**“Software Engineering”**

**Course**

**a.a. 2016-2017**

**Template version 3.0**

**Lecturer: Prof. Henry Muccini (henry.muccini@univaq.it)**

**Planner Path Calculator version v3 Deliverables**

|  |  |
| --- | --- |
| **Date** | 10/02/2017 |
| **Deliverable** | 3 |
| **Team (Name)** | Comaiam |

|  |  |  |
| --- | --- | --- |
| **Team Members** | | |
| **Name & Surname** | **Matriculation Number** | **E-mail address** |
| Davide De Marco | 228737 | davide.demarco@student.univaq.it |
| Francesco Fuschi | 239266 | francesco.fuschi@student.univaq.it |
| Alice Laraspata | 235943 | alice.laraspata@student.univaq.it |
| Nicolò Paoletti | 235800 | nicolo.paoletti@student.univaq.it |
| Mario Vetrini | 238806 | mario.vetrini@student.univaq.it |

**Project Guidelines**   
**[do not remove this page]**  
This page provides the Guidelines to be followed when preparing the report for the Software Engineering course. You have to submit the following information:

* This Report
* Diagrams (Analysis Model, Component Diagrams, Sequence Diagrams, Entity Relationships Diagrams)
* Effort Recording (Excel file)

**Important:**

* document risky/difficult/complex/highly discussed requirements
* document decisions taken by the team
* iterate: do not spend more than 1-2 full days for each iteration
* prioritize requirements, scenarios, users, etc. etc.

**Project Rules and Evaluation Criteria**

**General information:**

* This homework will cover the 80% of your final grade (20% will come from the oral examination)
* The complete and final version of this document shall be no longer than 40 pages (excluding this page and the Appendix)
* Groups composed of seven students (preferably)

I expect the groups to submit their work through GitHub

**Use the same file to document the various deliverable.**   
**Document in this file how Deliverable “i+1" improves over Deliverable “i".**

**Project evaluation:**

Evaluation is not based on “quantity” but on “quality” where quality means:

* Completeness of delivered Diagrams
* (Semantic and syntactic) Correctness of the delivered Diagrams
* Quality of the design decisions taken
* Quality of the produced code

Table of Contents of this deliverable

[List of Challenging/Risky Requirements or Tasks 4](#_Toc474501421)

[A. Requirements Collection 5](#_Toc474501422)

[A.1 Functional Requirements 5](#_Toc474501423)

[A1.1 GUI Requirements 9](#_Toc474501424)

[A1.2 Business Logic Requirements 9](#_Toc474501425)

[A1.3 DB Requirements 10](#_Toc474501426)

[A.2 Non Functional Requirements 11](#_Toc474501427)

[A.3 Content 11](#_Toc474501428)

[A.4 Assumptions 11](#_Toc474501429)

[A.5 Prioritization 12](#_Toc474501430)

[B. Analysis Model 13](#_Toc474501431)

[C. Software Architecture 14](#_Toc474501432)

[C.1 Component Diagram 14](#_Toc474501433)

[C.2 Sequence Diagram 17](#_Toc474501434)

[D. E-R Design 21](#_Toc474501435)

[E. Class Diagram 23](#_Toc474501436)

[F. Design Decisions 28](#_Toc474501437)

[G. Explain how the FRs and the NFRs are satisfied by design 29](#_Toc474501438)

[H. Effort Recording 30](#_Toc474501439)

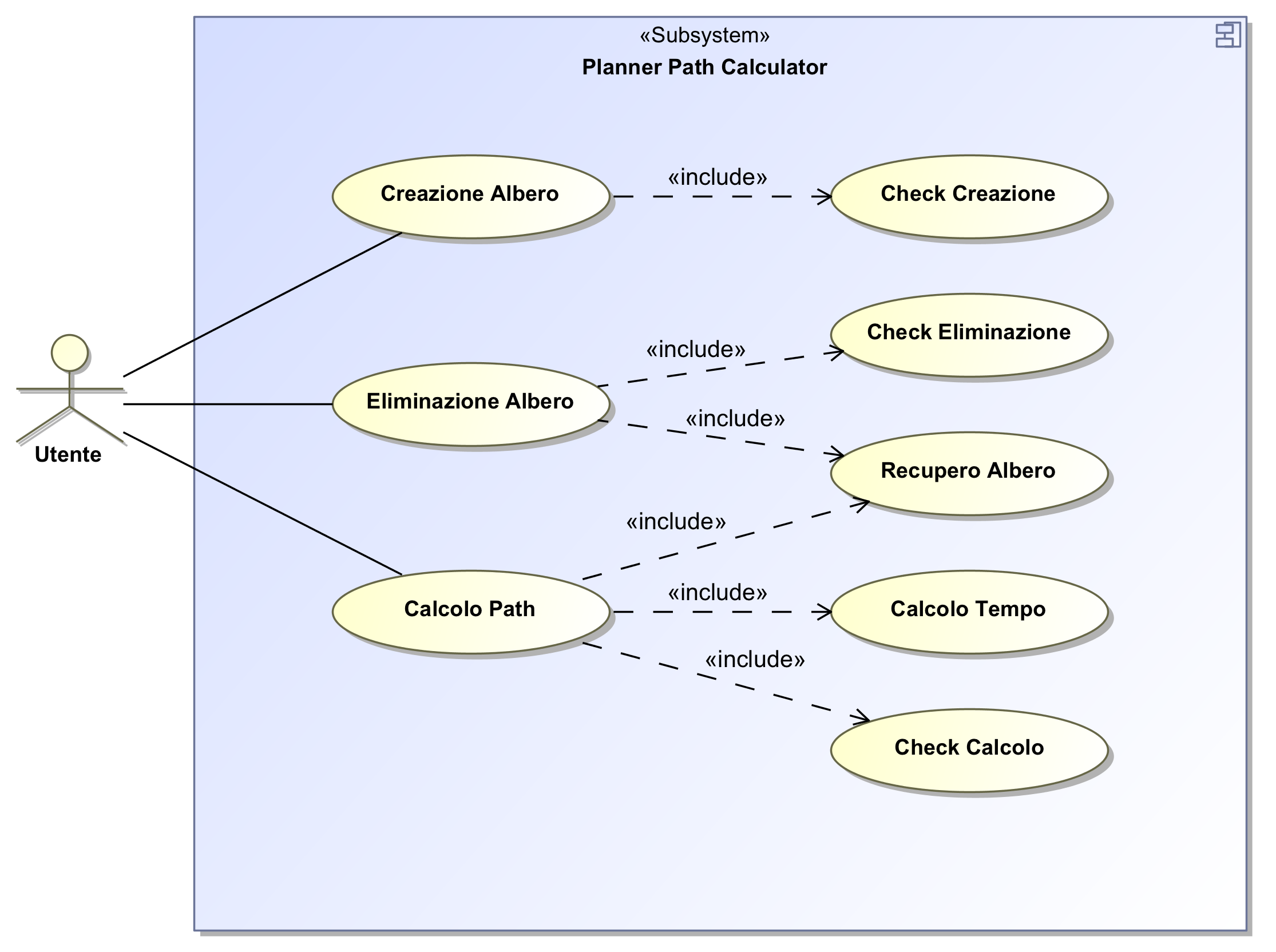
[Appendix Code 31](#_Toc474501440)

