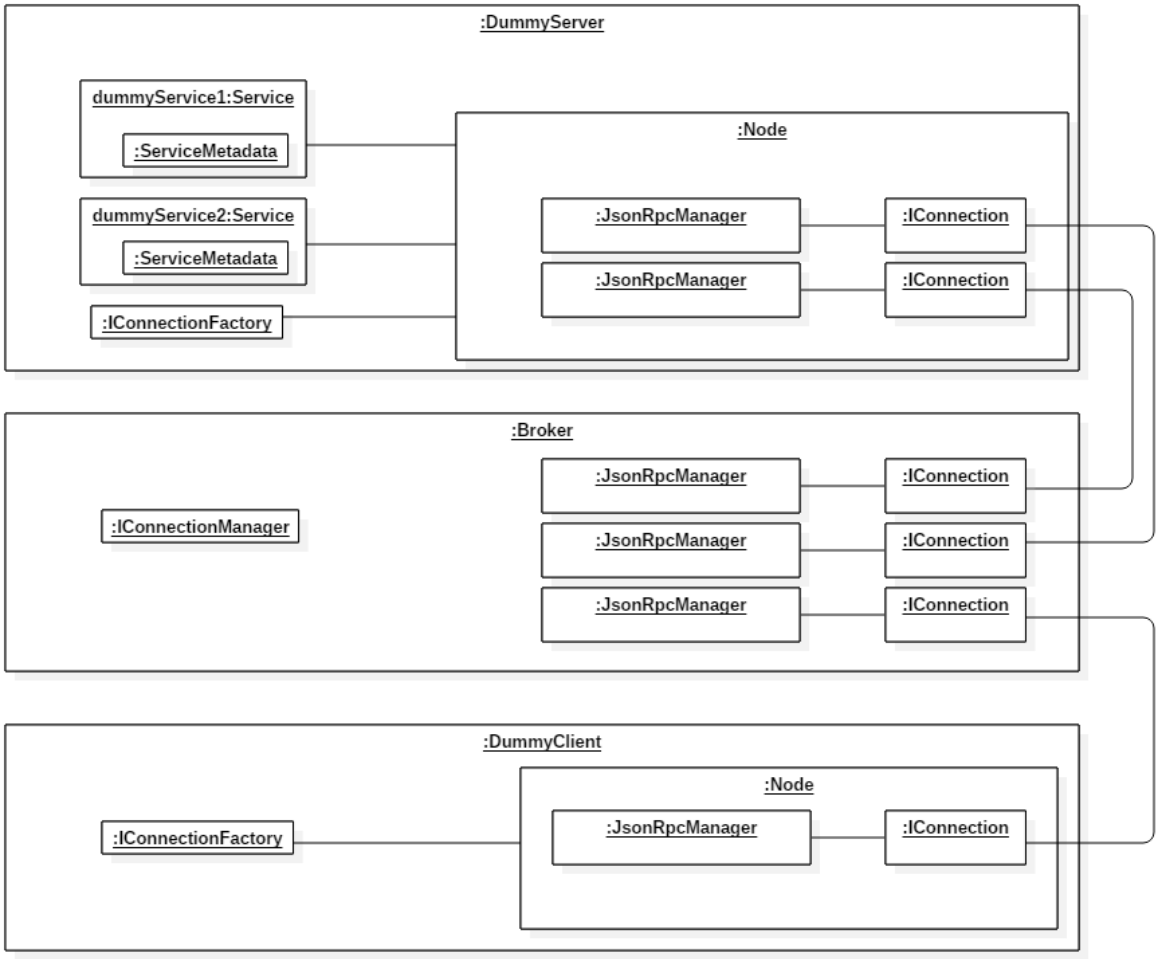


Commenti a pagina 7

Nel class diagram sono rappresentate le classi del tema comune (rappresentata dal package *JsonRpc*) e del tema B. Di seguito, alcune precisazioni:

- Al fine di ottenere l'applicazione e la libreria *JsonRpc* agnostiche rispetto al canale trasmissivo sono state previste le seguenti interfacce:
  - **IConnection**: è un'interfaccia che si occupa di gestire una comunicazione già instaurata. La presenza di questa interfaccia è necessaria affinché i metodi di ricezione e invio di risposte e richieste siano da implementare indipendentemente dalla libreria di comunicazione.
  - **IConnectionManager**: è l'interfaccia attraverso la quale il *Broker* ottiene per ogni connessione in arrivo una **IConnection**.
  - **IConnctionFactory**: è l'interfaccia attraverso la quale il **Node** può instaurare una connessione con il *Broker*.
- Per poter consentire la ricerca da parte del client, dei servizi disponibili nel sistema, è stata prevista nel modello la classe astratta ***SearchStrategy*** che permette l'indicazione di criteri di ricerca.
- Sebbene nella specifica JSON-RPC gli oggetti JSON-RPC Notification e JSON-RPC Error siano trattati, non abbiamo ritenuto opportuno prevedere le classi JsonRpcNotification e JsonRpcError poiché sono dei casi particolari rispettivamente di **JsonRpcRequest** e **JsonRpcResponse**. Per rispettare comunque la specifica sono stati previsti dei metodi per la realizzazione dei casi particolari sopracitati.
- **JsonRpcManager** è una classe che si occupa dello scambio di messaggi, secondo lo standard JSON-RPC, per una determinata **IConnection**.
- **addServicePolicy** è un metodo astratto della classe *Broker* la quale concretizzazione consentirà, in funzione dei metadati passati come parametro, di stabilire se il servizio può essere aggiunto nel sistema.
- Le funzionalità di cancellazione e registrazione di un servizio e di richiesta della lista dei servizi, sono trattate come tutti gli altri servizi, e quindi gestiti tramite **JsonRpcRequest** e **JsonRpcResponse.** L'unica differenza è che questi servizi saranno predefiniti e presenti nel *Broker*, che si occuperà di gestirli internamente senza inoltrarli ulteriormente.
- Le classi **DummyClient** e **DummyServer** sono programmi Client e Server d'esempio nel sistema.

