Università degli studi di Bari "Aldo Moro"

Dipartimento di Informatica

Corso di Laurea in Informatica e Tecnologie per la Produzione del Software

---

Tesi di laurea in

Laboratorio di informatica

UN WRAPPER PYTHON PER IL FUZZY MARKUP LANGUAGE

Relatore:

Prof. Corrado Mencar

Laureando:

Andrea Pastore

ANNO ACCADEMICO 2016/2017

A mia madre..

Indice

1 Capitolo 1: Introduzione

In questo primo capitolo verrà esposto l'obiettivo della tesi ovvero l'incapsulamento di Fuzzy Markup Language in Python. Come prima cosa verranno dati dei cenni su:

- Che cos'è un sistema logica *fuzzy* (FLS).
- L'utilità di avere uno standard Fuzzy Markup Language (FML);
- L'utilità di avere una libreria Java Fuzzy Markup Laguange (JFML).

1.1 Sistema a logica fuzzy

Un sistema a logica *fuzzy* [5] permette ad un sistema di elaborare dati allontanandosi dalla logica binaria ed imitando il modo di prendere decisioni degli esseri umani che coinvolge tutte le possibilità intermedie. L'inventore della logica *fuzzy*, Lofti A. Zadeh, ha osservato che, a differenza dei computer, il processo decisionale di un essere umano include una vasta scelta di possibilità tra "si e no" come alcune di esse:

- Certamente si;
- Possibilmente si;
- Forse;
- Possibilmente no;
- Certamente no.

Quindi un sistema a logica *fuzzy* produce un output accettabile ma definito in risposta a input incompleti, ambigui, distorti o inaccurati, ossia i *fuzzy*.

La scelta di questa logica è dovuta a scopi commerciali e pratici, quali:

- Controllo di macchine e prodotti di consumo;
- Fornire un ragionamento accettabile ma non accurato;
- Aiutare la gestione dell'incertezza dell'ingegneria.

1.2 Standard FML

Lo standard Fuzzy Markup Language nasce dalla necessità di avere un sistema unificato e di alto livello per la descrizione di sistemi *fuzzy*, in modo da non avere tanti sistemi diversi da progettista a progettista.

L'utilizzo di uno standard, inoltre, fornisce un risparmio di tempo e di costi poiché un progettista non è costretto a descrivere la struttura e il comportamento di un sistema *fuzzy* partendo da zero.

Lo scopo di questo standard è di consentire la creazione di Fuzzy Logic System (FLS) interoperabili. Questo approccio consente ai progettisti di sistemi *fuzzy* di ottenere la trasparenza del design. Significa che, utilizzando FML, è possibile implementare lo stesso FLS su diverse architetture hardware con il minimo sforzo e senza ulteriori fasi di progettazione e implementazione. In breve, FML rende possibile modellare un FLS in un modo leggibile dall'uomo e indipendente dall'hardware (nel secondo capitolo verrà spiegato meglio cos'è FML).

1.3 JFML

Negli ultimi anni è stato sviluppato un software open source *JFML*, nel linguaggio di programmazione Java, che offre la costruzione di un sistema *fuzzy*. La sua novità e rilevanza derivano dal fatto che *JFML* è la prima libreria al mondo ad implementare lo Standard *IEEE* 1855, pubblicato e sponsorizzato dal Comitato degli standard della *IEEE Computational Intelligence Society*. JFML inoltre permette agli sviluppatori l'integrazione di dati XML e le funzioni di elaborazione in applicazioni Java sfruttando la nuova API Java JAXB (Java Architecture for XML Binding) che fornisce un modo rapido e conveniente per legare schemi XML e rappresentazioni Java.

Nel secondo capitolo verrà data una spiegazione più dettagliata di questi concetti.

1.4 Obiettivo della tesi

L'obiettivo della tesi è quello di creare un software in linguaggio Python che permetta la creazione di un sistema *fuzzy*. Da qui nasce il software chiamato PY4JFML, ideato per rispettare lo standard FML. Anziché effettuare una traduzione dal linguaggio Java a quello Python si è preferito utilizzare un programma che permette il collegamento tra Python e Java. Questo è possibile attraverso il programma Py4J [1] che permette l'incapsulamento di JFML in PY4JFML. Fatto ciò è possibile riutilizzare classi e metodi, presenti nella libreria JFML, direttamente in Python. Questo lavoro di tesi è svolto per consentire alla comunità di programmatori Python di poter utilizzare il Fuzzy Markup Language e di non limitare l'utilizzo di questo standard ad un unico linguaggio di programmazione.

2 Capitolo 2: Stato dell'arte

In questo secondo capitolo verranno dati dei cenni sulla teoria degli insiemi *fuzzy* e verrà esposta una panoramica generale circa lo standard FML. Infine verranno spiegate le tecnologie utilizzate per raggiungere l'obiettivo proposto in tesi:

- JFML: libreria scritta in Java che definisce la logica *Fuzzy*;
- PY4J: programma che collega Python a Java.

2.1 Teoria insieme Fuzzy

La logica f*uzzy* nasce nel 1965, introdotta dal Professor Lofti A. Zadeh, attraverso il concetto di insieme sfumato (o sfocato). Questa logica è una branca della matematica che permette ad un computer di elaborare dati allontanandosi dalla logica binaria e avvicinandosi a quella dell'infinito ventaglio di scelte, propria della logica umana.

Cominceremo la trattazione osservando che cosa è un insieme *fuzzy*, le sue proprietà e le sue operazioni.

Per comprendere cosa sia un insieme *fuzzy* richiamiamo il concetto di insieme secondo la teoria classica che è caratterizzata da una funzione di appartenenza con codominio [0,1], dove il valore 0 indica il concetto di "*non appartenenza*" ed il valore 1 il concetto di "*appartenenza*". Questi insiemi, in cui si escludono le sfumature proprie dei *fuzzy*, sono detti *crisp* o nitidi. In base a questa teoria ogni elemento di un universo appartiene ad un insieme.

Zadeh ha proposto di generalizzare il concetto di funzione rappresentando un insieme attraverso una funzione di appartenenza che mappi gli oggetti presi in considerazione nell'intervallo [0,1].

Si può quindi dedurre che gli insiemi *fuzzy* siano una generalizzazione della teoria classica. In essa un oggetto appartiene o meno ad un dato insieme, negli insiemi *fuzzy*, con un certo grado di appartenenza. Ciò consente agli insiemi *fuzzy* di realizzare modelli più flessibili, capaci di catturare le incompletezze e

le imprecisioni delle informazioni o dei dati disponibili per un'applicazione. Si genera così il formalismo necessario a gestire concetti imprecisi e sfumati.

Ad esempio, se definiamo secondo un insieme *fuzzy* una persona "alta" partendo da 170 cm allora quest'ultima sarà alta con grado 0, una di 180 cm con grado 0.5 ed una di 190 cm con grado 1.

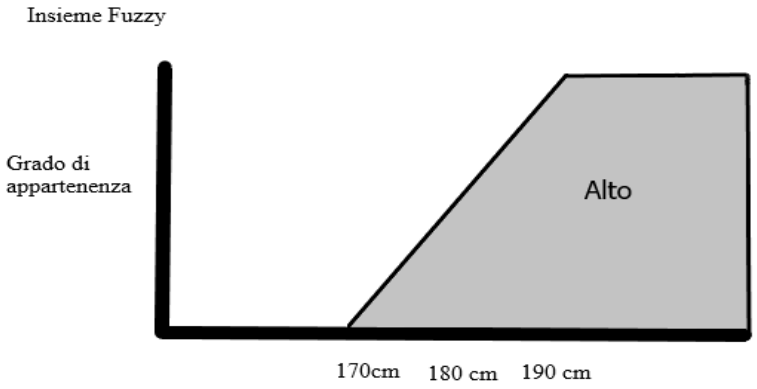


Figura (1) Esempio di grafico, definizione variabile "alto" su un insieme Fuzzy

In un sistema esperto tradizionale la regola o ha successo o non lo ha; nel sistema esperto *fuzzy* ,invece, la regola ha successo con differenti gradi (come per la "*scala dei grigi*"). Ovviamente in questo caso più di una regola può avere successo per un gruppo di input. Tuttavia una regola *fuzzy*, quando ha successo, lo ha con un certo grado dipendente dal livello di "*fiducia*" delle premesse antecedenti la regola. Queste premesse sono valutate usando funzioni "*qualità"* che producono livelli di fiducia combinati, usando gli operatori *fuzzy* per produrre il livello finale di attivazione dell'output.

2.1.1 Definizione e operazioni fuzzy

Secondo il formalismo matematico, un insieme *fuzzy* [2] A in una collezione di oggetti X è un insieme di coppie ordinate:

$$A = \{(x, \mu_A(x) | \, x \in X)\} \text{ [2]}$$

La variabile μA è chiamata funzione di appartenenza tale che: $\mu A : X \rightarrow M$, dove M è lo spazio di appartenenza nel quale viene mappato ogni oggetto di X. Se M contiene solo due punti, 0 ed 1, si ha un caso particolare in cui la teoria *fuzzy* riconduce alla teoria classica dove A è un insieme *crisp*.

Le operazioni della logica *fuzzy* sono così descritte:

- "a" AND "b" significa prendere il minimo tra i due elementi "a" e "b";
- "a" OR "b" significa prendere il massimo tra i due elementi "a" e "b";
- NOT "a" significa non prendere "a".

Negli insiemi *fuzzy* può accadere che non vengano soddisfatte alcune proprietà dell'algebra degli insiemi convenzionali come la "Legge del terzo escluso" ed il "Principio di non contraddizione".

Per quanto riguarda la "Legge del terzo escluso", il risultato dell'intersezione di un insieme con il proprio complementare è un insieme vuoto; questo non sempre si verifica per gli insiemi *fuzzy*.

Per "il Principio di non contraddizione" nella teoria classica, l'unione di un insieme con il proprio complementare coincide con l'universo del discorso. Nella teoria *fuzzy* non necessariamente si ottiene lo stesso risultato.