# List of Challenging/Risky Requirements or Tasks

|  |  |  |  |
| --- | --- | --- | --- |
| **Challenging**  **Task** | **Date the task is identified** | **Date the challenge is resolved** | **Explanation on how the** **challenge has been managed** |
| Scelta del metodo di calcolo del path | 14/11/2016 | 14/11/2016 | Valutando varie possibilità, si è ritenuto più conveniente (in termini di tempo e complessità) calcolare la somma degli attributi del path direttamente sul DB. |
| Dimenticanza da parte dell’utente del nome dell’albero per l’eliminazione e/o il calcolo del path | 15/11/2016 | 15/11/2016 | Si è deciso di strutturare la GUI in modo tale da permettere all’utente di ritrovare l’albero con facilità durante le fasi di eliminazione e di calcolo del path (ovvero visualizzando i nomi degli ultimi 10 alberi creati all’interno di una lista ordinabile per data e potendo ricercare un qualsiasi albero presente nel database tramite l’apposito tasto di ricerca). |
| Controllo di concorrenza nella cancellazione di un albero | 13/12/2016 | 13/12/2016 | Si è deciso di inserire un controllo da parte della BL, la quale dovrà verificare se l’albero selezionato dall’utente è ancora presente del DB. |
| Scelta del metodo di creazione dell’albero | 14/12/2016 | 14/12/2016 | Notando che la creazione dell’albero tramite l’utilizzo di una query richiedeva un tempo troppo elevato, si è deciso di utilizzare tre file CSV creati tramite PHP. |
| Scelta del metodo di eliminazione dell’albero | 12/01/2017 | 12/01/2017 | Notando che i tempi di eliminazione erano molto simili a quelli di creazione, si è deciso di creare un evento alla creazione del DB. Tale evento eliminerà ogni giorno (alle 1:00 di notte) ogni albero presente nel DB con attributo “usabilità” uguale a false. |

# A. Requirements Collection

## A.1 Functional Requirements



**FIGURA 1: USE CASE DIAGRAM**

Il Planner Path Calculator (PPC) è un sottosistema che permette di generare e/o rimuovere alberi che soddisfino determinate caratteristiche e condizioni ed effettuare su di essi calcoli e operazioni.

Per eseguire tali operazioni abbiamo bisogno di tre differenti sottosistemi del PPC:

* GUI
* Business Logic (BL)
* Database (DB)

Per ognuna delle tre funzionalità del sistema è incluso un diverso check di verifica della correttezza dei dati inseriti (suddiviso in GUI-check e DB-check). Il GUI-check e il DB-check verranno approfonditi successivamente nei rispettivi requisiti. Per il caso d’uso di “Calcolo Path” è incluso anche il “Calcolo Tempo”; verrà approfondito successivamente. Per i casi d’uso “Eliminazione Albero” e “Calcolo Path” è incluso il “Recupero Albero”; anch’esso verrà approfondito successivamente.

Di seguito vengono mostrate le descrizioni tabellari degli Use-case:

**Creazione Albero**

|  |  |  |
| --- | --- | --- |
| **USE CASE 1** | Creazione Albero | |
| **GOAL IN CONTEXT** | Far creare un albero ad un utente | |
| **SCOPE,**  **LEVEL** | Sistema PPC  Primary Task | |
| **PRECONDITIONS** | L’utente ha effettuato l’accesso al sistema | |
| **SUCCESS END CONDITION** | L’albero viene creato e l’utente riceve un messaggio di successo dalla GUI | |
| **FAILED END CONDITION** | L’albero non viene creato e l’utente riceve un messaggio di errore dalla GUI | |
| **PRIMARY** | Utente Micron | |
| **TRIGGER** | L’utente, nel menù della GUI, clicca su “Creazione Albero” | |
| **DESCRIPTION** | **STEP** | **ACTION** |
|  | **1** | L’utente, nel menù della GUI, clicca su “Creazione Albero” |
|  | **2** | Inserisce i dati di creazione attraverso la GUI |
|  | **3** | Clicca su “Crea Albero” |
|  | **4** | Riceve un messaggio di avvenuta creazione |
| **EXTENSIONS** | **STEP** | Branching Action |
|  | **3a** | Il nome dell’albero è già in uso:  Use Case “Check Creazione” |
|  | **3a** | I dati sono ancora vuoti o semanticamente errati:  Use Case “Check Creazione” |

|  |  |
| --- | --- |
| **RELATED INFORMATION** | Creazione Albero |
| **PRIORITY** | Alta |
| **PERFORMANCE** | Non espressamente dichiarate |
| **FREQUENCY** | 1 volta a settimana |
| **CHANNELS TO ACTORS** | Sistema PPC |
| **OPEN ISSUES** | Nessuno |
| **SUPERORDINATES** | Nessuno |
| **SUBORDINATES** | Check Creazione |

