Politecnico di Milano
Polo Territoriale di Como

Prova finale di Ingegneria del Software
Documento di progetto – Implementazione Java e casi di test JUnit

- Gruppo 1 -

Ghielmetti Nicolò
Quaglia Ennio
Zaffaroni Leonardo

Indice

Ambiente di sviluppo 3
- IDE 3
- JDK & JRE 3

Moduli 3
- Broker 3
- Node 3
- ConnectionInterfaces 3
- SearchStrategy 3
- Service 3
- Tests 4
- Logger 4
- JsonRpcLibrary 4
- ZeroMQImplementation 4

Artefatti 4

Dipendenze 4
- Broker 4
- Node 5
- SearchStrategy 5
- Service 5
- JsonRpcLibrary 6
- ZeroMQImplementation 6

Tests 6

Diagrammi aggiornati 7
- Class diagram 7
- Component diagram 8
- Deployment diagram 9

Ambiente di sviluppo

IDE

Lo sviluppo del progetto (Tema B e libreria JsonRPC) è stato effettuato mediante l'utilizzo di Intellij IDEA 2017.2.5. La collaborazione tra i membri del gruppo è avvenuta attraverso l'utilizzo del plug-in apposito di Git per Intellij IDEA.

JDK & JRE

JDK : 1.8.0_151-b12
JRE : 1.8.0_152 .
JVM: OpenJDK fornito da JetBrains s.r.o (incluso nel pacchetto di download dell'IDE).

Moduli

La divisione per moduli delle classi è utile per la creazione degli artefatti e per la corretta condivisione di codice da parte di più moduli attraverso le dipendenze. Strutturando in questo modo il codice si è riusciti ad evitare repliche e conseguente rischio di inconsistenza del codice.

Broker

Il modulo Broker si compone di due classi: "Broker.java" per la realizzazione della logica del Broker e "DummyBroker.java" per l'invocazione ed esecuzione del Broker.

Node

Similmente a quanto scritto per il modulo Broker, anche Node si compone di due classi: "Node.java" per la realizzazione di tutte le funzionalità che deve possedere un Node (funzionalità Client e Server) e "DummyNode.java" che ne è un esempio di applicazione delle funzionalità esposte in Node.

ConnectionInterfaces

Il modulo ConnectionInterfaces contiene tutte le interfacce utilizzate per la definizione del canale trasmissivo astraendo dalla particolare implementazione. Inoltre include la classe "TimeoutException.java" poiché la definizione del timeout di connessione tra Node e Broker, per non dipendere dalla particolare implementazione scelta, è stata previsto a questo livello. Questo permette al modulo JsonRpcLibrary di rimanere agnostico rispetto al canale di comunicazione.

SearchStrategy

Il modulo SearchStrategy contiene le definizioni delle diverse strategie applicabili dal Node e dal Broker per stabilire un criterio di ricerca dei servizi disponibili nel sistema. La classe "SearchStrategy.java" è la rappresentazione astratta delle possibili concretizzazioni. Le altre tre classi rappresentano tre implementazioni di strategie di ricerca dei servizi (derivando da "SearchStrategy.java" è possibile creare altri criteri di ricerca basati sui ServiceMetadata).

Service

Il modulo Service contiene le classi necessarie per la creazione ed erogazione di un servizio associato ad un Node oppure, in casi particolari, interno al Broker. È presente un'interfaccia "IServiceMethod.java" che serve per specificare l'algoritmo che deve computare il servizio nel Server per mandare una risposta al Client richiedente.

Vi è inoltre la classe "JsonRpcCustomError.java" che contiene la definizione degli errori riservati al livello applicativo (vedasi http://www.jsonrpc.org/specification sezione "Error Object"). Non è presente nel modulo JsonRpcLibrary poiché la rilevazione di questi errori è lasciata al livello applicativo.

Tests
Il modulo Tests contiene i casi di test delle classi del progetto sviluppati con JUnit4.