2.2 FML

Il Fuzzy Markup Language (*FML*) è uno Standard: un linguaggio di *markup* basato sul metalinguaggio dall'eXtensible Markup Language (XML). Esso sfrutta i vantaggi offerti da XML e gli strumenti correlati, per modellare un sistema a logica *fuzzy* volto a fornire una rappresentazione unificata e ben definita di *Fuzzy Logic Systems* (FLS). *FLS* è stato ampiamente utilizzato per la sua capacità di risolvere con successo una vasta gamma di problemi in diversi campi di applicazione. Tuttavia la sua replica e applicazione richiede un alto livello di conoscenza ed esperienza. Per questo motivo nasce la necessità di uno standard come FML così da poter facilitare il lavoro dei progettisti che vogliono utilizzare questa logica. FML include uno schema estendibile che definisce le componenti di base di un FLS e le abilità di modellazione diverse per categorie di motori di inferenza *fuzzy*, tra cui quello utilizzato in tesi: Mamdani.

La sua nascita risponde alla necessità di creare una struttura cooperativa basata sull'idea di albero etichettato, modello di dati derivati dal xml, che non dipenda dall'architettura dell'hardware usato.

Il linguaggio FML è un importate strumento per gli sviluppatori di algoritmi di estrazione di regole *fuzzy*. Infatti, come affermato da Mike Watts nel suo blog Computational Intelligence "*Sebbene la motivazione di Acampora per lo sviluppo di FML sembra essere quella di sviluppare controllori fuzzy incorporati per applicazioni di intelligenza ambientale, FML potrebbe essere una vera manna per gli sviluppatori di algoritmi di estrazione di regole fuzzy: dalla mia esperienza durante il mio dottorato, so che dover progettare un file formattare e implementare i parser appropriati per l'estrazione delle regole e i motori di inferenza fuzzy può essere un vero problema, impiegando tutto il tempo necessario per implementare l'algoritmo di estrazione delle regole stesso, preferirei usare qualcosa come FML per il mio lavoro*" [3], si può dire che l'utilizzo dello Standard, linguaggio scritto e approvato, comporterà un risparmio in termini di costi e tempo nel processo di sviluppo del *software*.

2.2.1 Struttura FML

FML è un programma che realizza una vista statica di un sistema *fuzzy* a cui viene fornito il cosiddetto *eXtensible Stylesheet Language Translator* (*XSLT*) per modificare la vista statica in una versione computabile. In particolare viene utilizzata la tecnologia *XSLT* per convertire una descrizione del controllore *fuzzy* in un linguaggio informatico per uso generico da calcolare su diverse piattaforme hardware.
Le componenti principali sono:

- Fuzzy knowledge base;
- Fuzzy rule base;
- Inerface system;
- Fuzzification inerface;
- Defuzzification inerface.

Queste componenti verranno spiegate meglio quando parleremo di JFML.
Attualmente è stato implementato un programma *FML* di conversione *XSLT* in codice Java eseguibile. In questo modo, grazie alle funzionalità di trasparenza fornite dalle macchine virtuali Java, è possibile ottenere un controller *fuzzy* modellato tramite *FML* ed eseguibile su una pletora di architetture hardware tramite tecnologie Java.

2.3 JFML

Come già accennato nel capitolo 1, JFML [4] è una libreria Java "*open source"* che mira a facilitare l'interoperabilità e l'usabilità dei sistemi *fuzzy*. La libreria *JFML* soddisfa le guide "*open source"* raccomandate per gli standard della comunità. Grazie all'adozione di questi standard, gli sviluppatori possono utilizzare e contribuire a *JFML* in modo più efficace.
JFML ha 4 importanti obiettivi:

- Consentire la creazione di un sistema *fuzzy*;
- Consentire il caricamento di un sistema *fuzzy*;
- Consentire la scrittura di un sistema *fuzzy*;

- Consentire la valutazione di un sistema *fuzzy* attraverso variabili di input e di ricevere i risultati attraverso l'output.

2.3.1 Struttura JFML

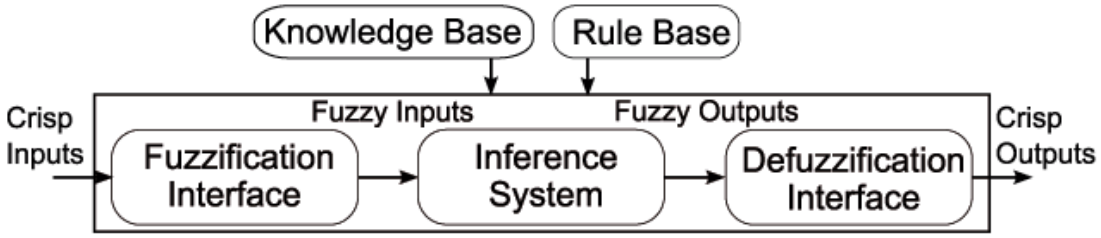


Figura (2) [4] Esempio di architettura Sistema Fuzzy

La creazione di un *Fuzzy Inference System* avviene attraverso:

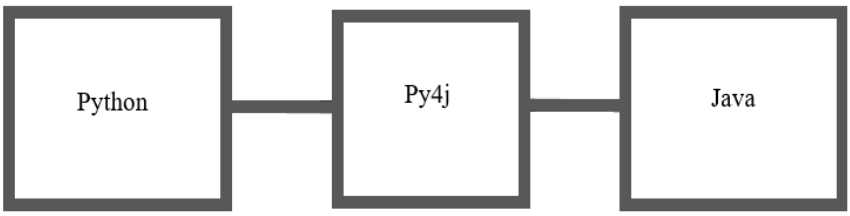
- Crisp inputs: le variabili inserite dall'utente che dovranno essere valutate dal sistema;
- Knowledge Base: la base di conoscenza *fuzzy* manipola le variabili utilizzate nel sistema corrispondenti agli input inseriti dall'utente;
- Rule Base: le regole base rappresentano l'insieme di relazioni tra le variabili *fuzzy* inserite nel sistema;
- Inference System: componente in grado di estrarre nuove conoscenze dal knowledge base e dalle rule base. Inoltre i valori inseriti sono valori reali mentre il sistema lavora con valori *fuzzy*. Tali valori sono gestiti da:
    - Fuzzification Interface: trasforma i valori reali inseriti dall'utente in valori *fuzzy;*
    - *Defuzzification Interface:* trasforma i valori *fuzzy* in valori reali da restituire all'utente.
- *Crisp outputs:* le variabili valutate dal sistema e restituite all'utente.

Si può creare uno qualsiasi dei quattro tipi di sistemi *fuzzy* racchiusi nello schema dello standard IEEE quali: *Mamdani, Tsukamoto, Takagi-Sugeno-*

Kang (*TSK*) e *AnYa*. In questa proposta di tesi ci soffermeremo sulle regole *Mamdani*.

2.4 PY4J

Py4J [1] è sviluppato da Barthélémy Dagenais con l'obiettivo di consentire agli sviluppatori, che programmano in Python, di beneficiare delle librerie Java. Quindi, grazie a questo "*ponte*", si possono utilizzare in Python le librerie scritte da utenti o le API già presenti in Java senza dover riscrivere tutto il codice. Ad esempio se si vogliono generare numeri casuali, anziché scrivere il codice, si può utilizzare direttamente la classe *Random()* già presente in Java permettendo così all'interprete Python di accedere in maniera dinamica agli oggetti Java presenti nella *virtual machine*.



Questo avviene tramite un modulo "py4j.java_gateway" che definisce la maggior parte delle classi necessarie per usare Py4J. JavaGateway è il principale punto di iterazione tra una Python Virtual Machine e una Java VM. Un'istanza JavaGateway è connessa a un'istanza Gateway presente in Java. Questo avviene grazie alla funzione *entry_point*, un metodo *JavaGateway* connesso all'istanza *Gateway.entryPoint* in Java.

La *JVM* di *JavaGateway* consente all'utente di accedere a classi, membri statici (campi e metodi) e costruttori di chiamate.

Inoltre il modulo *py4j.protocol* definisce la maggior parte dei tipi, delle funzioni e dei caratteri utilizzati nel protocollo Py4J. Non è necessario che venga utilizzato esplicitamente dai client di Py4J poiché viene caricato automaticamente dal modulo *java_gateway* e dal modulo *java_collections.*

3 Capitolo 3: PROPOSTA

L'obiettivo principale della tesi è stato quello di creare un programma Python capace di inglobare una parte delle classi presenti nella libreria JFML in modo da renderle direttamente fruibili in questo linguaggio attraverso l'utilizzo di py4j. In questo capitolo verranno presentati i due programmi scritti per il lavoro di tesi. La struttura di quest'ultimi è stata attuata seguendo le modalità dell'ingegneria del software: confini di sistema, casi d'uso, diagrammi di sequenza e UML.

3.1 Confini di sistema PY4JFML

I limiti del sistema PY4JFML si basano sulla mancanza di un totale *wrapping* di JFML su Python e sul mancato utilizzo delle regole *Tsukamoto*, *Takagi-Sugeno-Kang* (*TSK*) e *AnYa*.

Il sistema dunque si occuperà:

- Della creazione di un sistema di inferenza *fuzzy;*
- Del caricamento di un sistema *fuzzy* presente su un file xml;
- Della scrittura di un sistema *fuzzy* su un file xml.

seguendo le regole *Mamdani.*

Inoltre il client deve provvedere all'attivazione del server per poter permettere l'utilizzo della libreria JFML.