**Eliminazione Albero**

|  |  |  |
| --- | --- | --- |
| **USE CASE 2** | Eliminazione Albero | |
| **GOAL IN CONTEXT** | Far eliminare un albero ad un utente | |
| **SCOPE,**  **LEVEL** | Sistema PPC  Primary Task | |
| **PRECONDITIONS** | L’utente ha effettuato l’accesso al sistema ed è presente un albero da eliminare | |
| **SUCCESS END CONDITION** | L’albero viene eliminato e l’utente riceve un messaggio di successo dalla GUI | |
| **FAILED END CONDITION** | L’albero non viene eliminato e l’utente riceve un messaggio di errore dalla GUI | |
| **PRIMARY** | Utente Micron | |
| **TRIGGER** | L’utente, nel menù della GUI, clicca su “Eliminazione Albero” | |
| **DESCRIPTION** | **STEP** | **ACTION** |
|  | **1** | L’utente, nel menù della GUI, clicca su “Eliminazione Albero” |
|  | **2** | Seleziona il nome dell’albero nella lista attraverso la GUI |
|  | **3** | Clicca su “Elimina Albero” |
|  | **4** | Riceve un messaggio di avvenuta eliminazione |
| **EXTENSIONS** | **STEP** | Branching Action |
|  | **3a** | L’albero selezionato è già stato eliminato:  Use Case “Check Eliminazione” |

|  |  |
| --- | --- |
| **RELATED INFORMATION** | Eliminazione Albero |
| **PRIORITY** | Media |
| **PERFORMANCE** | Non espressamente dichiarate |
| **FREQUENCY** | Non espressamente dichiarata |
| **CHANNELS TO ACTORS** | Sistema PPC |
| **OPEN ISSUES** | Nessuno |
| **SUPERORDINATES** | Nessuno |
| **SUBORDINATES** | Check Eliminazione, Recupero Albero |

**Calcolo Path**

|  |  |  |
| --- | --- | --- |
| **USE CASE 3** | Calcolo Path | |
| **GOAL IN CONTEXT** | Far calcolare il path di un albero ad un utente | |
| **SCOPE,**  **LEVEL** | Sistema PPC  Primary Task | |
| **PRECONDITIONS** | L’utente ha effettuato l’accesso al sistema ed è presente un albero sul quale effettuare il calcolo del path | |
| **SUCCESS END CONDITION** | Il path viene calcolato e l’utente riceve un messaggio di successo (contenente il risultato) dalla GUI | |
| **FAILED END CONDITION** | Il path non viene calcolato e l’utente riceve un messaggio di errore dalla GUI | |
| **PRIMARY** | Utente Micron | |
| **TRIGGER** | L’utente, nel menù della GUI, clicca su “Calcolo Path” | |
| **DESCRIPTION** | **STEP** | **ACTION** |
|  | **1** | L’utente, nel menù della GUI, clicca su “Calcolo Albero” |
|  | **2** | Seleziona il nome dell’albero nella lista e inserisce il nome dei due vertici attraverso la GUI |
|  | **3** | Clicca su “Calcola Path” |
|  | **4** | Riceve un messaggio di successo (contenente il risultato) |
| **EXTENSIONS** | **STEP** | Branching Action |
|  | **3a** | L’albero selezionato è stato eliminato:  Use Case “Check Calcolo” |
|  | **3b** | L’albero selezionato non contiene quei vertici:  Use Case “Check Calcolo” |
|  | **3c** | L’albero selezionato non contiene quel path:  Use Case “Check Calcolo” |

|  |  |
| --- | --- |
| **RELATED INFORMATION** | Calcolo Path |
| **PRIORITY** | Media |
| **PERFORMANCE** | 30s con alberi da 1M di nodi, 60s con alberi da 2M di nodi |
| **FREQUENCY** | 3 volte al giorno |
| **CHANNELS TO ACTORS** | Sistema PPC |
| **OPEN ISSUES** | Nessuno |
| **SUPERORDINATES** | Nessuno |
| **SUBORDINATES** | Check Calcolo, Calcolo Tempo, Recupero Albero |

### A1.1 GUI Requirements

La GUI è l’interfaccia grafica che dà all’utente la possibilità di scegliere tra i tre servizi offerti dal PPC. A seconda della scelta, la pagina si aggiornerà in modo dinamico e mostrerà all’utente la rispettiva pagina per l'inserimento dei dati e la visualizzazione dei risultati.