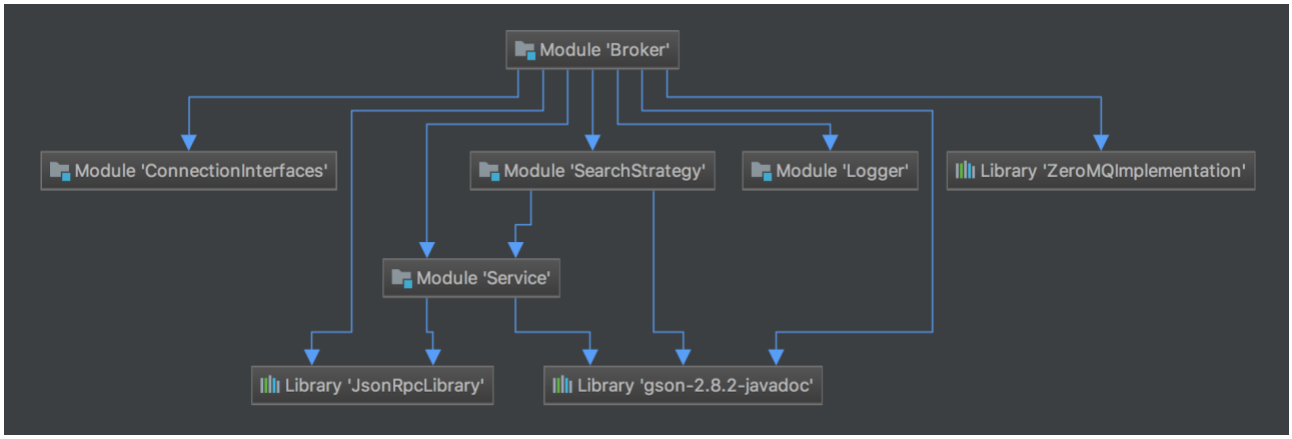
Logger
Il modulo Logger contiene la classe attraverso la quale si può ottenere un riscontro dell'avanzamento dei processi in modo centralizzato. In particolare attraverso l'attributo booleano "DEBUG" si possono attivare/disattivare le notifiche a video emesse dai processi.

JsonRpcLibrary
Il modulo JsonRpcLibrary implementa lo standard JsonRpc 2.0 secondo le specifiche (http://www.jsonrpc.org/specification). Le classi presenti descrivono Request e Notification, Response ed Error (ad eccezione di quelli "custom" descritti nella sezione Service) e relativi Batch. La classe JsonRpcManager si occupa di gestire in generale l'invio di Request, Response e Batch ed effettua i controlli di formattazione delle stringhe necessari per garantire il soddisfacimento dello standard JsonRpc 2.0.

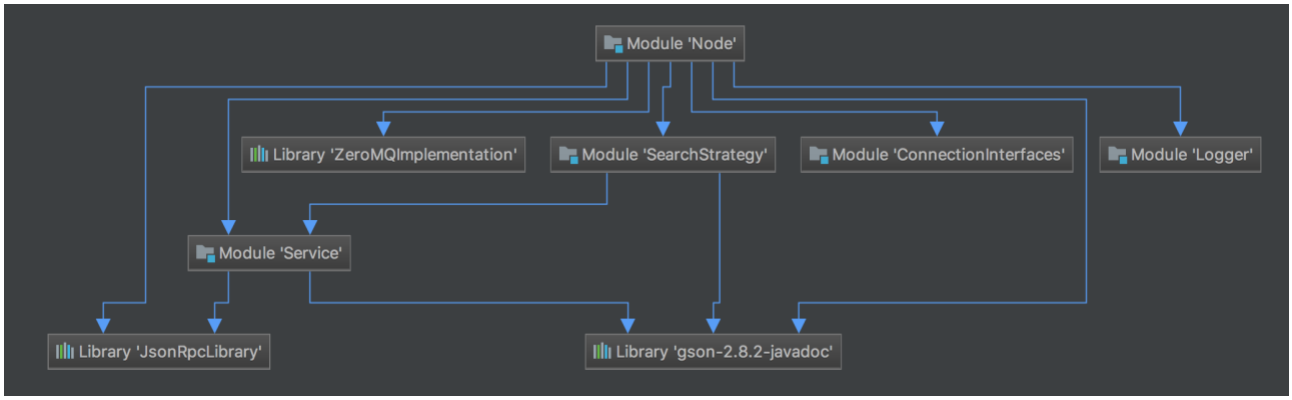
ZeroMQImplementation
Il modulo ZeroMQImplementation aggrega le implementazioni delle interfacce presenti nel modulo ConnectionInterfaces utilizzando le librerie di comunicazione ZeroMQ.