Object Diagram



In questo diagramma è rappresentata una possibile configurazione degli oggetti.
Nello specifico si possono notare tre entità principali:

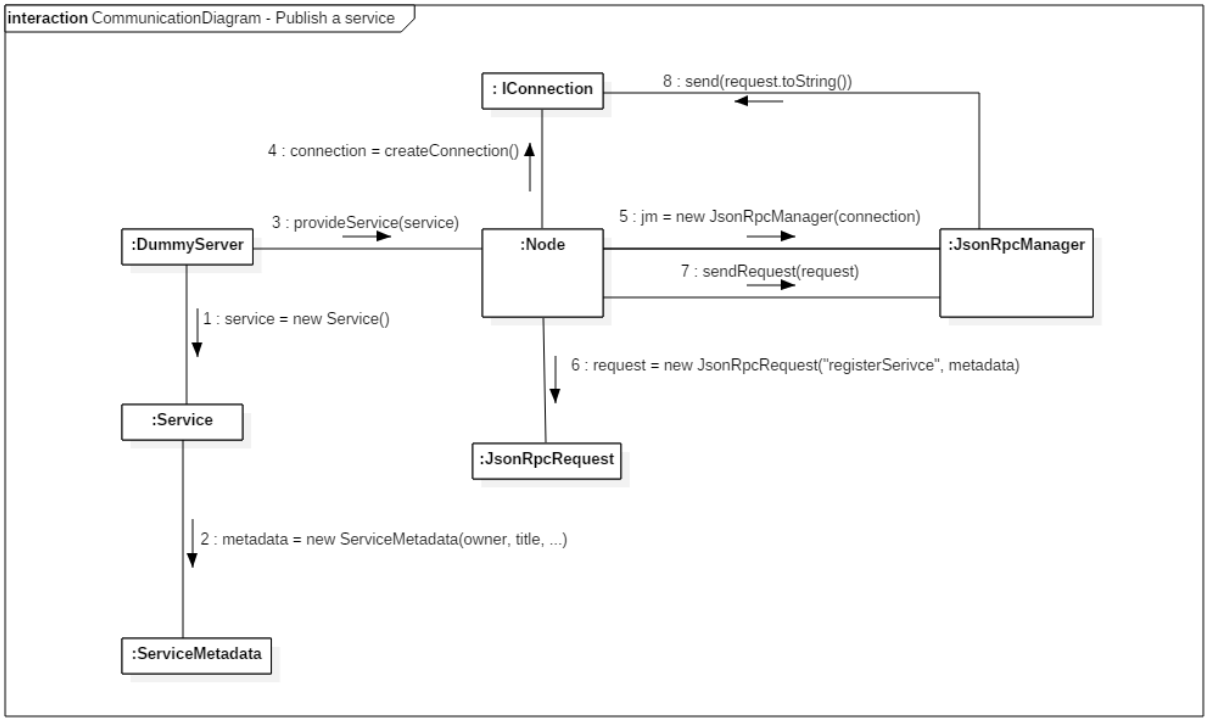
- **DummyServer** è un fornitore di due servizi,
- **DummyClient** è un utilizzatore di servizi,
- il **Broker**, che fa da tramite fra i due.

La situazione corrente si è raggiunto dopo questi passaggi:

- Il **DummyServer** ha correttamente creato e aggiunto i suoi due servizi al **Node**, che a sua volta li ha registrati nel **Broker**.
- Il **DummyClient** ha richiesto un servizio al **Broker**. La richiesta è ancora in corso, infatti sono ancora presenti il **JsonRpcManager** con la relativa **IConnection**: nel momento in cui la richiesta è stata soddisfatta, la connessione viene chiusa e il **JsonRpcManager** distrutto.

Dynamic Diagrams

Communication Diagram

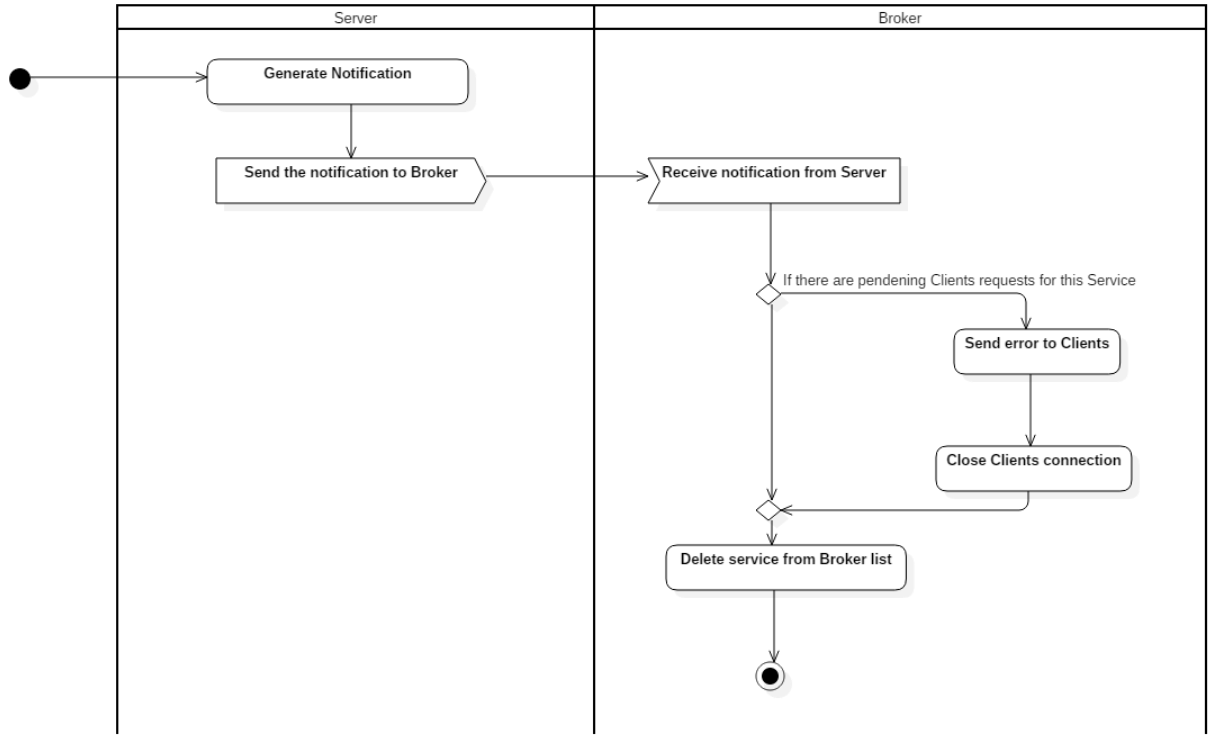


In questo communication diagram viene descritto lo scenario in cui un nodo *Server* pubblica un nuovo servizio da lui gestito.
Viene istanziato un oggetto di tipo **Service** che conterrà la funzione da svolgere per erogare il servizio ed al suo interno vengono creati i metadata (singola istanza della classe **ServiceMetadata**) necessari al *Broker* e ai *Clients* per identificare tale servizio.
Attraverso l'interfaccia pubblica **provideService*(…)*** l'oggetto **Node** può inviare la richiesta di pubblicazione del servizio al *Broker*:

- attraverso **IConncetion** crea una connessione verso il *Broker*.
- converte la richiesta istanziando una **JsonRpcRequest**, specificando l'intenzione di pubblicare un servizio.
- istanzia un oggetto della classe **JsonRpcManager**, il quale, attraverso **send(...)**, l'interfaccia pubblica di **IConncetion**, invia **JsonRpcRequest** al Broker.

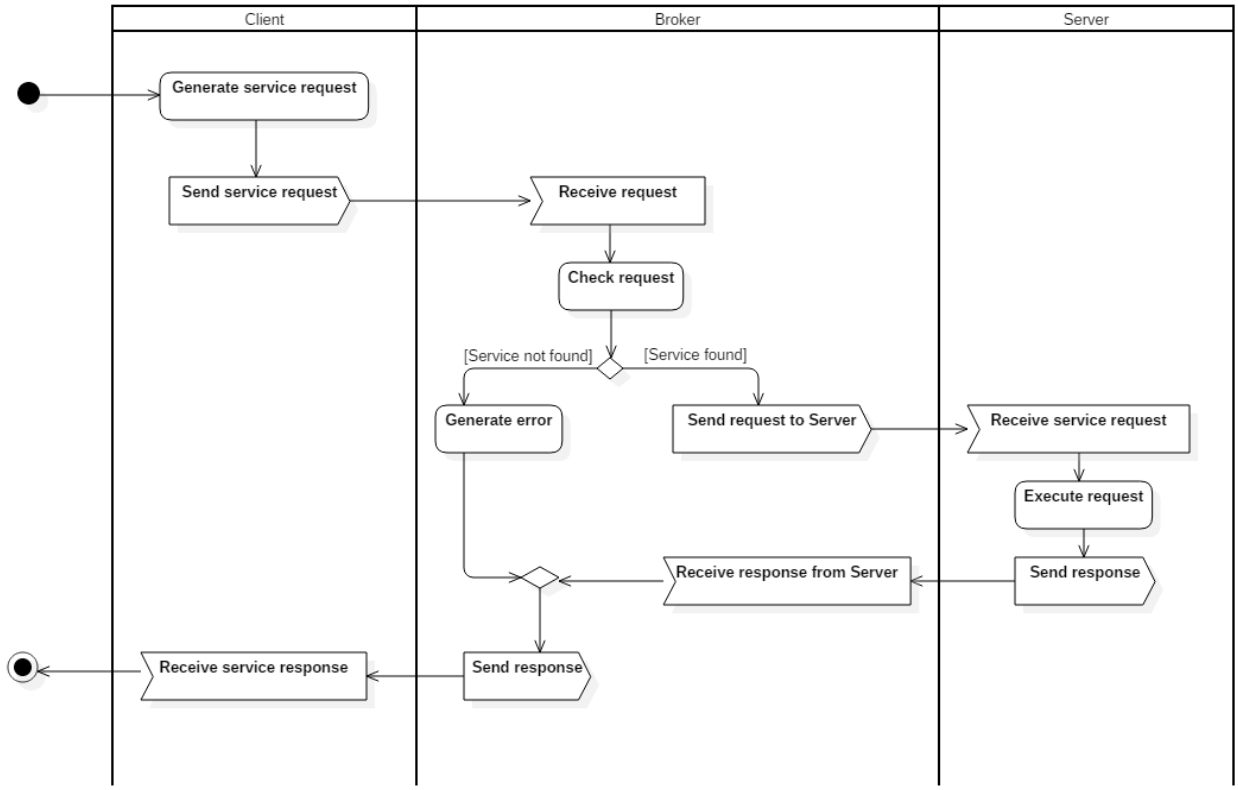
Activity Diagrams

Activity Diagram – Cancellazione di un servizio



Questo activity diagram descrive le operazioni che il sistema deve eseguire al fine di eliminare un servizio dalla lista dei servizi conosciuti dal **Broker**. Il **Server** può eliminare un servizio quando lo richiede più opportuno, *termina il servizio e notifica il* **Broker** *dell'eliminazione*. Il **Broker** *aggiorna la lista dei servizi eliminando il servizio non più disponibile*. Non è necessario, ai fini del progetto, che il **Broker** comunichi l'avvenuta eliminazione al **Server**.