* Per la creazione dell’albero, la GUI prende in input dall’utente i dati riguardanti le dimensioni e il nome dell'albero, il nome degli attributi (di vertici ed archi) e la regola di generazione dei loro valori (la quale può avere un range di valori stabilito dall'utente). La GUI esegue una prima fase di check sui dati inseriti (GUI-check). Nel caso in cui i dati immessi siano incorretti, la GUI mostra un messaggio di errore contenente i dati da dover modificare. In caso contrario, la GUI manda i dati alla BL; quest’ultima può restituire un messaggio di avvenuta creazione o un messaggio di errore. La GUI provvede a visualizzare tale messaggio all’utente. (**priorità alta**)
* Per l’eliminazione dell’albero, la GUI fornisce all’utente la lista ordinata (per nome e per data) degli ultimi 10 alberi presenti nel DB, nella quale l’utente può selezionare uno dei presenti o ricercare (tramite apposito tasto) un qualsiasi albero presente nel database ed eliminarlo. La GUI controlla che sia stato selezionato un albero (GUI-check). Nel caso in cui ciò non avvenga, la GUI mostra un messaggio di errore contenente il dato da dover selezionare. In caso contrario, la GUI manda il dato alla BL; quest’ultima può restituire un messaggio di avvenuta eliminazione o un messaggio di errore. La GUI provvede a visualizzare tale messaggio all’utente. (**priorità media**)
* Per il calcolo del path, la GUI fornisce all’utente la lista ordinata (per nome e per data) degli ultimi 10 alberi presenti nel DB. L’utente dovrà selezionare un albero tra i presenti o ricercarlo tramite l’apposito tasto di ricerca tra tutti gli alberi presenti nel database e inserire due dei suoi vertici. La GUI esegue una prima fase di check sui dati inseriti (GUI-check). Nel caso in cui i dati immessi siano incorretti, la GUI mostra un messaggio contenente i dati da dover modificare. In caso contrario, la GUI manda i dati alla BL; quest’ultima può restituire un messaggio di avvenuta operazione (contenente inoltre il risultato ed il tempo impiegato) o un messaggio di errore. La GUI provvede a visualizzare tale messaggio all’utente. (**priorità media**)

Specifichiamo che la tipologia di check effettuata dalla GUI è limitata alla sola correttezza dei tipi di dati inseriti (es. caselle lasciate vuote, alberi non selezionati, altezza o numero di figli negativo o decimale, ecc…).

**Assunzione**: Abbiamo deciso di attribuire priorità alta alla creazione dell’albero in quanto, senza tale operazione, gli altri due servizi non potrebbero essere eseguiti. Le priorità sono approfondite in modo dettagliato nella sezione “A.5 Prioritization”.

### A1.2 Business Logic Requirements

La Business Logic è il sottosistema che si occupa di effettuare o “lanciare” i servizi offerti dal PPC.

* Per la creazione dell’albero, la BL prende in input dalla GUI le dimensioni e il nome dell'albero, il nome degli attributi (di vertici ed archi) e la regola di generazione dei loro valori (compreso, se disponibile, il valore di range). La BL, in primo luogo, manda al DB il nome dell’albero per controllare che non sia già in uso. In caso non lo sia, la BL crea tre file CSV da inviare al DB; quest’ultimo può restituire un messaggio di avvenuta creazione o un messaggio di errore. La BL provvede a restituire tale messaggio alla GUI. (**priorità alta**)
* Per l’eliminazione dell’albero, la BL accede al DB recuperando la lista degli alberi eliminabili e li restituisce alla GUI. La BL prende in input dalla GUI il nome dell’albero da eliminare all’interno del DB e lo invia a quest’ultimo; il DB può restituire un messaggio di avvenuta cancellazione o un messaggio di errore. (**priorità media**)
* Per il calcolo del path, la BL accede al DB recuperando la lista degli alberi sui quali è possibile effettuare il calcolo del path e li restituisce alla GUI. La BL prende in input il nome dell’albero e due dei suoi vertici e li manda, in primo luogo, al DB per controllare che l’albero esista. In caso non lo sia, la BL restituisce alla GUI un messaggio di errore. In caso di avvenuta operazione, il DB restituisce un messaggio di successo (contenente inoltre il risultato) e la BL lo restituisce alla GUI (insieme al tempo impiegato). Altrimenti, un messaggio di errore. (**priorità media**)

**Assunzione**: Abbiamo deciso di attribuire priorità alta alla creazione dell’albero in quanto, senza tale operazione, gli altri due servizi non potrebbero essere eseguiti. Le priorità sono approfondite in modo dettagliato nella sezione “A.5 Prioritization”.

### A1.3 DB Requirements

Il DB è il sottosistema che si occupa di gestire e strutturare gli alberi creati dal PPC.

* Per la creazione dell’albero, il DB riceve in input dalla BL il nome dell’albero e verifica, in primo luogo, se non sia già in uso. In caso lo sia, Il DB restituisce alla BL un messaggio di errore, altrimenti le comunica che tale nome non è in uso. Il DB riceve così dalla BL i tre file CSV contenti l’albero e li struttura al suo interno, restituendo un messaggio di avvenuta creazione. Altrimenti, restituisce alla BL un messaggio di errore. (**priorità alta**)
* Per l’eliminazione, il DB restituisce alla BL la lista degli alberi eliminabili e riceve in input da quest’ultima il nome dell’albero e verifica, in primo luogo, se esiste (DB-check). In caso non esista, il DB restituisce alla BL un messaggio di errore, altrimenti imposta l’attributo “usabilità” a false, modifica il nome dell’albero con il suo rispettivo md5 (così da poter dare la possibilità ad un nuovo utente di creare un albero con il nome dell’albero appena eliminato) e restituisce alla BL un messaggio di avvenuta eliminazione (l’eliminazione reale avverrà alle 1:00 di notte successive all’eliminazione da parte dell’utente tramite un evento) (\*). (**priorità media**)
* Per il calcolo del path, il DB restituisce alla BL la lista degli alberi sui quali è possibile effettuare il calcolo del path e riceve in input da quest’ultima il nome dell’albero e due dei suoi vertici. Verifica, in primo luogo, se l’albero esiste (DB-check). In caso non esista, il DB restituisce alla BL un messaggio di errore. Il DB-check, inoltre, si occupa di verificare la presenza dei vertici all’interno dell’albero selezionato e di verificare la presenza di un path che li colleghi (DB-check). In caso di avvenuta operazione, il DB restituisce alla BL un messaggio di successo (contenente inoltre il risultato). Altrimenti, un messaggio di errore. (**priorità media**)

Specifichiamo che la tipologia di check effettuata dal DB riguarda l’integrità dei dati presenti al suo interno (es. verificare che l’albero da voler eliminare non sia stato già eliminato, verificare l’esistenza dei vertici per il calcolo del path, verificare l’esistenza di un cammino tra due vertici, ecc…).