Artefatti
Gli artefatti prodotti attraverso il comando "Build → Build Artifacts…" sono "Broker.jar", "Node.jar", "ZeroMQImplementation.jar" e "JsonRpcLibrary.jar", presenti nella directory di progetto out/artifacts. "Node.jar" e "Broker.jar" sono eseguibili da terminale attraverso il comando "java -jar Node.jar", una volta posizionato il riferimento alla directory nella cartella out/artifacts di progetto. "ZeroMQImplementation.jar" e "JsonRpcLibrary.jar" sono due librerie sulle quali si basa il codice di "Broker.jar" e "Node.jar". Il bytecode del codice di queste due librerie viene importato nei due file "Broker.jar" e "Node.jar" (non c'è una dipendenza a runtime ma a tempo di compilazione).
Sono inoltre presenti altre due librerie importate nel progetto utilizzando il tool di Maven: "jeromq-0.4.3" e "gson-2.8.2" che racchiudono rispettivamente un'implementazione dei socket attraverso lo standard TCP e un'implementazione dello standard Json.

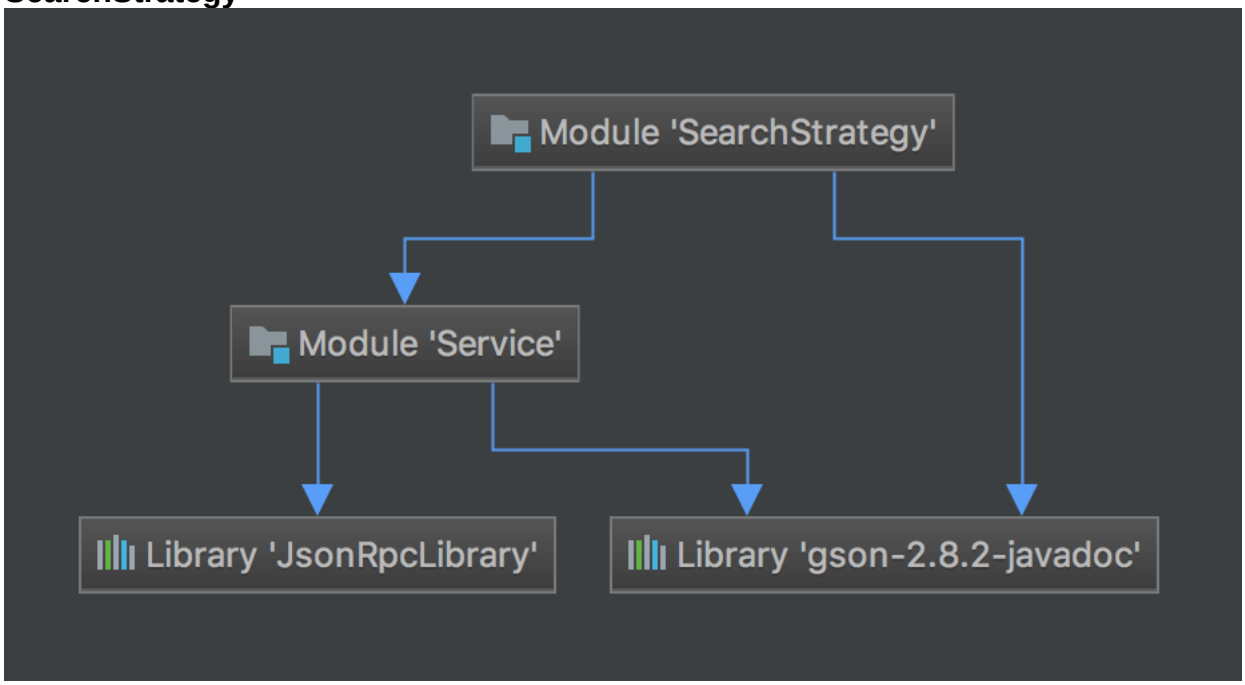
Dipendenze
Di seguito sono mostrate le dipendenze fra moduli e artefatti (sono stati omessi i moduli privi di dipendenze). Si noti come il modulo JsonRpcLibrary dipende dal modulo ConnectionInterfaces e non dal particolare canale di comunicazione (ZeroMQImplementation). In questo modo ogni implementazione che implementi le interfacce presenti in ConnectionInterfaces può essere utilizzata dalla libreria per comunicare nella rete.