Activity Diagram – Invocazione di un servizio

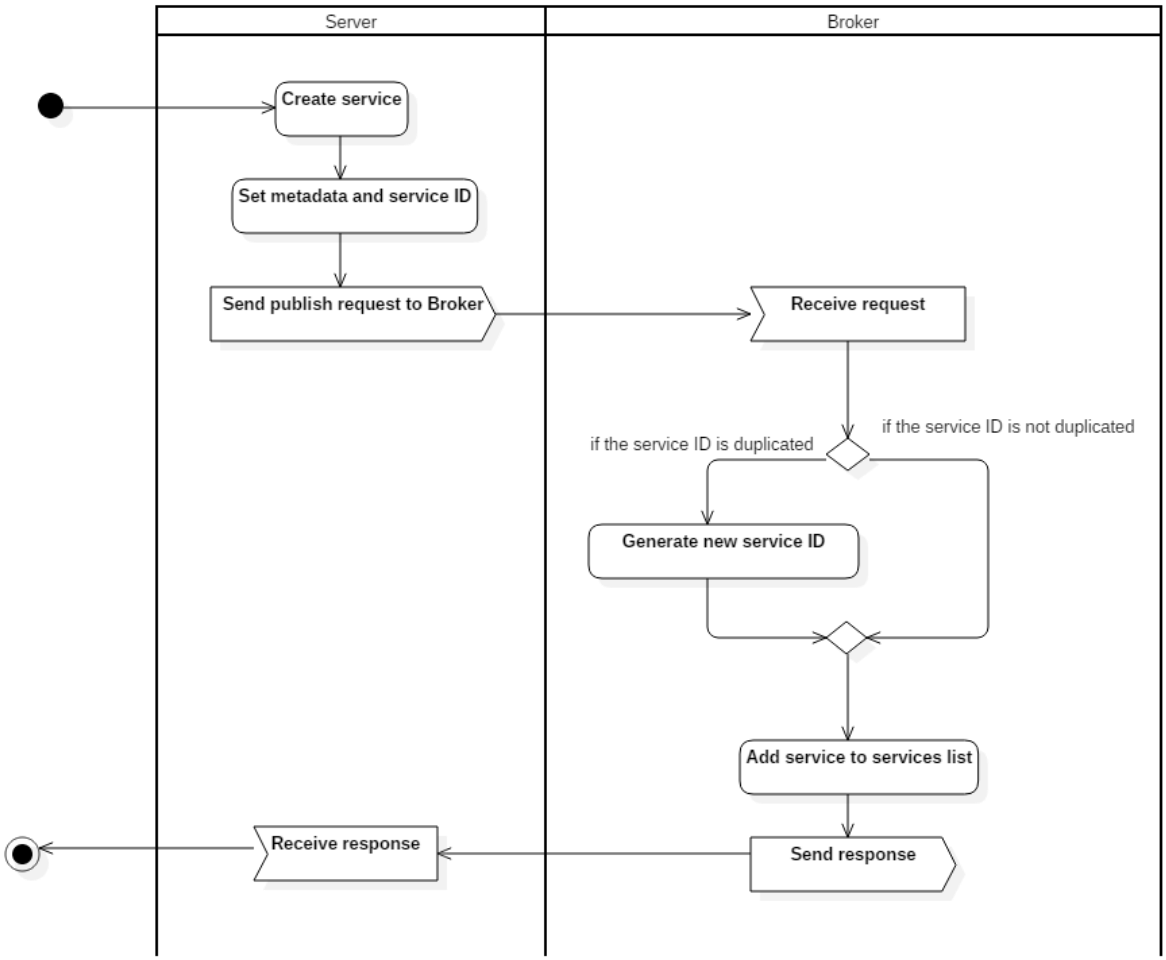


In questo activity diagram vengono mostrate le operazioni svolte , dall'intero sistema, al fine di erogare un servizio richiesto dal **Client**. Come prima operazione il **Client** *genera una richiesta di servizio* e la *inoltra al* **Broker** *restando in attesa di risposta.* Il **Broker** *verifica la presenza di tale servizio*:

- *se presente inoltra la richiesta al* **Server** che lo eroga rimanendo in *attesa di una sua risposta contenente il risultato del servizio richiesto.*
- altrimenti il **Broker** *genera una risposta di errore.*

In entrambi i casi *il* **Client** *riceve una risposta dal* **Broker** terminando correttamente l'operazione di richiesta di servizio.

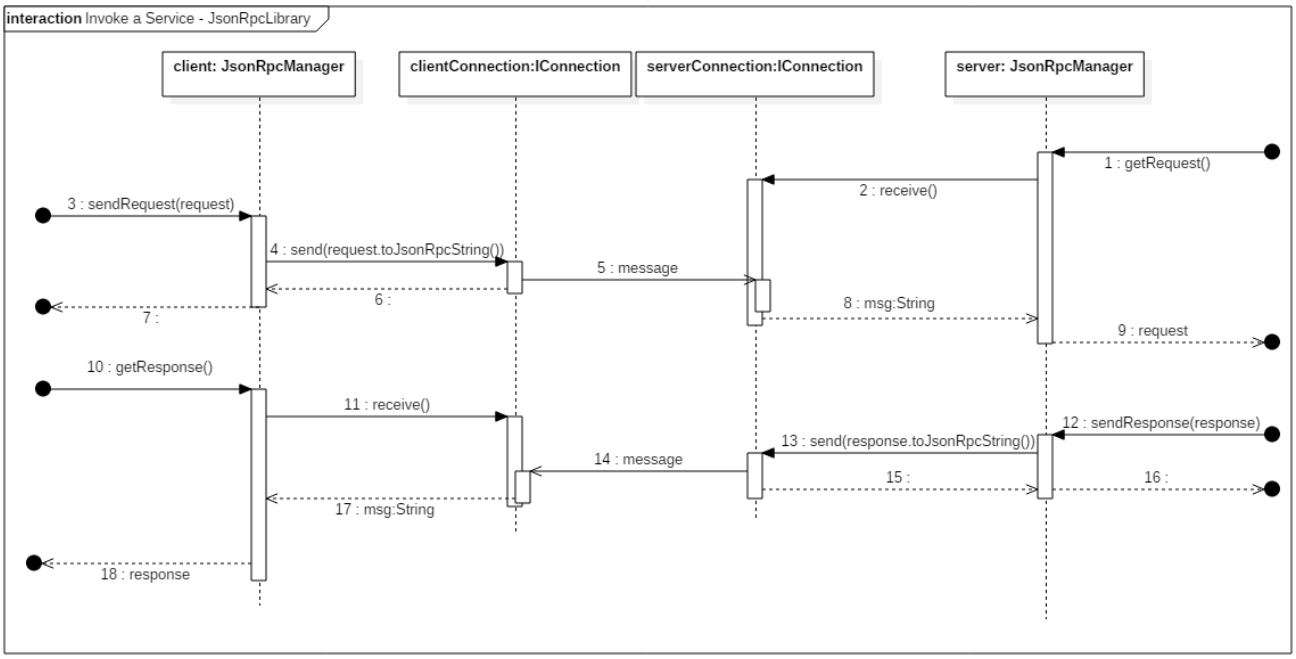
Activity Diagram – Pubblicazione di un servizio



In questo activity diagram viene specificato l'algoritmo che permette ad un fornitore di servizi (**Server**) la pubblicazione di un nuovo servizio da lui gestito. In particolare, viene mostrato come il **Broker** gestisce il caso in cui l'identificativo del servizio che si vuole aggiungere è già presente all'interno della lista dei servizi disponibili: se l'identificativo non è unico nella lista, il **Broker** lo modifica per renderlo univoco; altrimenti mantiene lo stesso identificativo. Infine il servizio viene aggiunto alla lista dei servizi disponibili. Il **Server** riceve una conferma di avvenuta pubblicazione.