**Assunzione**: Abbiamo deciso di attribuire priorità alta alla creazione dell’albero in quanto, senza tale operazione, gli altri due servizi non potrebbero essere eseguiti. Le priorità sono approfondite in modo dettagliato nella sezione “A.5 Prioritization”.

(\*) Tale evento verrà spiegato in modo approfondito nelle sezioni successive.

## A.2 Non Functional Requirements

* Il database non ha limiti nella memorizzazione di alberi
* Il sistema principale si occupa della sicurezza e della visibilità dei dati
* Il tempo di creazione di un albero non è rilevante data la rarità della sua esecuzione (una volta a settimana)
* Il tempo di calcolo del path su un albero da 2.000.000 di nodi deve essere inferiore ai 60 secondi. Oltre i 2.000.000 è accettabile un andamento esponenziale della velocità di calcolo.
* Il range (K-N) deve essere differente per ogni categoria di attributo. I valori possono essere sia interi che decimali
* Gli alberi possono essere visualizzati e cancellati da qualsiasi utente, non solo da chi li ha inizialmente generati
* Gli alberi, una volta salvati nel DB, non possono essere modificati
* Il DBMS da utilizzare deve essere MSSQL o open-source
* Non è necessario che l’albero supporti lo Split Size = 1

## A.3 Content

Dopo una prima fase di testing riguardante la creazione di un albero tramite query eseguita direttamente sul DB, ci si è accorti che tale operazione richiedeva un tempo superiore rispetto al previsto. Si è dunque deciso di optare per l’utilizzo del file CSV creato tramite PHP in modo da abbattere drasticamente i tempi. Tali CSV (tre, per la precisione) verranno inviati al DB per strutturarli al suo interno. Questa scelta ha comportato la perdita del dato gerarchico offerto da MSSQL. Di conseguenza, si è deciso di abbandonarne l’utilizzo ed utilizzare MySQL, essendo più conosciuto da parte del team.

Per la GUI si è deciso di utilizzare il framework CSS “Materialize”, il quale rende la GUI più user-friendly, e il template engine “Smarty”, per dividere la parte logica da quella visuale.

## A.4 Assumptions

* Assumiamo che non sia necessario fornire le credenziali di accesso al sistema in quanto è un sottosistema di un software in cui è già previsto un sistema di autenticazione
* Assumiamo che il PPC giri su un server Micron
* Assumiamo che il server vada in timeout dopo 30 minuti per una singola operazione (assumiamo ciò per evitare che un utente crei un albero di dimensioni troppo elevate che causi un blocco del server e per evitare che il server si blocchi prima della fine del processo di creazione ed eliminazione. Il team ha avuto problemi in fase di creazione con tempi di timeout del server troppo corti provocando la creazione di alberi incompleti sui quali non si potevano effettuare operazioni di calcolo del path o eliminazione)
* Assumiamo che la GUI non permetta all’utente di bloccare un processo una volta avviato
* Assumiamo che l’albero sia orientato dal basso verso l’alto (dal figlio al padre)
* Assumiamo che dalle 1:00 di notte alle 7:00 non sia in corso nessuna operazione di creazione, così da evitare che l’evento di eliminazione cancelli degli alberi in fase di strutturazione nel DB (e quindi con l’attributo “usabile” momentaneamente posto a false)
* Assumiamo che il PPC supporti alberi di taglia minore o uguale di circa 2.000.000 di nodi; sopra tale taglia non assicuriamo che i tempi delle varie operazioni vengano rispettati