Broker


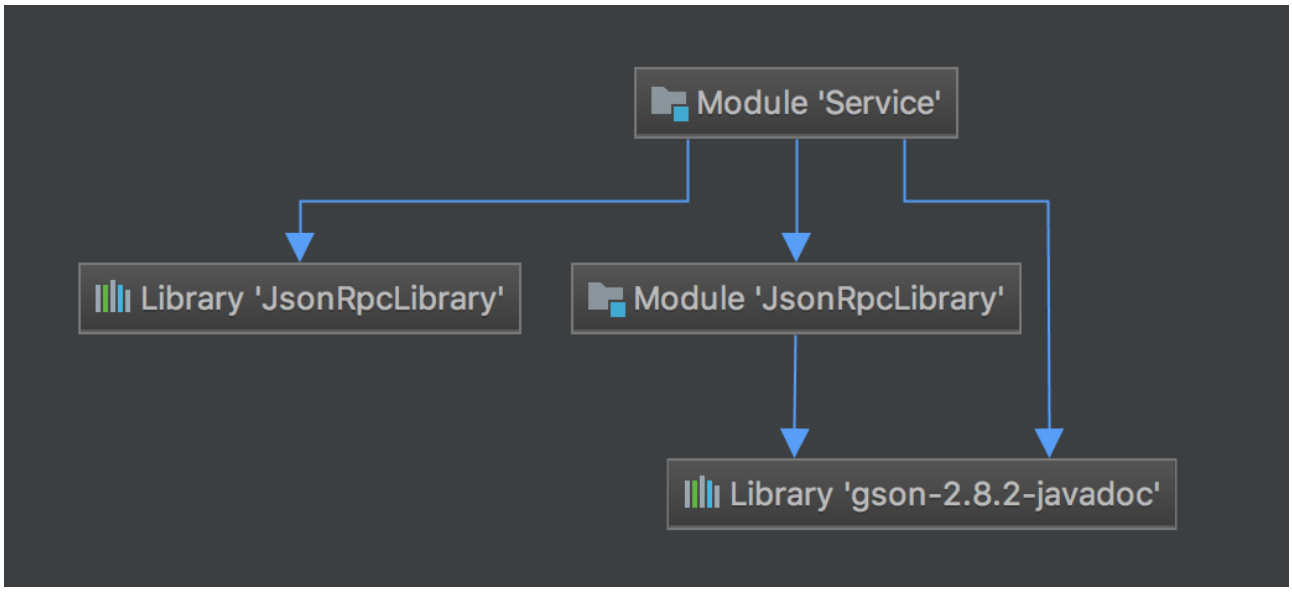
Node



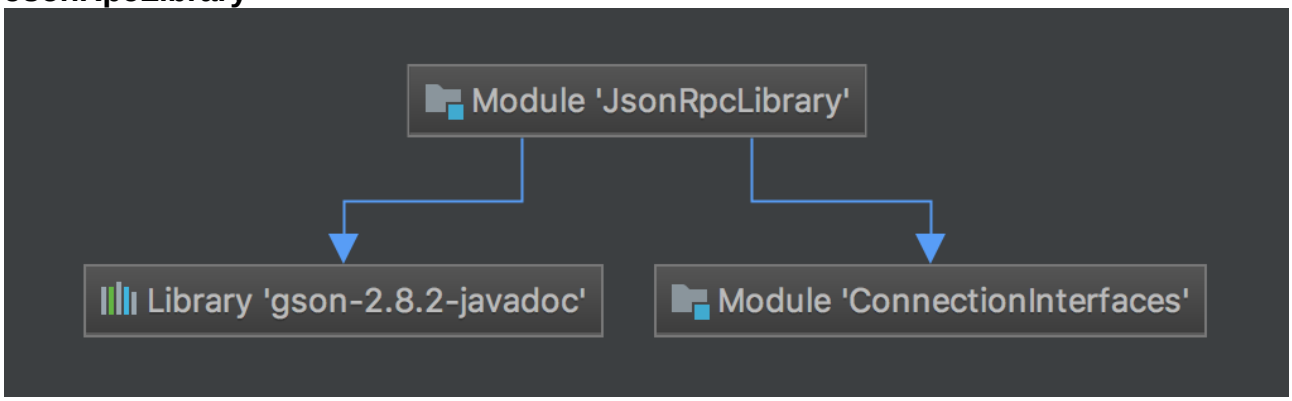
SearchStrategy



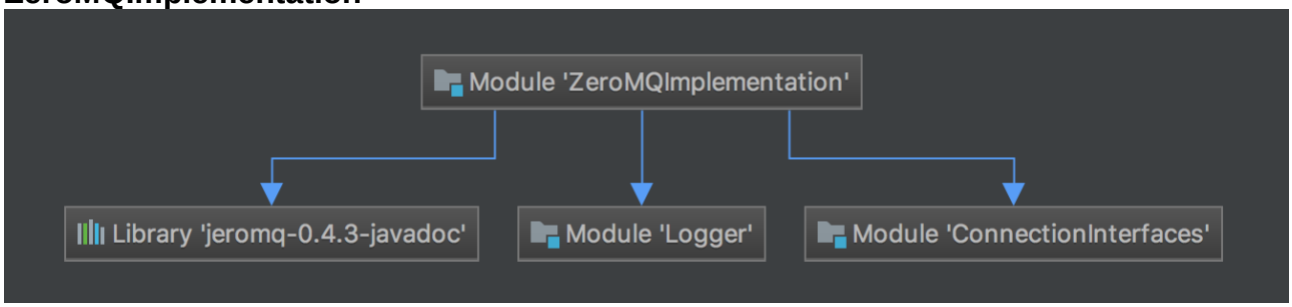
Service



JsonRpcLibrary