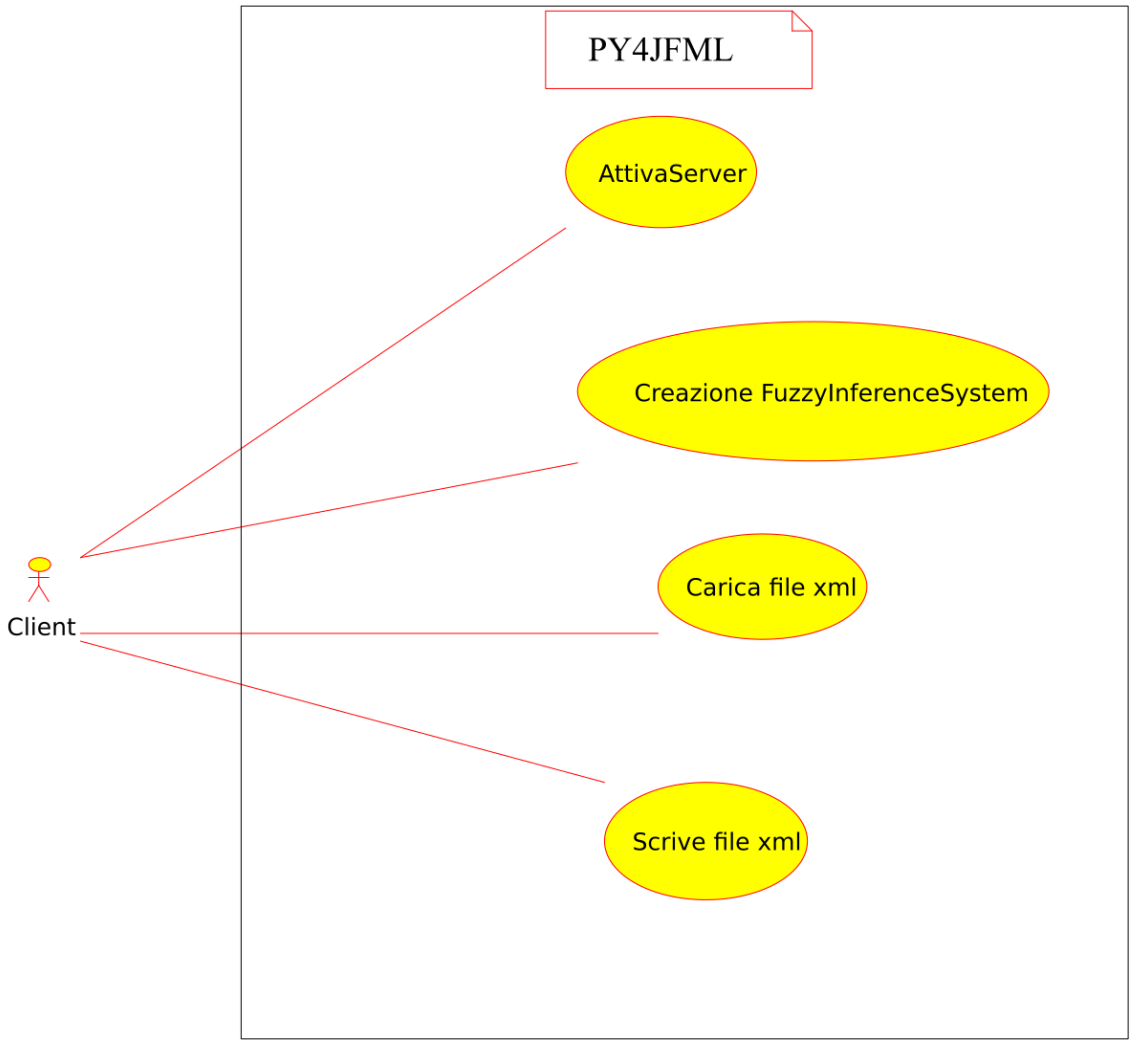


Figura (3) Confine di Sistema

Attori:

Attore	Client
ID	C1
Padre	
Semantica	Chiunque utilizza il sistema

3.2 Scelta delle classi

La scelta delle classi, da utilizzare nel software PY4JFML, si è basata sulla costruzione di un UML cartaceo della libreria JFML per avere un quadro generale delle varie classi presenti nella libreria, della suddivisione di quest'ultime nei vari package e di quali siano i loro collegamenti. Una volta fatto ciò si è seguito il manuale di esempio, presente nella documentazione di JFML, estrapolando le classi che servono per la creazione di un sistema *fuzzy*. Infine sono state aggiunte la classe di scrittura di un sistema *fuzzy* in un file xml e la classe di lettura di un file xml contenente un sistema *fuzzy* riscritte nella classe PY4JFML. Di seguito verranno riportate le classi sufficienti per creare un sistema *fuzzy* utilizzate in PY4JFML:

- FuzzyInferenceSystem
- KnowledgeBaseType
- FuzzyTerm
- FuzzyTermType
- FuzzyVariableType
- FuzzyRuleType
- MamdaniRuleBaseType
- AntecedentType
- ConsequentType
- ClauseType

3.3 Sistema: JFML_EntryPoint e PY4JFML

Il sistema trattato in tesi si divide in due parti: JFML_EntryPoint, scritto in Java (il server), e PY4JFML, scritto in Python (il Client).

JFML_EntryPoint è una classe scritta in Java che viene utilizzata come server e rimane in attesa di essere chiamata dal modulo PY4JFML. La classe importa due librerie: la libreria JFML e la libreria py4j. La libreria JFML contiene la logica e la definizione di un sistema *fuzzy*, mentre la libreria py4j contiene la

logica necessaria per il collegamento, per la gestione e l'utilizzo della JVM attraverso la *JavaGateway* che gode di 3 parametri:

- *Gateway_parameters:* un'istanza di *GatewayParameters* viene utilizzata per configurare le varie opzioni del gateway;
- *callback_server_parameters*: un'istanza di *CallbackServerParameters* viene utilizzata per configurare varie opzioni del server gateway e deve essere fornito per avviarlo. In caso contrario, i *callback* non saranno disponibili;
- *python_server_entry_point :* può essere richiesto dal lato Java se Java sta guidando la comunicazione.

La JFML_EntryPoint gestisce i metodi che chiamano i costruttori presenti nella libreria JFML che poi saranno richiamati da PY4JFML. Inoltre in questa classe è presente un "*main*" che le consentirà di essere avviata come un server.

Le classi che vengono gestite da PY4JFML andranno a formare il cuore dell'applicazione inglobando JFML.

Tali classi sono:

- PY4JFML: la classe principale che permette di scrivere e/o caricare un sistema *fuzzy*;
- FuzzyInferenceSystem: contiene la logica per la creazione di un sistema *fuzzy;*
- KnowledgeBaseType: include la definizione delle variabili di sistema, ossia la conoscenza del sistema *fuzzy*;
- *FuzzyVariableType*: rappresenta una variabile *fuzzy* di input o output. Ogni oggetto di questa classe contiene le seguenti informazioni:
    - *Name*: stringa univoca per variabile fuzzy, richiesta per la creazione;
    - *Scale*: stringa con la scala usata per misurare una variabile *fuzzy*. Informazione facoltativa quando creiamo la variabile;

- *domainLeft*: float che rappresenta il limite sinistro dell'universo del discorso di una variabile *fuzzy*, richiesta per la creazione;
- *domainRight*: float che rappresenta il limite destro dell'universo del discorso di una variabile *fuzzy*, richiesta per la creazione;
- *Type*: stringa con la posizione di una variabile *fuzzy* in una regola, antecedente o conseguente. I suoi valori possono essere "*input*" o "*output*". Alla sua creazione, questa informazione è facoltativa e il suo valore di default è "*input*".

- FuzzyTermType: consente la creazione dei termini delle variabili *fuzzy*. Le funzioni di appartenenza relative ai FuzzyTerm sono:
  - *Triangular:* questa opzione definisce la funzione triangolare in base ai parametri a, b e c;
  - *Left Linear:* questa opzione definisce la funzione lineare sinistra in base ai parametri a e b;
  - *Right Linear:* questa opzione definisce la funzione lineare destra in base ai parametri a e b;
  - *Trapezoidal:* questa opzione definisce la funzione trapezoidale in base ai quattro parametri a, b, c e d;
  - *Rectangular Shape:* questa opzione definisce la funzione rettangolare in base a due parametri a e b;
  - *Gaussian:* questa opzione definisce la funzione gaussiana simmetrica in base a due parametri c e σ;
  - *Left Gaussian:* questa opzione definisce la funzione gaussiana sinistra in base a due parametri c e σ;
  - *Right Gaussian:* questa opzione definisce la funzione gaussiana destra in base a due parametri c e σ;
  - *Pi-Shaped:* questa curva *spline-based* è così denominata a causa della sua forma Π e dipende da quattro parametri a, b, c e d;
  - *Z-Shape:* questa curva *spline-based* è così denominata a causa della sua forma Z e dipende da due parametri a e b;

    - *S-Shape:* questa curva *spline-based* è così denominata a causa della sua forma S e dipende da due parametri a e b;
    - *Singleton Shape:* questa opzione definisce la funzione *singleton* in base a un parametro a.
- FuzzyRuleType: consente la creazione delle regole del sistema *fuzzy;*
- MamdaniRuleBaseType: consente l'utilizzo delle regole Mamdani;
- AntecedentType: consente la creazione del set di regole antecedenti del sistema *fuzzy*;
- ConsequentType: consente la creazione del set di regole conseguenti del sistema *fuzzy*;
- ClauseType: consente la creazione delle clausole del sistema *fuzzy*.

3.4 Convenzioni nomi

I nomi di classi, metodi e variabili usati su PY4JFML sono stati gli stessi usati nella libreria JFML. Per identificare gli oggetti Java utilizzati da PY4JFML si è utilizzata una terminologia basata sull'utilizzo del termine "java" seguito da un underscore e un acronimo (ad esempio FuzzyInferenceSystem viene indicato con "*java_fis*"). Per i nomi dei metodi presenti su JFML_EntryPoint si è seguita la politica dell'abstract factory, cioè scrivere "*create*" seguito dal nome della classe usata per richiamare l'omonima in JFML.

3.5 Conversione tipo

Python, a differenza di Java, non è un linguaggio tipizzato. Ciò significa che quando si definisce una variabile non attribuiamo un tipo (ad esempio double, float e così via). Quando forniamo una variabile in Java, attraverso py4j, questa deve essere definita con una definizione forzata che su Python avviene tramite dei metodi che gestiscono questi tipi (ad esempio "float(*nomeVariabile*)"). Nonostante questa forzatura, in Java viene lanciata un'eccezione perché da Python riceve solo una variabile di tipo *int* o *double*.

Per questo si è dovuto inserire un "*cast*", in Java, che trasformasse questa variabile da double a float.

Uno stesso problema lo si è riscontrato nella gestione dei File perché la definizione di File, fornita da Python, non è la stessa di quella Java. È stato dunque necessario gestire questa situazione attraverso la JavaGateway per provvedere alla creazione di un file, richiamando la Java Virtual Machine che contiene la libreria *I/O* in modo da utilizzare la gestione dei File presente in Java, su Python.

3.6 Requisiti del sistema PY4JFML

Requisito_1: il sistema software è in grado di riprodurre un sistema di inferenza *fuzzy attraverso le regole Mamdani*;

Requisito_2: il sistema software è in grado di caricare un file xml contenete il sistema *fuzzy*;

Requisito_3: il sistema software è in grado di scrivere un file xml contenete il sistema *fuzzy*.

3.7 Casi D’uso

I casi d’uso definiscono cosa può fare il sistema. Essi sono definiti da:

- ID: indica la numerazione dei casi;
- Caso d’uso: indica il nome del caso d’uso;
- Breve descrizione: indica cosa fa il caso d’uso.