Sequence Diagrams

Invocazione di un metodo Json-Rpc



Il *sequence diagram* sopraillustrato rappresenta il funzionamento di un'invocazione di procedura remota mediante Json-Rpc. Ricapitolando per passi principali:

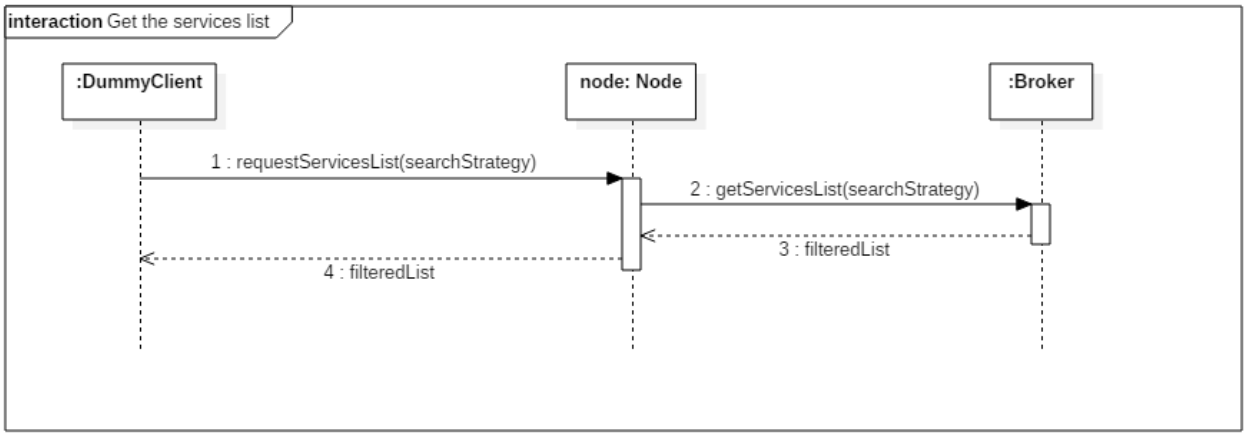
- (1,2): questi passi consistono nella preparazione del Server. Effettuato il passo 2 il Server è in attesa di una richiesta. Il suo processo sarà quindi bloccato in attesa di una connessione.
- (3,4,5,6,7) e (8,9): questi passi rappresentano l'invio di una richiesta da parte del Client (3,4,5,6,7) e la ricezione della stessa da parte del Server (8,9).
- (10,11): in modo duale rispetto a quanto descritto per i passi (8,9), il Client nei passi (10,11) si pone in attesa di una risposta.
- (12,13,14,15,16) e (17,18): questi passi consistono rispettivamente nell'invio di una risposta da parte del Server e nella ricezione da parte del Client della stessa.

Si noti che nei casi particolari di notifica ed errore il procedimento di invio è del tutto analogo, in particolare:

- **Notifica**: poiché il Client invia una Notifica quando non è interessato ad ottenere un feedback da parte del Server, il procedimento sarà identico ma limitato sino al punto 9 compreso. Il Client infatti non si porrà in attesa di alcuna risposta e in modo duale il Server non si preoccuperà di fornirne una al Client.
- **Errore**: poiché l'errore è incapsulato all'interno di una JsonRpcResponse il procedimento di inoltro di un errore da parte del Server e la ricezione dello stesso da parte del Client sono identici a quelli descritti nel sequence diagram sopraillustrato.

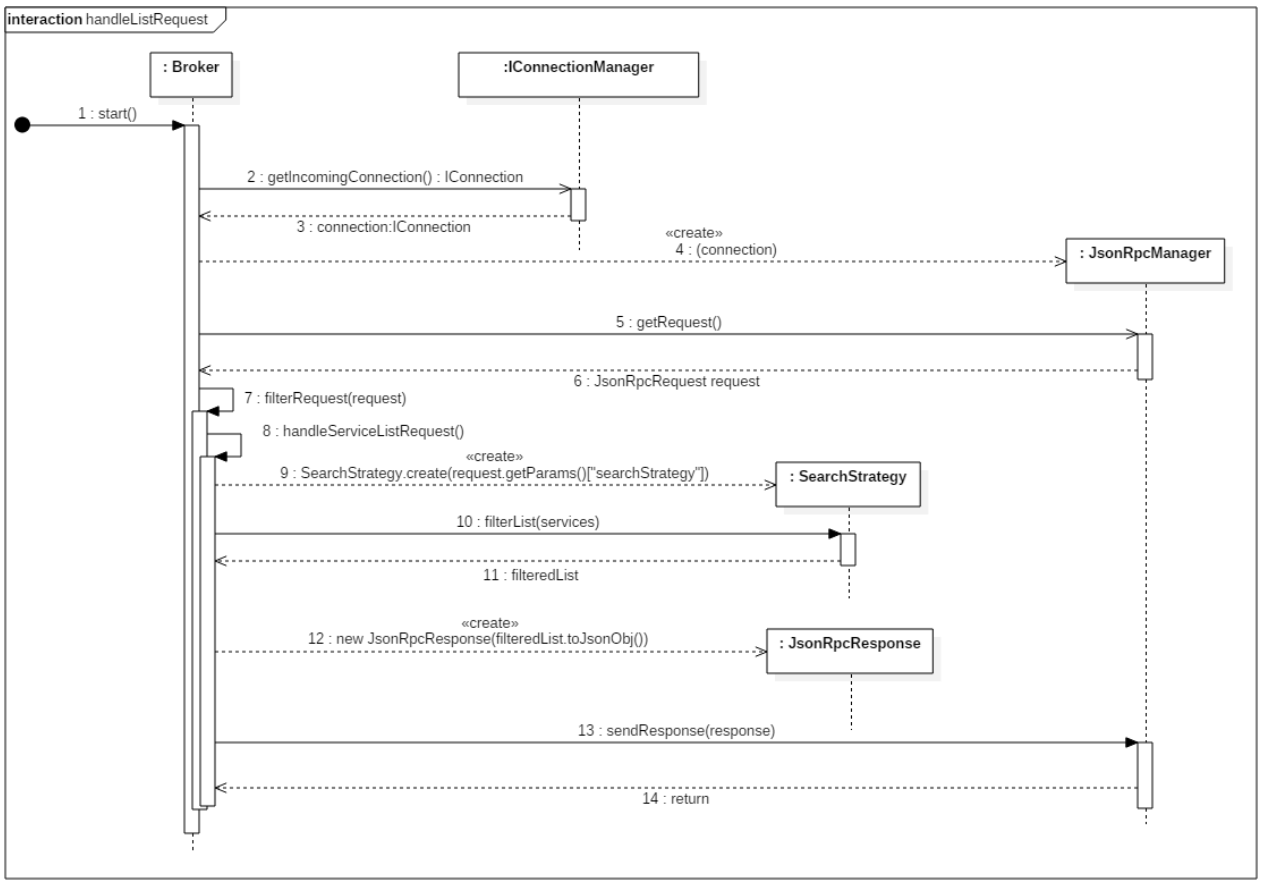
Infine, è importante evidenziare nuovamente che a livello applicativo questo procedimento è comune per qualsiasi servizio si voglia invocare all'interno del sistema, compreso il servizio di richiesta della lista dei servizi disponibili che è gestito dal Broker.