## A.5 Prioritization

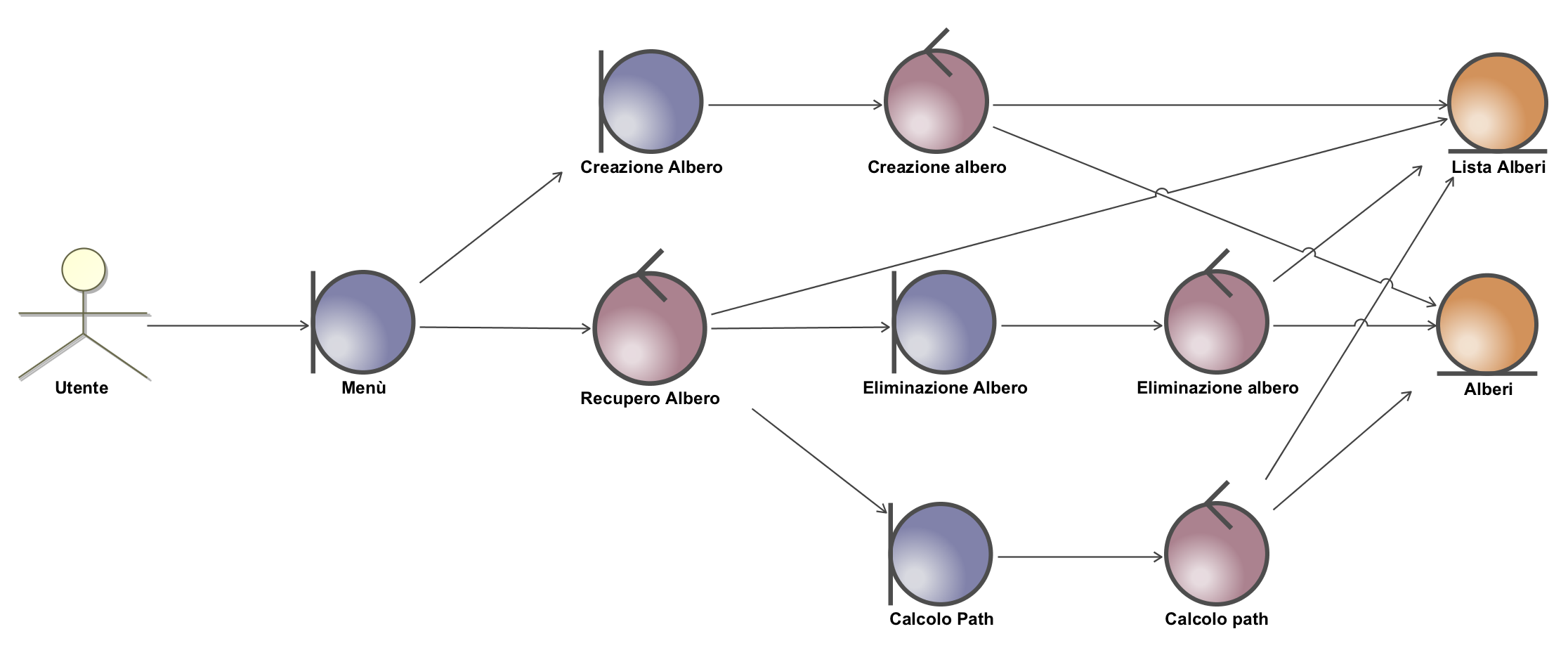
**PRIORITA’ USE-CASE PRINCIPALI**

|  |  |  |
| --- | --- | --- |
| **USE-CASE** | **PRIORITA’** | **SPIEGAZIONE** |
| Creazione Albero | Alta | Si è attribuita massima priorità in quanto, senza di essa, tutte le altre operazioni verrebbero meno. Risulta, inoltre, essere l’unica operazione nella quale la CONCORRENZA è rilevante ai fini della sua esecuzione. |
| Calcolo Path | Media | Pur essendo l’operazione centrale dell’intero PPC nella quale sono richieste prestazioni particolari e specifiche, si è scelto di attribuirle priorità media data la semplicità nel soddisfare tali prestazioni. |
| Eliminazione Albero | Media | Si è attribuita una elevata priorità (media) non per la difficoltà dell’esecuzione dell’operazione in sé, bensì per l’accurata gestione nell’evitare la concorrenza con altre operazioni. |

**PRIORITA’ USE-CASE INCLUSI**

|  |  |  |
| --- | --- | --- |
| **USE-CASE** | **PRIORITA’** | **SPIEGAZIONE** |
| Recupero Albero | Alta | Si è attribuita priorità massima in quanto è un’operazione che deve restituire la lista dei soli alberi sui quali è possibile effettuare operazioni. |
| Check Creazione | Media | Si è attribuita priorità media poiché deve effettuare semplici controlli dai quali dipende l’operazione di creazione. |
| Check Eliminazione | Media | Si è attribuita priorità media poiché, pur effettuando semplici controlli, risulta dipendere da esso l’esito dell’operazione di eliminazione. |
| Check Calcolo | Media | Si è attribuita priorità media poiché, pur effettuando semplici controlli, risulta dipendere da esso l’esito dell’operazione del calcolo del path. |
| Calcolo Tempo | Bassa | Si è attribuita priorità bassa in quanto, tra tutte le operazioni disponibili, risulta essere la più semplice da realizzare ed effettuare. |

# B. Analysis Model



**FIGURA 2: ROBUSTNESS DIAGRAM**

L’utente si interfaccia con il *boundary* del menù, dal quale può selezionare uno dei tre servizi offerti dal sistema PPC:

* Creazione Albero
* Eliminazione Albero
* Calcolo Path

Nel caso scelga il servizio “Creazione Albero”, l’utente si interfaccia con il *boundary* “Creazione Albero”, nel quale potrà inserire i dati relativi all’albero da voler creare. Si passa successivamente al *controller* “Creazione Albero”, il quale ne effettuerà la creazione e lo strutturerà nelle due *entities* “Lista Alberi” e “Alberi”.