ID	Caso d’uso	Breve descrizione
1	Avvio Server	Il client avvia il server *JFML_EntryPoint* che resterà in ascolto.
2	Creazione *FuzzyInferenceSystem*	Il client crea la definizione della variabile del sistema
3	Caricamento file xml	Il client carica il sistema fuzzy
4	Scrittura file xml	Il client scrive il sistema fuzzy

3.7.1 Dettagli Casi D’uso

Nei dettagli dei casi d’uso sono presenti le seguenti descrizioni:

- Caso D’uso: il nome del caso d’uso;
- ID: numerazione del caso d’uso;
- Breve descrizione: cosa fa il caso d’uso;
- Attori primari: chi sono i principali attori che attuano il caso d’uso;
- Attori secondari: chi sono gli altri attori che prendono parte al caso d’uso;
- Precondizioni: cosa serve al caso d’uso per essere avviato;
- Sequenza di eventi: cosa succede attivando il caso d’uso;
- Post condizioni: cosa succede dopo l’attivazione del caso d’uso;
- Sequenza degli eventi alternativa: cosa può succedere se c’è un’anomalia attivando il caso d’uso.

Caso D’uso: Avvio Server
ID: 1
Breve descrizione: Il Client avvia il Server e dopo l’avvio può usare JFML direttamente da Python.
Attori Primari: Client.
Attori Secondari: Java.
Precondizioni: Il Client avvia il Server prima di usare PY4JFML.
Sequenza di eventi: 1. Il client Avvia il Server. 2. Il sistema permette di utilizzare le librerie java. 3. Il client può utilizzare le funzioni della libreria JFML.
Post condizioni: Il Client può usare la libreria JFML su Python.
Sequenza degli eventi alternativa: Nessuna.

Caso D’uso: Creazione *FuzzyInferenceSystem*
ID: 2
Breve descrizione: Il Client può creare variabili *fuzzy*.
Attori Primari: Client.
Attori Secondari: Python, Java.
Precondizioni: Il Client avvia il server prima di usare PY4JFML.
Sequenza di eventi: 1. Il client crea la variabile *fuzzy*.
Post condizioni: Variabile creata.
Sequenza degli eventi alternativa: Nessuna.

Caso D’uso: Caricamento file xml
ID: 3
Breve descrizione: Il Client può caricare un sistema *fuzzy* precedentemente creato.
Attori Primari: Client.
Attori Secondari: Python, Java.
Precondizioni: Il Client avvia il Server prima di usare PY4JFML.
Sequenza di eventi: 1. Il client carica il file xml. 2. Il sistema rende leggibile il file. 3. Il client può leggere il file.
Post condizioni: Il client legge il file xml.
Sequenza degli eventi alternativa: Nessuna.

Caso D’uso: Scrittura file xml
ID: 4
Breve descrizione: Il Client può registrare su un file xml il sistema *fuzzy* scritto.
Attori Primari: Client.
Attori Secondari: Python.
Precondizioni: Il Client avvia il Server prima di usare PY4JFML.
Sequenza di eventi: 1. Il client crea il sistema *fuzzy*. 2. Il Sistema scrive su un file xml il sistema appena creato.
Post condizioni: Il file xml è creato.
Sequenza degli eventi alternativa: Nessuna.

3.7.2 Diagrammi di analisi di casi d'uso

Avvio Server: il client, per eseguire PY4JFML, deve avviare il server direttamente da Eclipse, dall'apposita classe JFML_EntryPoint. Nel seguente schema viene spiegato come questo collegamento avviene:

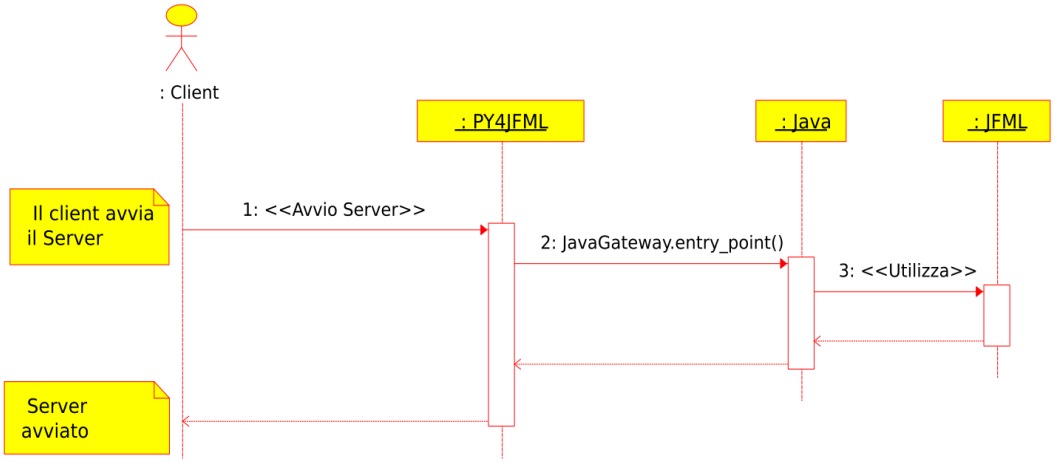


Figura (4) Diagramma di analisi di avvio server

PY4JFML, per funzionare, deve essere collegato alla libreria JFML e per fare ciò avrà bisogno di un collegamento con un server. Questo avviene attraverso un "*ponte*" che collega la classe PY4JFML, in Python, alla rispettiva classe JFML_EntryPoint presente in Java.

Creazione FuzzyInferenceSystem: PY4JFML permette al client la creazione di un sistema *fuzzy* attraverso il semplice inserimento di un nome, come mostrato nel diagramma di sequenza:

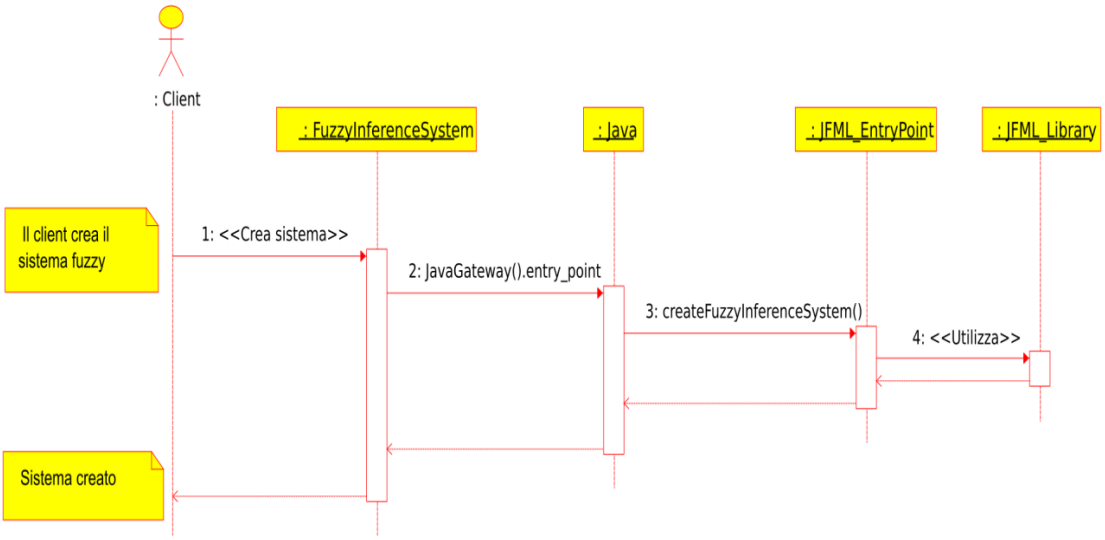


Figura (5) Diagramma di analisi della creazione di sistema fuzzy

FuzzyInferecenceSystem è una classe python che si collega alla JVM attraverso PY4J, sfruttando il metodo *entry_point* che fa parte della classe JavaGateway. Una volta effettuato il collegamento, viene richiamata la classe JFML_EntryPoint la quale contiene un metodo che chiama direttamente la classe presente nella libreria JFML. Così facendo si crea un sistema *fuzzy* su Python, sfruttando la libreria JFML presente in Java.

Caricamento file xml: PY4JFML permette al client anche di caricare un sistema *fuzzy* direttamente da un file xml precedentemente scritto:

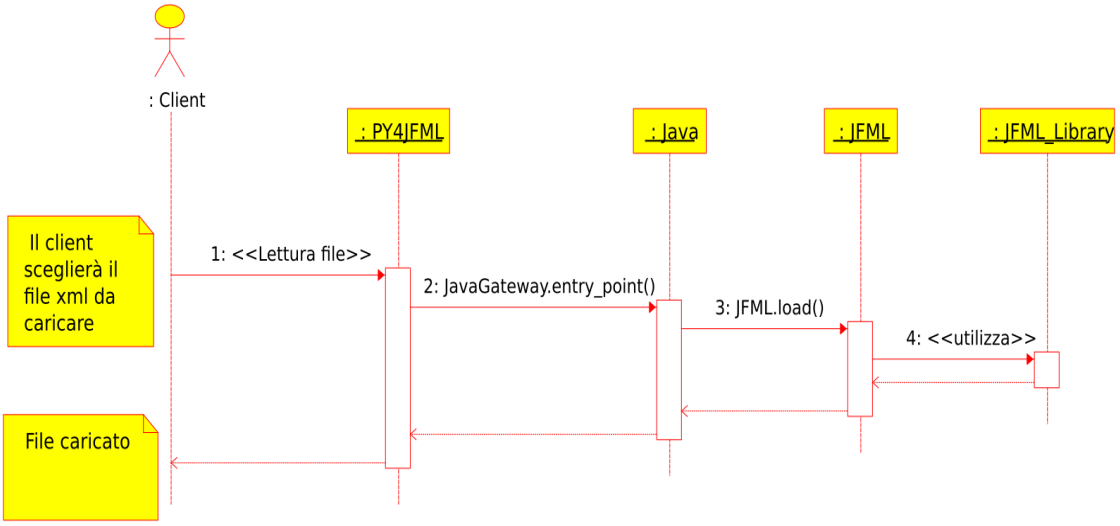


Figura (6) Diagramma di analisi della lettura di sistema fuzzy

PY4JFML è una classe python che si collega alla JVM attraverso PY4J, sfruttando il metodo *entry_point* che fa parte della classe JavaGateway. Una volta effettuato il collegamento viene richiamata la classe JFML_EntryPoint la quale contiene un metodo che chiama direttamente la classe presente nella libreria JFML. Così facendo, viene caricato un sistema *fuzzy*, su Python, sfruttando la libreria JFML presente in Java.