Richiesta lista dei servizi – livello applicativo



In questo diagramma si può vedere come utilizzare la classe **Node** per ottenere la lista dei Servizi attivi registrati nel **Broker**.
Si possono richiedere liste di servizi diverse in base al criterio di ricerca selezionato, indicato tramite il parametro **searchStrategy**. Tramite questo parametro sarà quindi possibile richiedere tutti i servizi aventi un nome particolare, o tutti quelli registrati da un particolare fornitore piuttosto che tutti quelli aventi dei particolari tag.
A livello implementativo, la **searchStrategy** verrà passata al **Broker** sotto forma di oggetto **Json** all'interno dei parametri previsti in *JsonRpc*, quindi abbiamo previsto un metodo per convertirla in **Json** ed un *factory method* statico che possa fare il procedimento inverso (come si può vedere nel class diagram).
Nel diagramma successivo si spiega nel dettaglio come viene gestita una richiesta del genere dal Broker.

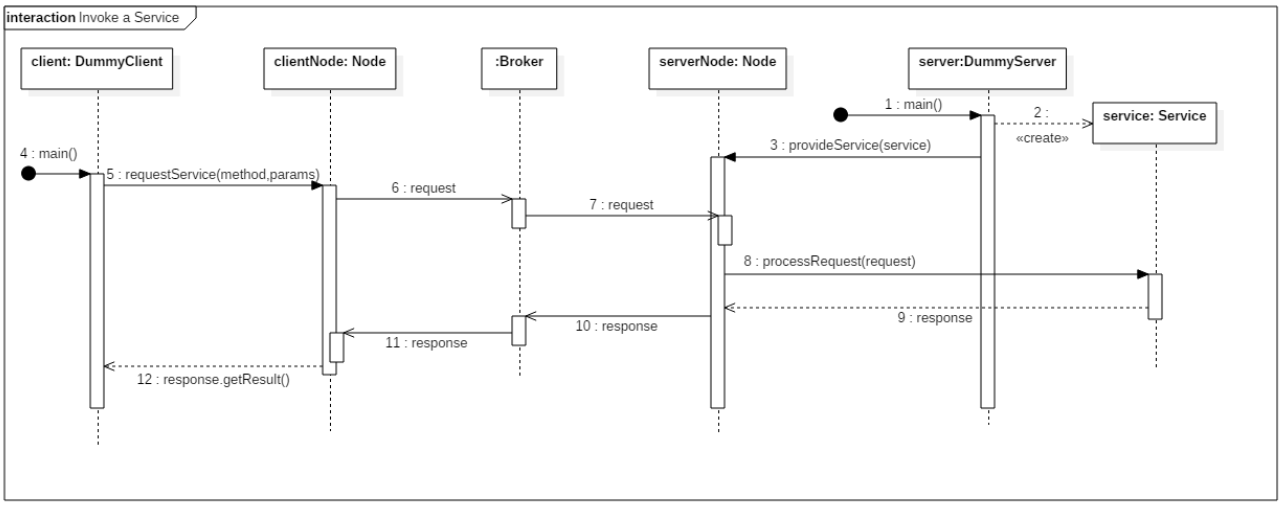
Gestione della richiesta della lista dei servizi – dettaglio broker



A livello di Broker la richiesta della lista dei servizi viene gestita come una comune richiesta di servizio con la differenza che al posto che essere inoltrata dal Broker ad un server di destinazione viene gestita autonomamente dal Broker.
I macropassaggi svolti dal **Broker** per inviare la lista dei servizi disponibile a un **Node** che ne fa richiesta sono i seguenti:

- **Start del Broker e gestione delle connessioni in ingresso** : dopo essere stato avviato, il *Broker* gestirà le richieste e le risposte entranti attraverso il *IConnectionManager* il quale si occuperà di fornire la *IConnection* opportuna al *Broker*.
- **Gestione delle rpc attraverso jsonrpcmanager**: una volta ottenuta la Connection opportuna, il *Broker* crea un *JsonRpcManager* passando come paramentro una *IConnection*. In questo modo il Broker potrà gestire le *request* entranti utilizzando il metodo *getRequest()* che fornirà come valore di ritorno la request entrante ricevuta.
- **Filtraggio della request e sua gestione** : una volta ricevuta la *request* entrante il Broker può controllare se si tratta di una *request* da inoltrare oppure se è relativa ad un servizio messo a disposizione da lui. Nel caso considerato, viene invocata la funzione *handleServiceListRequest* poiché la richiesta ricevuta è finalizzata all'ottenimento dei servizi disponibili.
- **Gestione della Strategy ed effettuazione della risposta** : per stabilire secondo quale criterio restituire al *Node* la lista dei servizi disponibili, il Broker utilizza la strategy ricevuta come parametro nella *request* per creare la lista filtrata secondo i criteri specificati dal *Node*. Una volta ottenuta la lista filtrata il *Broker* la impacchetta adeguatamente all'interno del parametro *Result* della *JsonRpcResponse*.