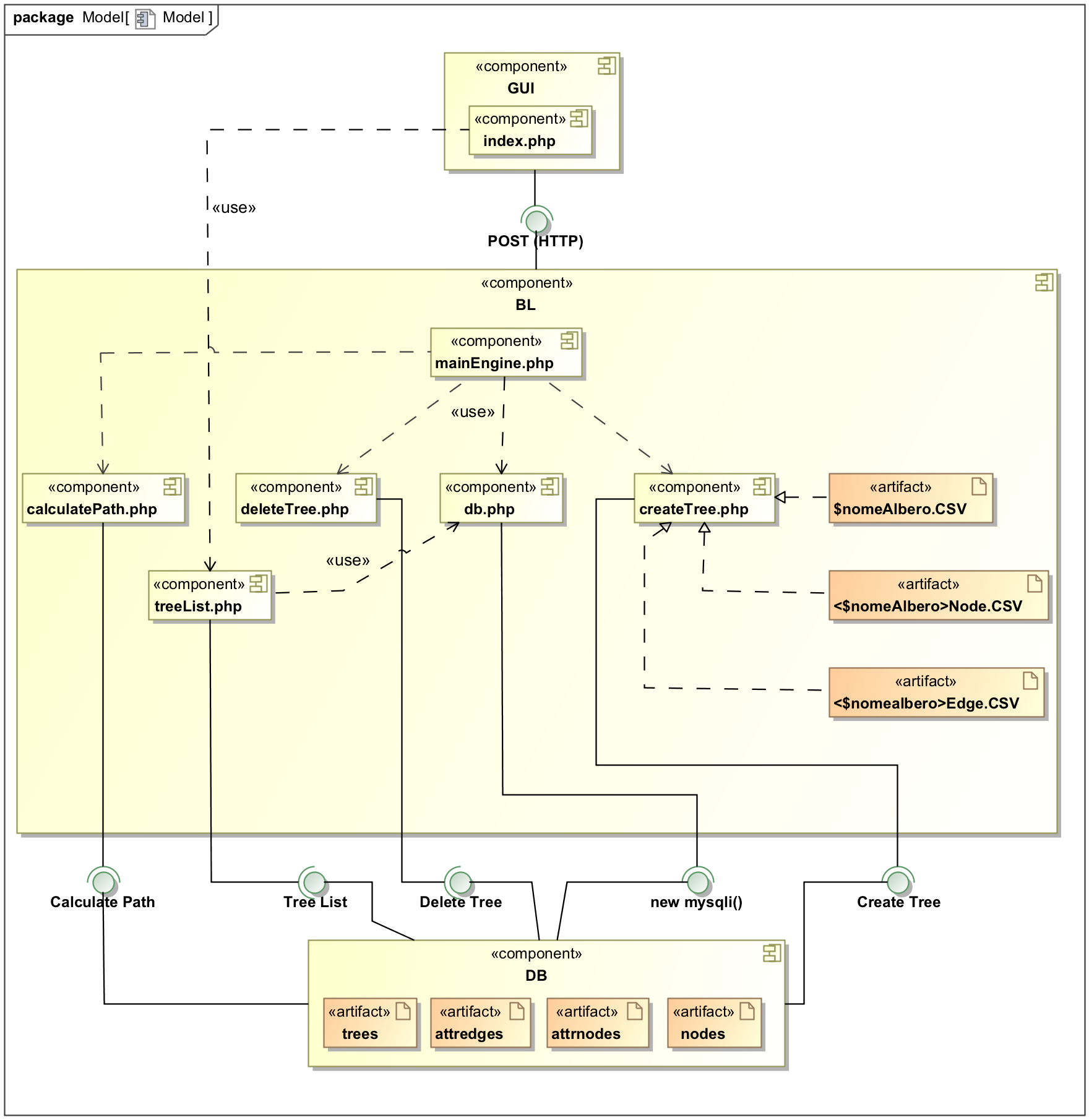
Nel caso scelga i servizi “Eliminazione Albero” e “Calcolo Path”, viene eseguito il *controller* “Recupero Albero”, il quale si occuperà di recuperare la lista degli alberi (accedendo alla *entity* “Lista Alberi”) sui quali è possibile effettuare le operazioni scelte. Il controllo poi passa ad uno dei due *boundaries*,rispettivamente “Eliminazione Albero” e “Calcolo Path”, nei quali l’utente potrà selezionare un albero e, nel caso del calcolo del path, anche i due vertici. Questi ultimi due *controllers* effettueranno delle operazioni sulle *entities* “Lista Alberi” e “Alberi”.

**Assunzioni:**

1. Nel Robustness Diagram non sono presenti i 3 *controllers* riguardanti il check degli errori poiché tali features sono gestite direttamente dai *controllers*. In più, le *entities* “Alberi” e “Lista Alberi” hanno al loro interno le business rules necessarie a controllarne l’integrità e la correttezza.
2. La *entity* “Alberi” è stata semplificata per una maggior chiarezza e semplicità all’interno del diagramma. Tale entità in realtà dovrebbe essere esplosa in tre *entities* quali: “Attributo\_Vertice”, “Vertice\_Arco” e “Attributo\_Arco”. Tali *entities* rispecchiano lo schema E-R (Sezione D. E-R Design).

# C. Software Architecture

## C.1 Component Diagram



**FIGURA 3: COMPONENT DIAGRAM**

Il Component Diagram mostra la divisione dell’intero sistema PPC in tre componenti:

* GUI
* BL
* DB

Le componenti presenti al loro interno e le interfacce verranno analizzate singolarmente in seguito;nella GUI è presente un solo componente (“index.php”). Tale pagina mostra un menù iniziale, nel quale l’utente può selezionare quale servizio utilizzare e la rispettiva form (il quale codice è presente sempre nella “index.php” per ridurre il numero di richieste di pagina).

Nel caso in cui l’utente scelga di eliminare un albero o di calcolare il path su di esso, la pagina “index.php” usa la pagina “treeList.php” che restituisce, tramite una query sul DB (accedendo alla tabella “trees”), gli alberi sui quali è possibile effettuare le operazioni e la “index.php” li visualizza all’utente.

Dopo che l’utente riempie la form e lancia l’operazione, tali dati vengono inviati tramite il protocollo HTTP in modalità POST alla pagina “mainEngine.php”; questa pagina innanzitutto avvia la connessione con il DB utilizzando la pagina “db.php”, creando un oggetto di tipo mysql e successivamente manipola i dati ricevuti dalla form della “index.php” e li invia alla rispettiva pagina del servizio selezionato, insieme ai dati relativi alla connessione in corso:

* Nel caso l’utente voglia calcolare il path, la pagina “mainEngine.php” invia alla pagina “calculatePath.php” i dati relativi all’operazione da eseguire e la pagina “calculatePath.php” invia una query al DB per accedere alle tabelle “trees”, “attredges”, “attrnodes” e “nodes”. Il calcolo del path viene eseguito direttamente sul DB
* Nel caso l’utente voglia eliminare un albero, la pagina “mainEngine.php” invia alla pagina “deleteTree.php” il dato relativo all’operazione da eseguire e la pagina “deleteTree.php” invia delle query al DB, le quali impostano l’attributo “usabile” uguale a false e modificano il nome dell’albero con il suo rispettivo md5 (così da poter dare la possibilità ad un nuovo utente di creare un albero con il nome dell’albero appena eliminato), accedendo alla sola tabella “trees”. L’evento creato alla creazione del DB eliminerà definitivamente l’albero alle 1:00 di notte successive all’eliminazione da parte dell’utente, accedendo alle tabelle “trees”, “attredges”, “attrnodes” e “nodes”
* Nel caso in cui l’utente voglia creare un albero, la pagina “mainEngine.php” invia alla pagina “createTree.php” i dati relativi all’operazione da eseguire. La pagina “createTree.php” crea i tre file CSV relativi all’albero, rispettivamente “<$nomealbero>.CSV”, “<$nomealbero>Node.CSV” e “<$nomealbero>Edge.CSV”. Una volta terminata la creazione, invia questi file tramite tre query al DB per poterli strutturare al suo interno nelle tabelle “trees”, “attredges”, “attrnodes” e “nodes” ed infine li elimina