Scrittura file: PY4JFML infine, permette di scrivere un sistema *fuzzy* precedentemente creato su un file xml:

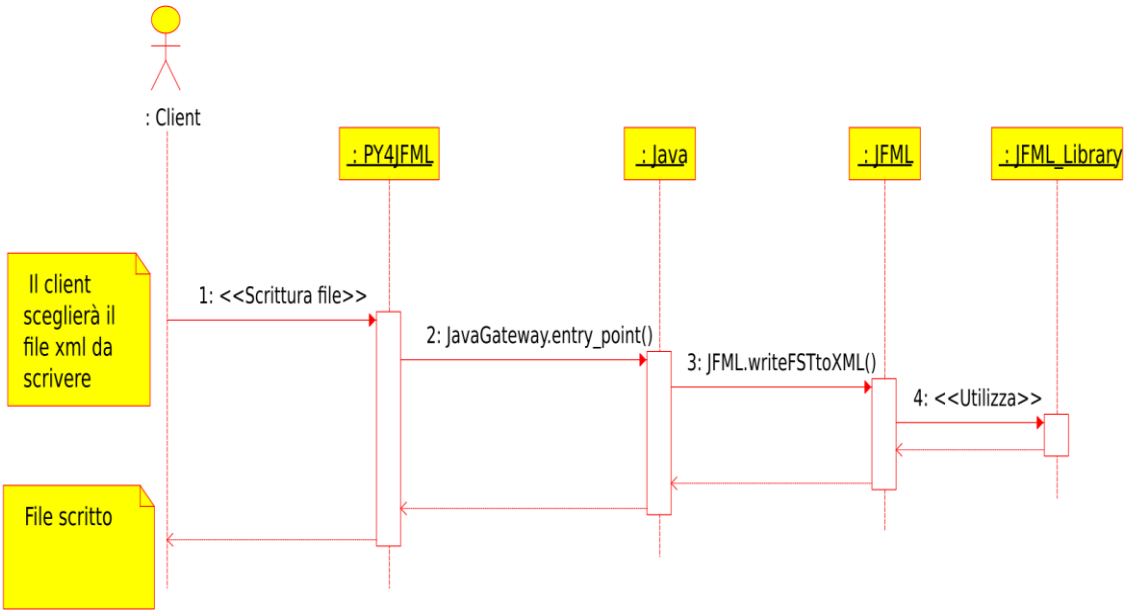


Figura (7) Diagramma di analisi della scrittura di sistema fuzzy

PY4JFML è una classe Python che si collega alla JVM attraverso PY4J, sfruttando il metodo *entry_point* che fa parte della classe JavaGateway. Una volta effettuato il collegamento viene richiamata la classe JFML_EntryPoint la quale contiene un metodo che chiama direttamente la classe presente nella libreria JFML. Così facendo, viene scritto un sistema *fuzzy* su un file sfruttando la libreria JFML presente in Java.

3.8 Architettura

L'architettura della soluzione verrà sviluppata seguendo, come linee guida, "il Principio di Separazione degli Interessi", "la Modularità" e "il Principio di Singola Responsabilità". Ciò permette di ottenere una buona predisposizione al cambiamento e una più semplice manutenzione.

Tali moduli saranno realizzati tramite un approccio bottom-up, sviluppando soluzioni per problemi atomici che costituiranno la base per le soluzioni dei problemi più complessi, combinate fino a comporre il sistema completo.

L'accesso ai dati verrà effettuato attraverso l'uso di un connettore PY4J che ha lo scopo di garantire il collegamento tra Python e Java attraverso la *JavaGateway*. Il pattern utilizzato è *l'abstract factory*, un design pattern che crea una famiglia di classi attraverso un interfaccia che ne definisce i metodi. Grazie al suo utilizzo si possono definire i tipi di classi distinte, una astratta ed una concreta, con lo scopo di separarne gli interessi nell'architettura. Nella classe astratta si definisce l'utilizzo di una certa classe, le cui funzioni verranno definite dalla classe concreta.

3.9 Diagramma di dispiegamento

In questo schema vengono riportate le componenti necessarie al sistema per essere eseguito e come avviene il collegamento tra Python e Java:

- Py4jfml: il software che si connette alla JavaGateway presente in py4j;
- Py4j JavaGateway: il metodo che permette la connessione;
- Py4j: lo strumento utilizzato per incapsulare JFML in PY4JFML;
- Java Fuzzy Markup Language: la libreria che contiene la logica *fuzzy*.

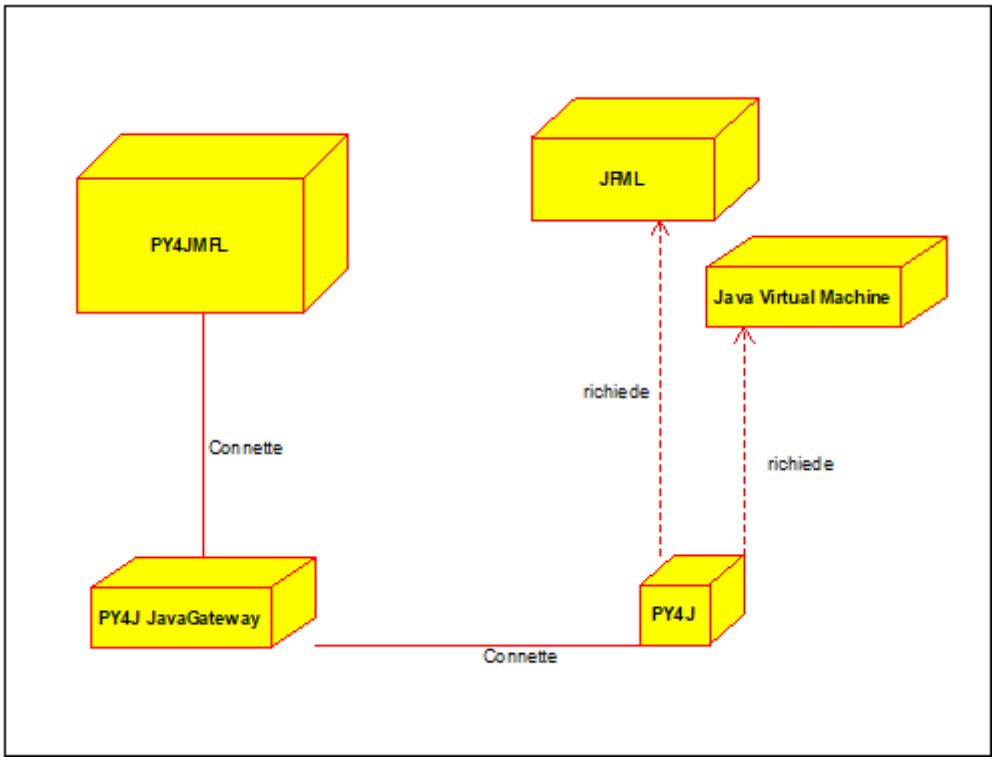


Figura (8) Diagramma di dispiegamento

3.10 UML PY4JFML

Nel seguente UML viene riportato un quadro generale della struttura del sistema proposto in tesi, senza l'inserimento delle classi e dei metodi ma con l'utilizzo delle componenti e di come avviene il collegamento tra il server e il client. Le componenti sono:

- JFML: la libreria che si vuole incapsulare in PY4JFML;
- JFML_EntryPoint: il server che permette l'esportazione della libreria JFML che sfrutta l'utilizzo dell'Abstract Factory;
- Abstract Factory: un design pattern che permette la creazione di una famiglia di classi;
- GatewayServer: il server che si crea avviando JFML_EntryPoint;
- Py4j: lo strumento utilizzato che permette lo scambio di dati tra Java e Python, utilizzando la JavaGateway;
- Py4jfml: il software che incapsula la libreria JFML e permette l'utilizzo della logica *fuzzy* in Python.