ZeroMQImplementation

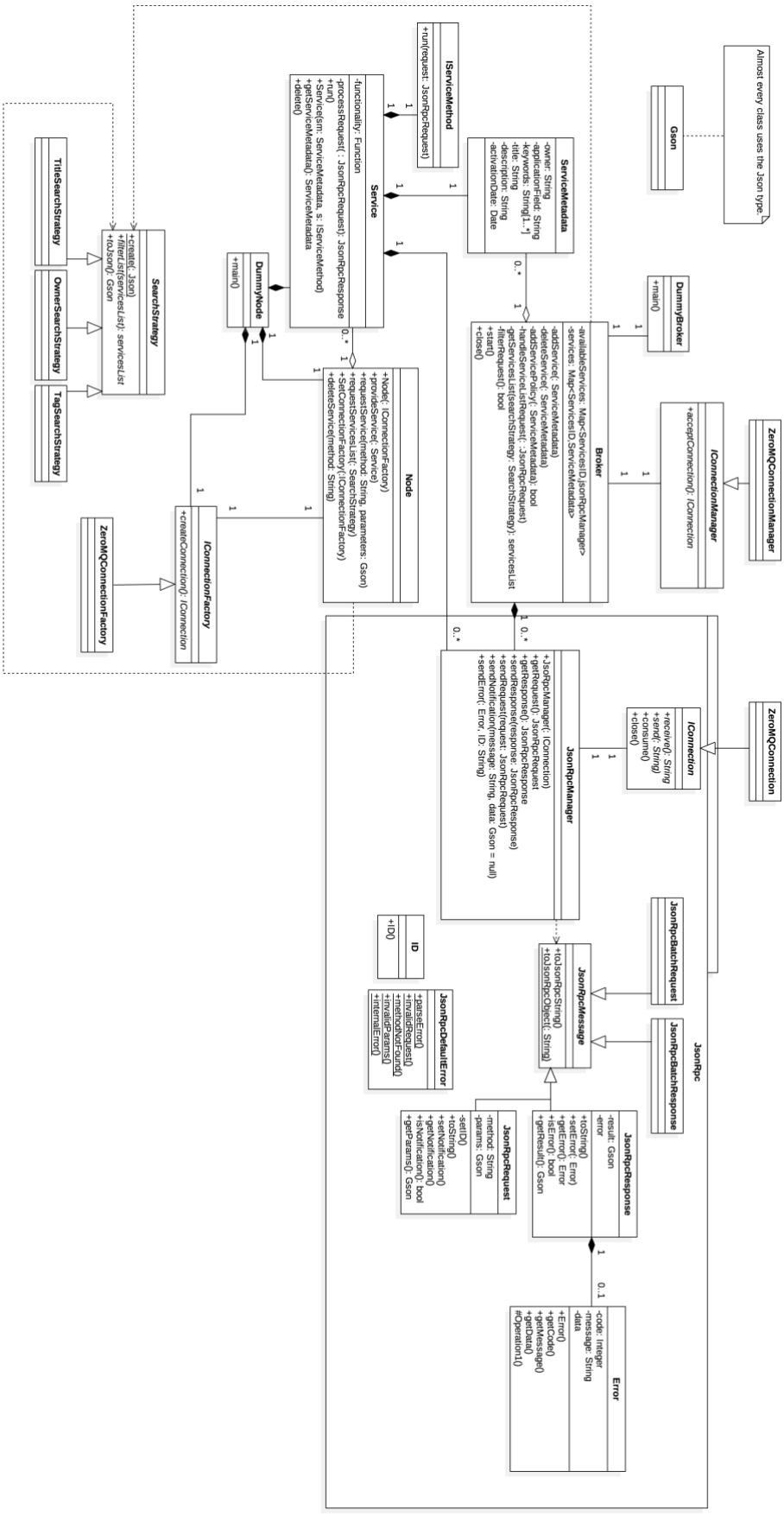


Tests

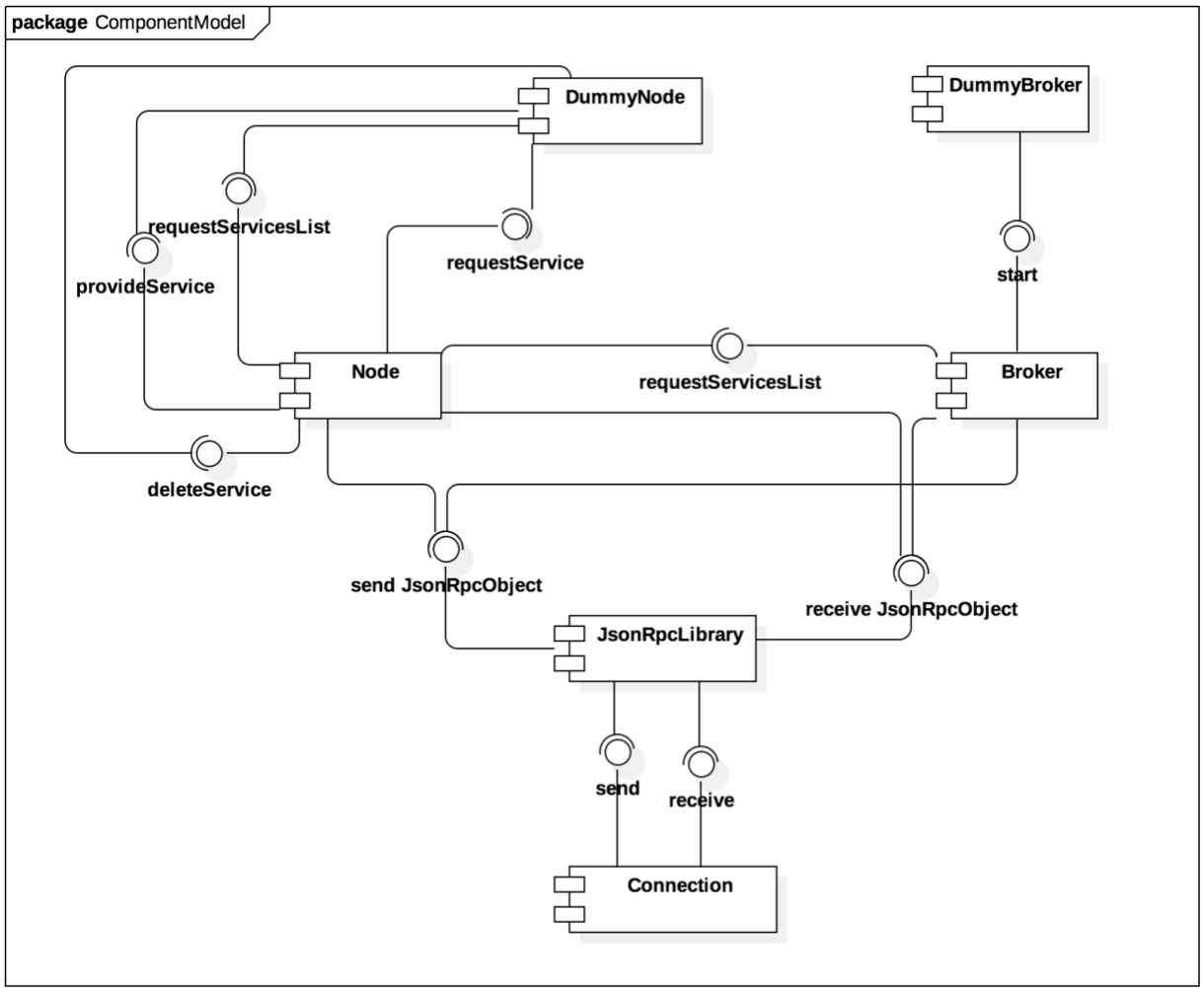
Sono stati effettuati quindici casi di test utilizzando JUnit4. I casi scelti ricoprono tutte le principali funzionalità di Node, Broker e JsonRpcManager.

Diagrammi UML aggiornati

Class diagram



Component diagram



Deployment diagram