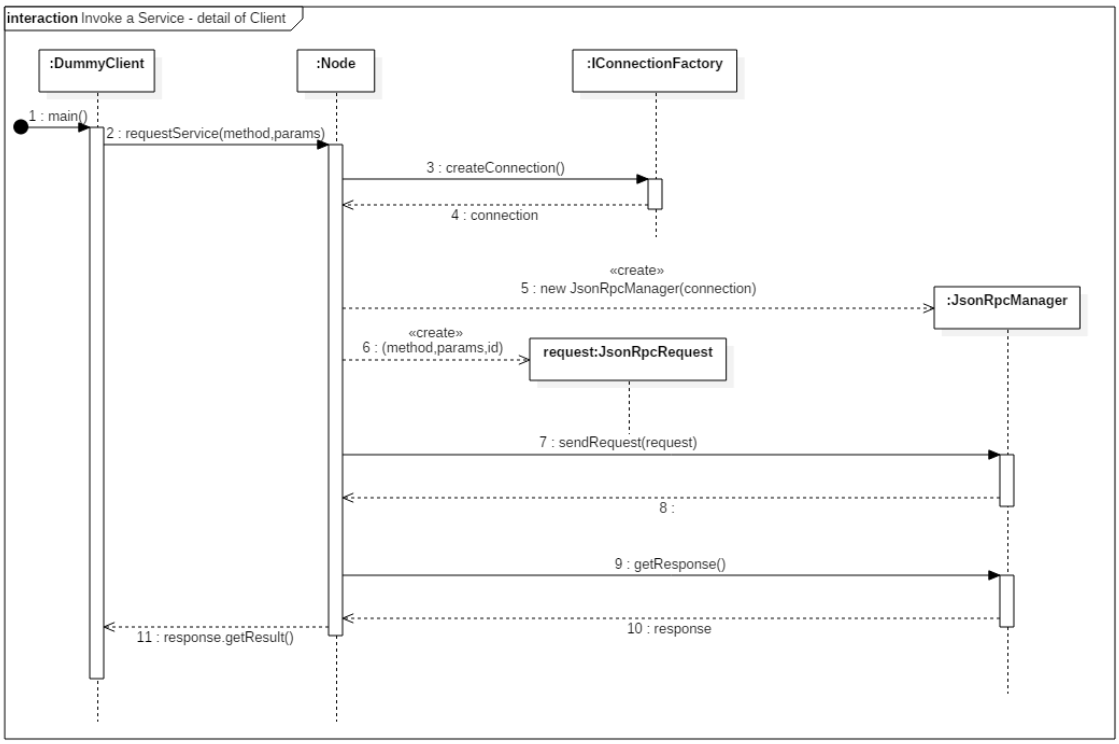
Invocazione di un Servizio – livello applicativo



Da questo diagramma si descrive l'utilizzo dei metodi forniti dalla classe Node, sia lato client che lato server. Nello specifico:

- Il metodo provideService() si occupa di rendere disponibile un servizio, a livello d'implentazione questo metodo aprirà un thread in modo che ogni singolo servizio sia gestito indipendentemente.
- Il metodo requestService() che si occupa di invocare un servizio e ritornarne il risultato.

Invocazione di un servizio – dettaglio client



La possibilità di invocare un servizio viene fornita al livello superiore sotto forma di una semplice funzione, **requestService**, che accetta l'identificativo del metodo da invocare e i parametri impacchettati in un oggetto **Json**, e ritorna i risultato anch'esso sotto forma di **Json**.
I passaggi che vengono svolti per invocare il servizio sono:

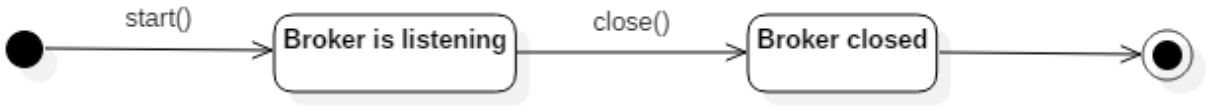
1. Instaurazione di una connessione col broker, gestita dal **IConnectionFactory** (passaggi 2,3,4).
2. Costruzione del **JsonRpcManager** con la **IConnection** appena ottenuta (passaggio 5).
3. Costruzione della **JsonRpcRequest** con i parametri passati dall'utente e opportuni dati aggiuntivi(come per esempio il campo ID, generato al momento) (passaggio 6).
4. Invio della richiesta (passaggi 7,8).
5. Ricezione della risposta, da cui viene estratto il campo **result** e ritornato all'utente sotto forma di oggetto **Json** (passaggi 9,10,11).

Note:

- Tutti i metodi di ricezione, quindi **getResponse()** del **JsonRpcManager**, **receive()** di **IConnection** e lo stesso **requestService(…)**, sono stati pensati come bloccanti: fintanto che non hanno un valore da ritornare bloccano l'intero flusso di istruzioni, sfruttando il fatto che le code di messaggi sono già gestite da *ZeroMQ*.
  Per questo motivo si sfrutteranno i thread, nello specifico ogni servizio fornito all'interno del **Node** sarà gestito da un thread personale, con relativo **JsonRpcManager** e **IConnection**.
- La **IConnectionFactory** potrebbe gestire le **IConnection**(s) tramite una *ObjectPool*, visto che le richieste hanno comunque un tempo di esecuzione limitato (nel caso una risposta non arrivi entro un certo tempo, si genererà un errore di timeout).