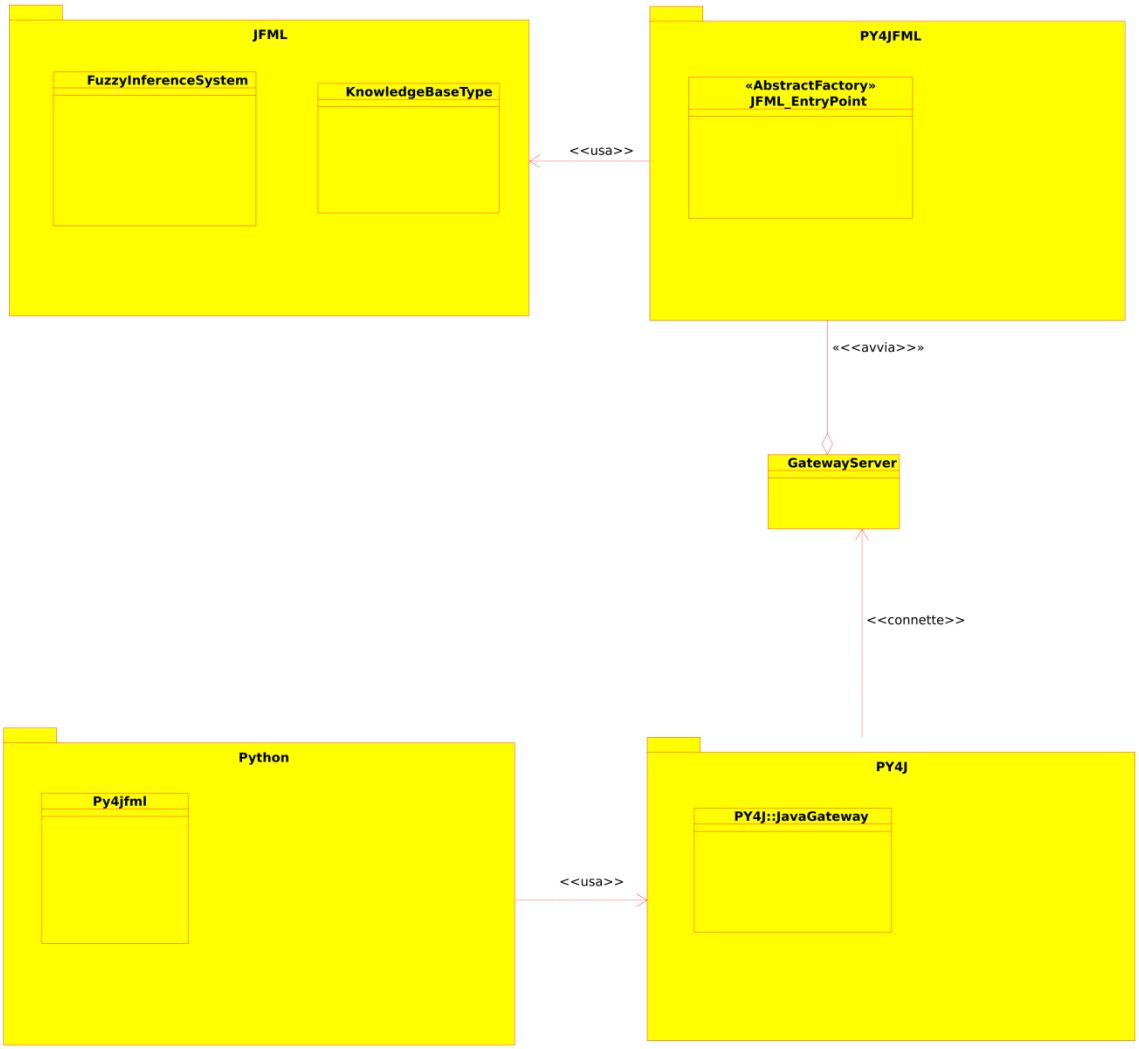


Figura (9) UML sistema fuzzy

Il programma presente in Python, Py4jfml, utilizza py4j attraverso la JavaGateway per connettersi al GatewayServer, che si avvia da Eclipse, eseguendo la classe JFML_EntryPoint. Quest'ultima utilizza la libreria JFML attraverso dei metodi che chiameranno le opportune classi. In questo modo Py4jfml incapsula la libreria JFML.

3.11 Diagramma di comportamento

Il diagramma di FuzzyInferecenceSystem è sufficiente a descrivere le operazioni che vengono effettuate dal sistema PY4JFML. Tutte le sue classi effettueranno le stesse operazioni per collegarsi alla libreria presente in Java, JFML. Una classe Python si collega alla JVM attraverso PY4J, sfruttando il metodo *entry_point* che fa parte della classe JavaGateway. Una volta effettuato il collegamento viene richiamata la classe JFML_EntryPoint la quale contiene un metodo che chiama direttamente la classe presente nella libreria JFML. Così facendo otteniamo tutte le operazioni presenti nel sistema PY4JFML sfruttando la libreria JFML di Java, ottenendo così i costruttori e i metodi nel programma scritto in Python.

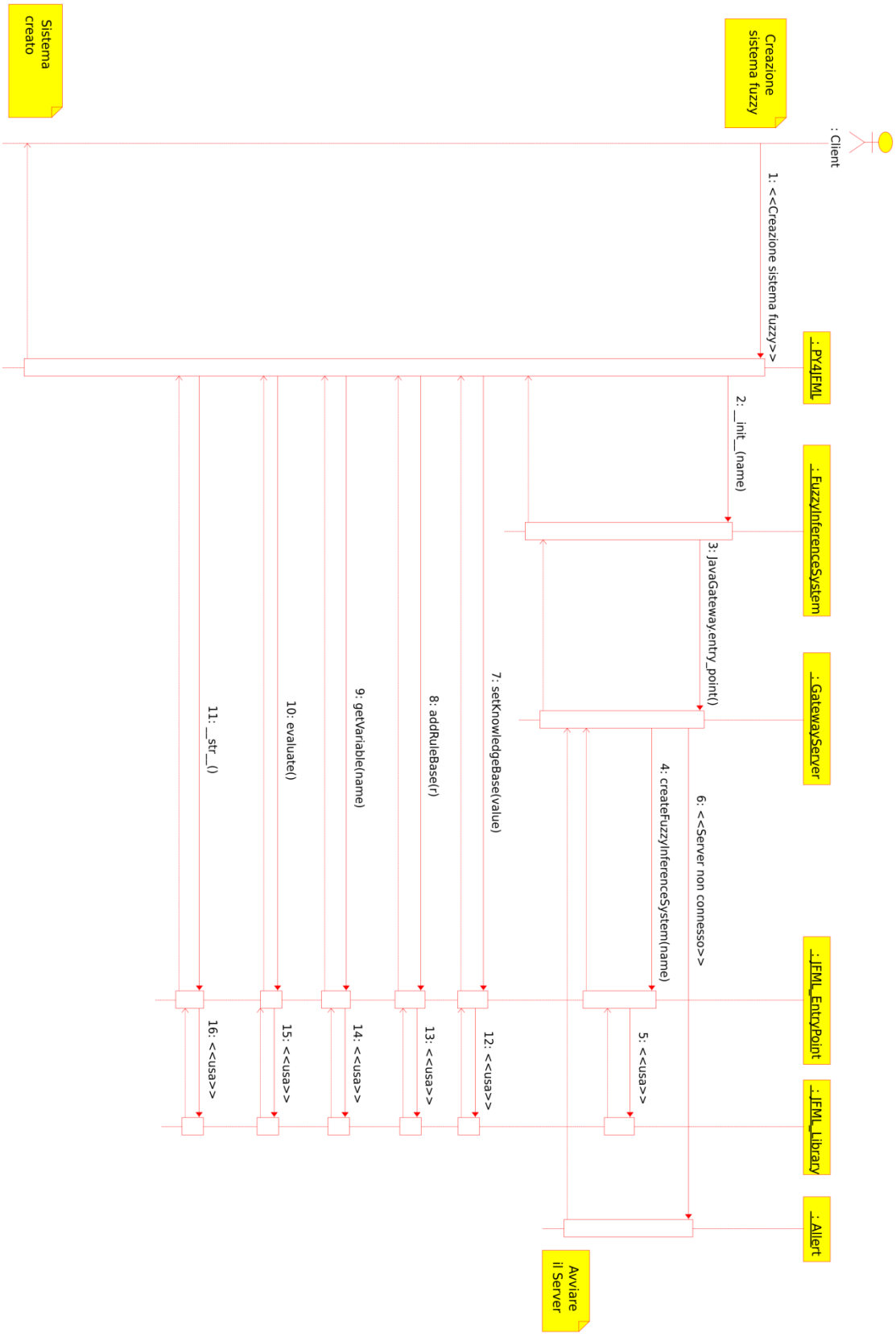


Figura (10) Diagramma di comportamento sistema fuzzy

4 Capitolo 4: SPERIMENTAZIONE

In questo capitolo verranno definiti i casi di test e verranno presentati i risultati della campagna di sperimentazione.

4.1 Piano di sperimentazione

La pianificazione dei casi di test è stata fatta seguendo i casi presenti sul manuale di JFML per poi comparare i risultati ottenuti sia da questa libreria sia da PY4JFML (in Java e in Python).

- La prima fase di test si è svolta sulla creazione di un sistema *fuzzy*. È stato dunque necessario definire il nome del sistema, creare le variabili (indicandone limite destro e sinistro), definire i termini di quest'ultime e aggiungerle alla conoscenza di base, creare le regole *Mamdani* e aggiungerle al sistema, definire le regole antecedenti e conseguenti ed infine trascrivere il sistema creato su un file xml. Una volta fatto ciò è possibile caricare il file per vederne la struttura e valutarlo con il sistema scritto in Java (come sperimenteremo nella seconda fase di test).
- La seconda fase di test si è svolta caricando un file xml contenente un sistema *fuzzy*. Una volta ottenute le variabili *fuzzy* è possibile impostarne i valori per poi proseguire nella valutazione del sistema ottenendo così i risultati, avendo seguito le regole imposte dal sistema. Infine verranno stampate le variabili di input, di output e il sistema *fuzzy* valutando la coerenza con il sistema scritto in Java.
- La terza fase dei test si è basata sulla scrittura di tre sistemi *fuzzy*: IrisMamdani1, IrisMamdani2 e IrisMamdani3 per poi riportarli su tre file xml. Si sono scelti i file IrisMamdani1 e IrisMamdani3 perché riportano diverse regole base, e l'IrisMamdani2 perché contiene diverse variabili *fuzzy* e diverse regole base. Dopo aver definito il nome del sistema, le variabili di input e output, la regola base Mamdani, le

regole antecedenti e conseguenti, si può procedere con la valutazione dei file xml, constatando che sia i file scritti usando PY4JFML sia quelli scritti usando JFML sono uguali.

- La quarta fase dei test, a differenza della seconda fase, si è sviluppata valutando i sistemi *fuzzy*, creati nella terza fase, attraverso dei casi di test che vengono effettuati attraverso il caso minimo, massimo e medio calcolati attraverso il limite sinistro e destro delle variabili di input. Successivamente si valuterà il sistema ottenendo delle variabili di output. Infine sarà possibile constatare se i risultati ottenuti da PY4JFML e da JFML sono uguali.
- La quinta e ultima fase dei test si è basata sulla valutazione dell'efficienza ed efficacia del programma PY4JFML. La valutazione dell'efficienza si è ottenuta attraverso un modulo presente in Python, "*cProfile*", che ci ha fornito dei dati sul tempo impiegato per il collegamento tra Python e Java e il tempo impiegato per l'esecuzione del sistema *fuzzy*. La valutazione dell'efficacia si è ottenuta comparando tutti i risultati ottenuti da PY4JFML con i risultati ottenuti in JFML.

4.2 Risultati

I risultati ottenuti dai casi di test sono stati soddisfacenti in quanto hanno rispettato una uguaglianza con il pacchetto presente in Java. I risultati sono stati confrontati con un opportuno programma notando che quest'ultimi sono uguali sia in PY4JFML che in JFML.

4.2.1 Risultato Test 1

La prima fase di test si è conclusa con un sistema *fuzzy* correttamente scritto su un file xml. È stato possibile eseguire lo stesso caso di test ottenendo gli stessi risultati sia su Java, sfruttando la libreria JFML, che su Python, sfruttando il software PY4JFML.

Di seguito verrà riportato il file xml ottenuto sia dal caso di test eseguito con PY4JFML che da JFML.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<fuzzySystem xmlns="http://www.ieee1855.org" name="tipper">
    <knowledgeBase>
        <fuzzyVariable name="tip" scale="" domainleft="0.0" domainright="10.0"
type="input">
            <fuzzyTerm name="cheap" complement="false">
                <triangularShape param1="1.0" param2="2.0" param3="3.0"/>
            </fuzzyTerm>
        </fuzzyVariable>
        <fuzzyVariable name="service" scale="" domainleft="0.0" domainright="10.0"
type="input">
            <fuzzyTerm name="poor" complement="false">
                <triangularShape param1="1.0" param2="2.0" param3="3.0"/>
            </fuzzyTerm>
        </fuzzyVariable>
        <fuzzyVariable name="food" scale="" domainleft="0.0" domainright="10.0"
type="input">
            <fuzzyTerm name="rancid" complement="false">
                <triangularShape param1="0.0" param2="2.0" param3="4.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="rancid" complement="false">
                <leftLinearShape param1="0.0" param2="4.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="delicious" complement="false">
                <rightLinearShape param1="4.0" param2="10.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="rancid" complement="false">
                <trapezoidShape param1="0.0" param2="1.0" param3="3.0" param4="4.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="good" complement="false">
                <rectangularShape param1="4.0" param2="7.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="good" complement="false">
                <gaussianShape param1="5.0" param2="1.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="good" complement="false">
                <gaussianShape param1="5.0" param2="1.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="rancid" complement="false">
                <leftGaussianShape param1="2.0" param2="1.0"/>
```

```
            </fuzzyTerm>
            <fuzzyTerm name="delicious" complement="false">
                <rightGaussianShape param1="8.0" param2="1.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="good" complement="false"/>
            <fuzzyTerm name="rancid" complement="false">
                <zShape param1="2.0" param2="4.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="delicious" complement="false">
                <rightGaussianShape param1="6.0" param2="8.0"/>
            </fuzzyTerm>
            <fuzzyTerm name="good" complement="false">
                <singletonShape param1="5.0"/>
            </fuzzyTerm>
        </fuzzyVariable>
    </knowledgeBase>
    <mamdaniRuleBase name="MamdaniRB1" activationMethod="MIN" andMethod="MIN"
orMethod="MAX">
        <rule name="rule1" orMethod="MAX" connector="or" weight="1.0">
            <antecedent>
                <clause>
                    <variable>food</variable>
                    <term>rancid</term>
                </clause>
                <clause modifier="very">
                    <variable>service</variable>
                    <term>poor</term>
                </clause>
            </antecedent>
            <consequent>
                <then>
                    <clause>
                        <variable>tip</variable>
                        <term>cheap</term>
                    </clause>
                </then>
            </consequent>
        </rule>
    </mamdaniRuleBase>
</fuzzySystem>
```

4.2.2 Risultato Test 2

Nella seconda fase di test si è eseguito uno script di codice adattato ai due linguaggi di programmazione. La sperimentazione si è conclusa con una valutazione di un sistema *fuzzy*, di uno stesso file xml, preso dagli esempi della libreria di JFML che ha portato ad un identico output rispettando le stesse regole *fuzzy*.

PY4JFML:

```
RESULT:
 (INPUT): food = 6.0, service=8.0
 (OUTPUT): tip = 12.345622

FUZZY SYSTEM: tipper - MAMDANI
KNOWLEDGEBASE:
  *food - domain[0.0, 10.0] - input
        rancid -  triangular [a: 0.0, b: 2.0, c: 5.5]
        delicious -  rightLinear [a: 5.5, b: 10.0]

  *service - domain[0.0, 10.0] - input
        poor -  leftGaussian [c: 0.0, sigma: 2.0]
        good -  gaussian [c: 5.0, sigma: 4.0]
        excellent -  rightGaussian [c: 10.0, sigma: 2.0]

  *tip - domain[0.0, 20.0] - Accumulation:MAX; Defuzzifier:COG - output
        cheap -  triangular [a: 0.0, b: 5.0, c: 10.0]
        average -  triangular [a: 5.0, b: 10.0, c: 15.0]
        generous -  triangular [a: 10.0, b: 15.0, c: 20.0]

RULEBASE:
  *mamdani - rulebase1: OR=MAX; AND=MIN; ACTIVATION=MIN
        RULE 1: reg1 - (1.1253517E-7) IF food IS rancid OR service IS very poor THEN tip IS
cheap [weight=1.0]
```

RULE 2: reg2 - (0.7548396) IF service IS good THEN tip IS average [weight=1.0]

RULE 3: reg3 - (0.60653067) IF service IS excellent OR food IS delicious THEN tip IS generous [weight=1.0]

JFML:

```
RESULTS
 (INPUT): food=6.0, service=8.0
 (OUTPUT): tip=12.345622

FUZZY SYSTEM: tipper - MAMDANI
KNOWLEDGEBASE:
  *food - domain[0.0, 10.0] - input
      rancid -  triangular [a: 0.0, b: 2.0, c: 5.5]
      delicious -  rightLinear [a: 5.5, b: 10.0]

  *service - domain[0.0, 10.0] - input
      poor -  leftGaussian [c: 0.0, sigma: 2.0]
      good -  gaussian [c: 5.0, sigma: 4.0]
      excellent -  rightGaussian [c: 10.0, sigma: 2.0]

  *tip  -  domain[0.0,  20.0]  -  Accumulation:MAX;  Defuzzifier:COG  -
output
      cheap -  triangular [a: 0.0, b: 5.0, c: 10.0]
      average -  triangular [a: 5.0, b: 10.0, c: 15.0]
      generous -  triangular [a: 10.0, b: 15.0, c: 20.0]

RULEBASE:
  *mamdani - rulebase1: OR=MAX; AND=MIN; ACTIVATION=MIN
      RULE 1: reg1 - (1.1253517E-7) IF food IS rancid OR service IS
very poor THEN tip IS cheap [weight=1.0]
      RULE 2: reg2 - (0.7548396) IF service IS good THEN tip IS
average [weight=1.0]
      RULE 3: reg3 - (0.60653067) IF service IS excellent OR food IS
delicious THEN tip IS generous [weight=1.0]
```

Il risultato del test è stato positivo, in quanto i risultati sono stati confrontati con un opportuno programma notando che quest'ultimi sono uguali sia in PY4JFML che in JFML.

4.2.3 Risultato Test 3

La terza fase di test si è conclusa con la scrittura di tre file xml IrisMamdani1, IrisMamdani2 e IrisMamdani3 che sono stati scritti attraverso lo stesso script di codice sia su Java che su Python, rispettando la loro architettura. Infine i risultati sono stati confrontati con un opportuno programma notando che quest'ultimi sono uguali sia in PY4JFML che in JFML.

4.2.4 Risultato Test 4

La quarta fase di test si è conclusa con una valutazione dei tre file IrisMamdani1, IrisMamdani2 e IrisMamdani3, attraverso lo stesso script di codice sia su Java che su Python, ottenendo gli stessi risultati che si sono confrontati come nel caso di test 1. Durante l'esecuzione dei casi di test si è notato un lancio di un'eccezione in JFML quando si violavano il limite destro o sinistro. Tale eccezione si propaga anche in Python. Ne consegue: java.lang.RuntimeException: The value 0.0 in variable PetalWidth is out of range domain [0.1, 2.5]).

In quest'ultima si nota che il valore è fuori dal range. I casi di test eseguiti nello script IrisMamdani1 e IrisMamdani3 sono stati eseguiti valutando la variabile del sistema fuzzy attraverso il caso minimo imposto dal limite sinistro, il caso medio (facendo una media tra i due limiti) e il caso massimo imposto dal limite destro.

Nome variabile	Minimo	Massimo	Medio
PetalWidth	0.1	2.5	1.2

Per quanto riguarda lo script di test dell'IrisMamdani2, visto che le variabili erano quattro, è stato attuato attraverso 81 casi di test combinando i casi minimo, medio e massimo delle quattro variabili ottenendo dei risultati sia su Java che su Python.

Nome variabile	Minimo	Massimo	Medio
SepalLenght	4.3	7.9	6.1
SepalWidht	2	4.4	3.2
PetalLenght	1	6.9	3.9
PetalWidth	0.1	2.5	1.2

Quest'ultimi sono stati confrontati con un opportuno programma notando che i risultati ottenuti sono uguali sia in PY4JFML e sia in JFML.

4.2.5 Risultato Test 5

La quinta fase di test si è conclusa con la valutazione dell'efficacia ed efficienza del sistema *fuzzy*. L'efficacia è stata valutata durante i quattro casi di test visti in precedenza, notando che i risultati ottenuti sia utilizzando PY4JFML, su Python, e sia JFML, su Java sono gli stessi. Mentre dai risultati dell'efficienza si nota che il tempo impiegato da py4j a collegare PY4JFML e JFML è basso e che non influenza l'esecuzione di PY4JFML.

Nella seguente tabella vengono mostrati alcuni dati di questa valutazione che sono stati scelti per la loro utilità: creazione connessione, invio comando, costruzione dell'oggetto "*java_gateway*" e esecuzione di Py4jfml:

Ncalls	Tottime	Percall	cumTime	Percall	Filname:fuction
2	0.000	0.000	0.009	0.004	java_gateway.py: 848(_get_connection)
2	0.000	0.000	0.010	0.005	java_gateway.py: 885(send_command)
1	0.000	0.000	0.009	0.009	java_gateway.py: 857(_create_connection)
1	0.000	0.000	0.008	0.008	java_gateway.py: 959(__init__)
1	0.000	0.000	0.013	0.013	Py4jfml.py:88(__init__)

Spiegazione tabella:

- Ncalls: il numero di chiamate effettuate;
- Tottime: il tempo totale trascorso nella funzione assegnata (escluse sotto funzioni);
- Percall: il quoziente tra Tottime/Ncalls;
- cumTime: il tempo cumulativo trascorso in questa e in tutte le sotto funzioni;
- Percall: il quoziente tra cumTime/Ncalls;
- Filename:funciont: contiene il nome del file e fornisce i rispettivi dati di ciascuna funzione.

5 Capitolo 5: CONCLUSIONI

Nei capitoli precedenti si è dimostrato che gli obiettivi della tesi sono stati soddisfatti in maniera parziale in quanto non è stata incapsulata l'intera libreria JFML ma solo la parte necessaria alla creazione e valutazione di un sistema *fuzzy* e alla sua scrittura e lettura seguendo le regole Mamdani. L'esportazione della libreria JFML è attuabile senza riscrivere tutto il codice in Python bensì utilizzando uno strumento py4j. Infatti si è dimostrato che l'utilizzo di py4j non aumenta in maniera rilevante i tempi computazionali.

A fronte di quanto detto, si conclude che si evincono benefici apprezzabili dell'incapsulamento del libreria JFML in Python attraverso l'utilizzo di py4j.

5.1 Sviluppi futuri

Durante il lavoro di tesi ci si è basati, principalmente, sulla formazione del cuore di questa struttura PY4JFML per vedere se questo metodo fosse attuabile e se i tempi di collegamento fossero troppo incisivi sul progetto.

Gli sviluppi futuri potrebbero basarsi:

- Sull'implementazione delle classi mancanti della libreria JFML così da avere direttamente tutta la libreria in Python;
- Sulla possibile eliminazione dell'avvio da parte del client da Eclipse, facendo un' esportazione del jar avviandolo direttamente in Python.

A Documentazione Pydoc

NAME: progettoPy4jfml.Py4jfml

CLASSES: builtins.object

- AntecedentType
- ClauseType
- ConsequentType
- FuzzyInferenceSystem
- FuzzyRuleType
- FuzzyTerm
- FuzzyTermType
- FuzzyVariableType
- KnowledgeBaseType
- MamdaniRuleBaseType
- PY4JFML

A.1 Py4jfml

class PY4JFML(builtins.object)

| La classe PY4JFML permette di caricare o scirvere un file xml contenente un sistema fuzzy

|

| Static methods defined here:

|

| load(xml_fileName)

| il metodo load permette di caricare un file xml

| :param xml_fileName: contiene il percorso del file

| :param JFML: crea una istanza vuota della classe scritta in java

| :param xml: contiene il file xml

| :param java_fis: è la variabile che si collega alla libreria JFML

| :param fuzzyInferenceSystem: è la variabile che contiene il file appena caricato

| :return: fuzzyInferenceSystem

|

| writeFSTtoXML(fst, str_output)

| il metodo writeFSTtoXML permette di scrivere un file xml

| :param fst: contiene il sistema fuzzy

| :param str_output: contiene il percorso del file

| :param JFML: crea una istanza vuota della classe scritta in java

| :param xmlOutput: contiene il file xml

| :return: JFML.writeFSTtoXML(fst.java_fis, xmlOutput)

A.2 FuzzyVariableType

class FuzzyVariableType(builtins.object)

| Definisce i tipi di variabile fuzzy

|

| Methods defined here:

|

| __init__(self, name=None, domainLeft=None, domainRight=None)

| Costruttori dei tipi di variabile fuzzy

| :param name: nome della regola

| :param domainLeft: dominio sinistro