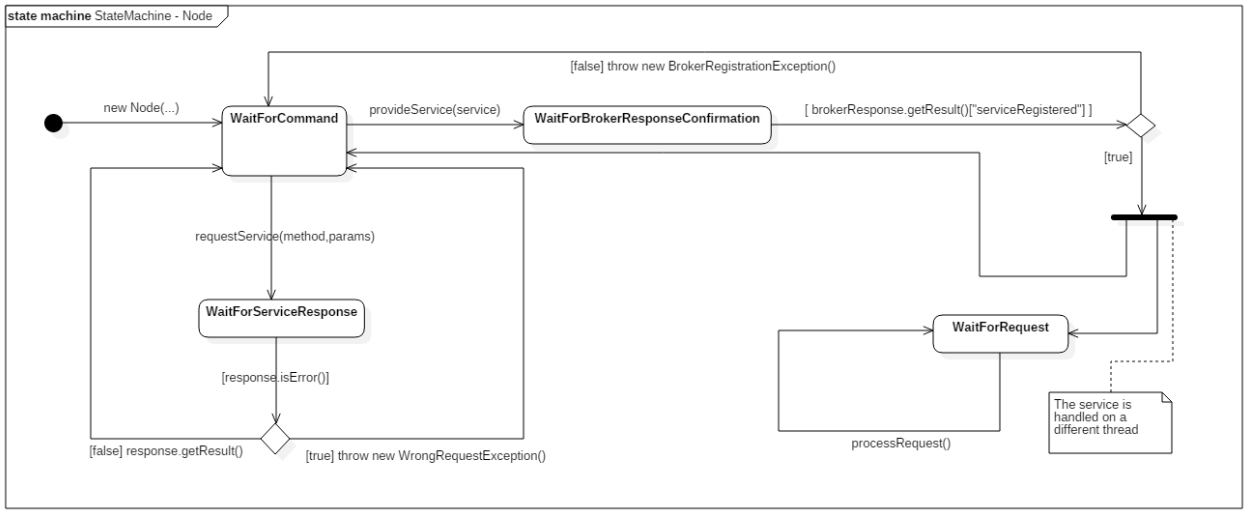
State Diagrams

State Diagram – Broker



Il **Broker** ha fondamentalmente due stati, quello in cui è attivo e accetta e gestisce le richieste, e quello in cui è disattivato.
La gestione delle richieste è specificata nell'*activity diagram* riguardante *l'invocazione di un servizio* (a pagina 11), nel caso di richieste dirette a server esterni; e nel *sequence diagram* riguardante la *gestione della richiesta della lista dei servizi* (a pagina 15), nel caso di richieste dirette ai servizi interni al *Broker*.

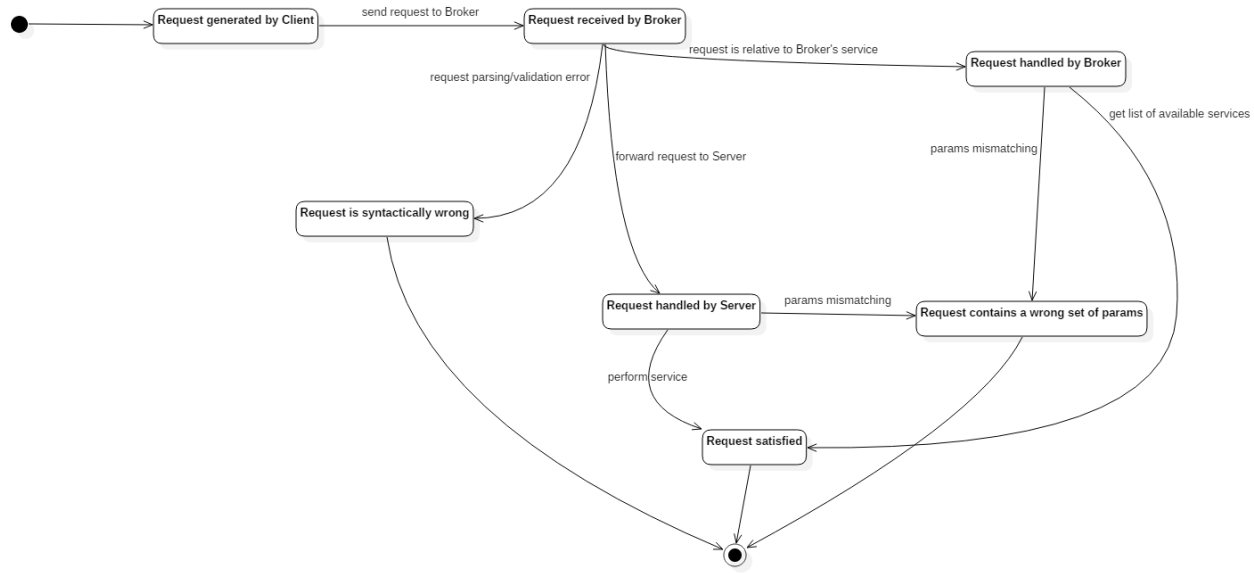
State Diagram – Node



La macchina a stati di cui sopra evidenzia due macro stati in cui il Node si può trovare. Questi due set di stati possono essere individuati dagli archi uscenti dallo stato **"WaitForCommand"**:

- **waitForServiceResponse**: il Node si trova in questo stato subito dopo aver inviato la richiesta relativa ad servizio. è perciò in attesa della risposta da parte del *Node* a cui quel servizio è associato (in realtà la risposta verrà inoltrata dal *Broker*). Quando riceve la risposta, se non si tratta di un errore, il *Node* legge il risultato della risposta e torna nello stato iniziale (**"WaitForCommand"**). Altrimenti solleva un'eccezione relativa all'errore ricevuto e torna nello stato iniziale (**"WaitForCommand"**).
- **WaitForBrokerResponseConfirmation**: il Node si trova in questo stato non appena gli è stato associato un servizio. In questo stato il *Node* è in attesa di una risposta da parte del *Broker* che gli da conferma (o meno) dell'avvenuta registrazione del servizio all'interno del sistema. Se il *Broker* conferma l'avvenuta registrazione del servizio il *Node* creerà un thread per la gestione delle richieste mentre il thread principale tornarerà nello stato **"WaitForCommand"** iniziale. Altrimenti verrà sollevata un'eccezione relativa all'errore che il *Broker* avrà incapsulato nella sua risposta.

State Diagram – Request



La macchina a stati rappresentata mette in risalto tutti gli stati in cui una richiesta si trova a partire dalla generazione da parte del client fino al soddisfacimento della stessa. Una volta creata e inoltrata al *Broker*, quest'ultimo controllerà se la gestione della richiesta dev'essere a carico suo. Questo capita nel caso di una richiesta della lista dei servizi disponibili (vedasi i due sequence diagrams associati). É bene evidenziare che già a questo livello della comunicazione il *Broker* effettua il controllo della formattazione della richiesta: nel caso non fosse nel formato *Json* o non rispetti lo standard *JsonRPC* la richiesta verrà considerata come sintatticamente errata. Sia che la richiesta può essere gestita dal *Broker*, sia che la richiesta debba essere inoltrata ad un servizio, verrà ritenuta soddisfatta qualora non vi fossero errori nei parametri incapsulati nella risposta stessa, utili al servizio (che sarà fornito nel *Broker* o in un *Node*) per fornire un risultato.