```
|     :param domainRight: dominio destro
|     :param java_fvt: contiene il collegamento con JFML
|
| addFuzzyTerm(self, ft)
|     Aggiunge la regola al sistema fuzzy
|     :param ft: regola fuzzy
|
| getName(self)
|     Ottiene il nome delle regole fuzzy
|     :return: restituisce il nome delle regole fuzzy
|
| getValue(self)
|     Ottiene il valore delle regole fuzzy
|     :return: restituisce il valore
|
| setAccumulation(self, value)
|     Imposta il valore di accumulazione
|     :param value: valore di accumulazione
|     :return: restituisce il valore di accumulazione
|
| setDefaultValue(self, value)
|     Imposta i valori di default
|     :param value: valore di default
|     :return:
```

```
|
| setDefuzzifierName(self, value)
|     Imposta il nome del defuzzifier
|     :param value: nome deffuzifier
|     :return: restituisce il nome
|
| setType(self, value)
|     Imposta il tipo delle regole fuzzy
|     :param value: tipo
|
| setValue(self, x)
|     Imposta il valore alle regole fuzzy
|     :param x: contiene il valore
```

A.3 KnowledgeBaseType

```
class KnowledgeBaseType(builtins.object)
| Conoscenza di base del sistema fuzzy
|
| Methods defined here:
|
| __init__(self)
|     Costruttore della conoscenza di base
|
```

| addVariable(self, var)

| Aggiunge la variabile alla conoscenza base

| :param var: contiene la variabile da aggiungere

A.4 FuzzyTerm

class FuzzyTerm(builtins.object)

| La classe FuzzyTerm contiene i tipi a cui fanno riferimento i grafici

| :param TYPE_rightLinearShape: contiene il riferimento alla linea destra

| :param TYPE_leftLinearShape: contiene il riferimento alla linea sinistra

| :param TYPE_piShape: contiene il riferimento al piShape

| :param TYPE_triangularShape: contiene il riferimento al triangolo

| :param TYPE_gaussianShape: contiene il riferimento alla gaussiana

| :param TYPE_rightGaussianShape: contiene il riferimento alla gaussiana destra

| :param TYPE_leftGaussianShape: contiene il riferimento alla gaussiana sinistra

| :param TYPE_trapezoidShape: contiene il riferimento al trapezio

| :param TYPE_singletonShape: contiene il riferimento alla linea

| :param TYPE_rectangularShape: contiene il riferimento al rettangolo

| :param TYPE_zShape: contiene il riferimento alla Z shape

| :param TYPE_sShape: contiene il riferimento alla S shape

| :param TYPE_pointSetShape: contiene il riferimento al punto

| :param TYPE_pointSetMonotonicShape: contiene il riferimento al pointSetMonotonicShape

| :param TYPE_circularDefinition: contiene il riferimento al circularDefinition

| :param TYPE_customShape: contiene il riferimento al customShape

| :param TYPE_customMonotonicShape: contiene il riferimento al customMonotonicShape

A.5 FuzzyTermType

class FuzzyTermType(builtins.object)

| Definisce i termini del sistema fuzzy

|

| Methods defined here:

|

| __init__(self, name, type_java, param)

| Costruttore FuzzyTermType

| :param name: contiene il nome dei termini

| :param type_java: contiene i tipi descritti nella classe FuzzyTerm

| :param param: è un array di numeri float che indicano i parametri dei grafici

| :param java_ftt: contiene il collegamento con JFML

|

| setComplement(self, value)

| Imposta i valori dei complementi

| :param value: contiene il complemento

A.6 FuzzyVariableType

```
class FuzzyVariableType(builtins.object)
 |  Definisce i tipi di variabile fuzzy
 |
 |  Methods defined here:
 |
 |  __init__(self, name=None, domainLeft=None, domainRight=None)
 |      Costruttori dei tipi di variabile fuzzy
 |      :param name: nome della regola
 |      :param domainLeft: dominio sinistro
 |      :param domainRight: dominio destro
 |      :param java_fvt: contiene il collegamento con JFML
 |
 |  addFuzzyTerm(self, ft)
 |      Aggiunge la regola al sistema fuzzy
 |      :param ft: regola fuzzy
 |
 |  getName(self)
 |      Ottiene il nome delle regole fuzzy
 |      :return: restituisce il nome delle regole fuzzy
 |
 |  getValue(self)
 |      Ottiene il valore delle regole fuzzy
 |      :return: restituisce il valore
 |
```

```
| setAccumulation(self, value)
|     Imposta il valore di accumulazione
|     :param value: valore di accumulazione
|     :return: restituisce il valore di accumulazione
|
| setDefaultValue(self, value)
|     Imposta i valori di default
|     :param value: valore di default     |
| setDefuzzifierName(self, value)
|     Imposta il nome del defuzzifier
|     :param value: nome deffuzifier
|     :return: restituisce il nome
|
| setType(self, value)
|     Imposta il tipo delle regole fuzzy
|     :param value: tipo
|
| setValue(self, x)
|     Imposta il valore alle regole fuzzy
|     :param x: contiene il valore
```

A.7 MamdaniRuleBaseType

class MamdaniRuleBaseType(builtins.object)

| Definisce le regole di base Mamdani del sistema fuzzy

|

| Methods defined here:

|

| __init__(self, name)

| Costruttore Mamdani

| :param name: contiene il nome delle regole base Mamdani

| :param java_mrbt: contiene il collegamento con JFML

|

| addRule(self, rule)

| Aggiunge una lista di regole Mamdani

| :param rule: contiene una lista di regole Mamdani

A.8 FuzzyRuleType

class FuzzyRuleType(builtins.object)

| Definisce il tipo di regole fuzzy

|

| Methods defined here:

|

| __init__(self, name=None, connector=None, connectorMethod=None, weight=0.0)

| Costruttore FuzzyRuleType

| :param name: contiene il nome delle regole fuzzy

| :param connector: contiene il connettore che collega le diverse clausole nella parte antecedente (e/o)

| :param connectorMethod: contiene l'algoritmo da utilizzare se il connettore scelto è e od o.

| :param weight: contiene il peso della regola da utilizzare

| :param java_frt contiene il collegamento con JFML

|

| setAntecedent(self, value)

| Imposta il valore antecedente alle regole

| :param value: contiene il valore antecedente alle regole

|

| setConsequent(self, value)

| Imposta il valore conseguente alle regole

| :param value: contiene il valore conseguente alle regole

A.9 AntecedentType

class AntecedentType(builtins.object)

| Definisce le regole antecedenti del sistema fuzzy

|

| Methods defined here:

|

| __init__(self)

| Costruttore AntecedentType

| :param java_at: contiene il collegamento con JFML

|

| addClause(self, c)

| Aggiunge una lista di clausole nelle regole Antecedenti

| :param c: lista di clausole

A.10 ConsequentType

class ConsequentType(builtins.object)

| Definisce le regole conseguenti del sistema fuzzy

|

| Methods defined here:

|

| __init__(self)

| Costruttore ConsequentType

|

| addThenClause(self, c)

| Aggiunge una lista di clausole nelle regole conseguenti

| :param c: lista di clausole

A.11 ClauseType

class ClauseType(builtins.object)

| Definisce le clausole di un sistema fuzzy

|

| Methods defined here:

|

| __init__(self, variable, term, modifier=None)

| Costruttore ClauseType

| :param variable: contiene un oggetto KnowledgeBaseVariable

| :param term: contiene un oggetto FuzzyTerm

| :param modifier: contiene il modificatore di una stringa secondo lo StandardModifierType

| :param java_ct: contiene il collegamento con JFML

Bibliografia e Sitografia

[1] Barthélémy Dagenais, “Py4j – A Bridge between Python and Java”, https://www.py4j.org/, visitato il 14/03/2018

[2] Francesco Masulli, Appunti su Insiemi e Sistemi Fuzzy, Genova 2002-2003

[3] Mike Watts, “Computational Intelligence: Fuzzy Markup Language”, Computational-Intelligence.blogspot.it, visitato il 10/03/2018

[4] J.M. Soto-Hidalgo, Jose M. Alonso, Jesús Alcalá-Fdez, “Java Fuzzy Markup Language”, http://www.uco.es/JFML/, visitato il 12/03/2018

[5] Mohtashim M., “Artificial Intelligence – Fuzzy Logic System”, https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_fuzzy_logic_systems.htm, visitato il 18/03/2018