Riguardo la gestione di cento utenti in parallelo, il problema della concorrenza viene risolto in maniera differente per le tre tipologie di operazioni:

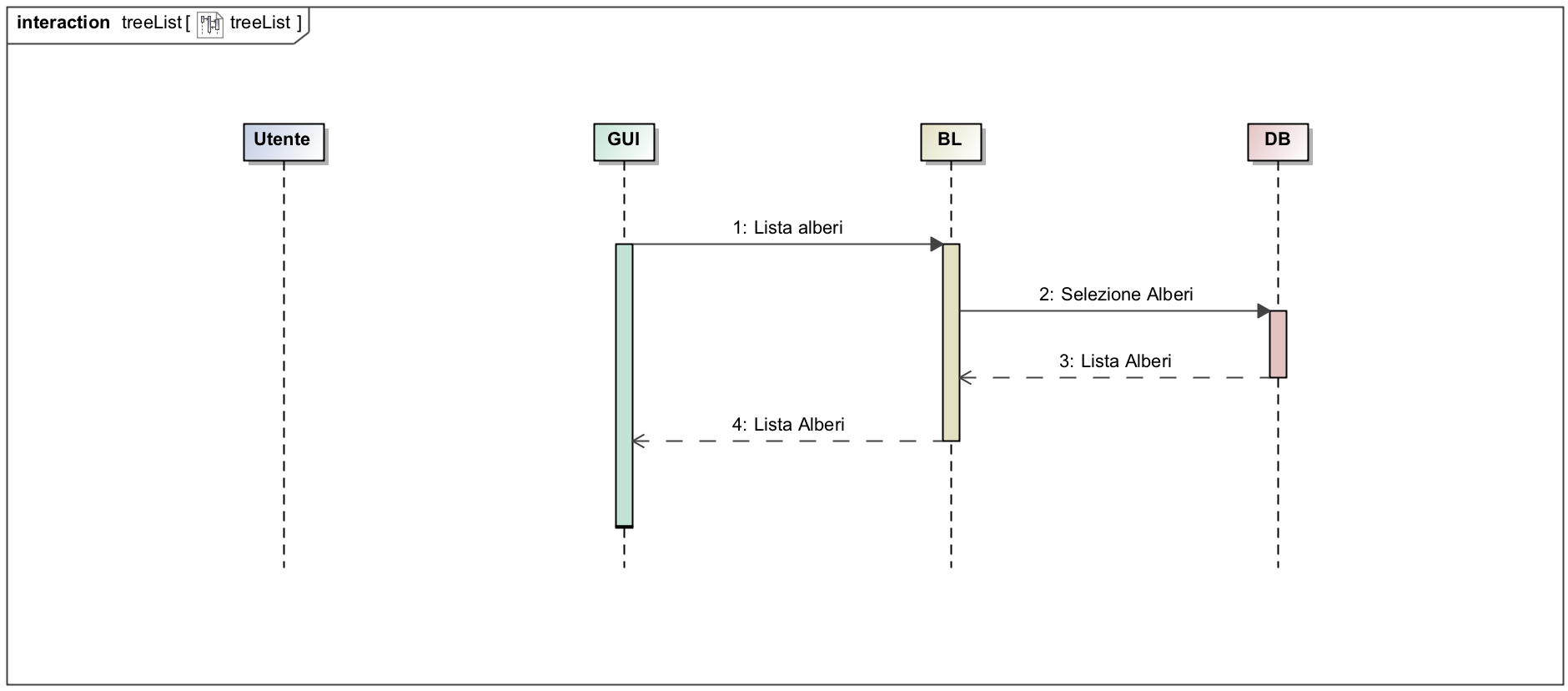
* Una volta che l’utente seleziona l’opzione per il calcolo del path o per l’eliminazione dell’albero, gli viene mostrata la lista degli alberi al momento disponibili per effettuare tali operazioni (con attributo “usabile” uguale a true). Se, nel tempo che intercorre tra la visualizzazione della lista e l’invio dell’operazione sull’albero selezionato, il dato albero non è più disponibile (è già stato eliminato o si sta calcolando il path su di esso), all’utente verrà mostrato un messaggio di errore. In caso contrario, l’attributo “usabile” sarà posto uguale a false e verrà avviata l’operazione di calcolo del path o di eliminazione dell’albero. Nel caso del calcolo del path, tale attributo verrà riposto uguale a true una volta terminata l’operazione sull’albero. Nel caso dell’eliminazione dell’albero, l’evento elimina effettivamente l’albero a partire dalle 1:00 di notte successive all’eliminazione da parte dell’utente così da non effettuarlo durante la giornata, evitando di creare concorrenza con altre operazioni
* Una volta che l’utente seleziona l’opzione per la creazione dell’albero e viene effettuato un primo controllo sui dati da egli inseriti (GUI-check), viene controllato che il nome dell’albero inserito dall’utente non esista già all’interno del DB. In caso ciò non avvenga e non ci siano altre operazioni in corso, l’operazione di creazione dell’albero viene effettuata immediatamente andando a creare i 3 file CSV e lanciando 3 query al DB. Se invece sono presenti altre operazioni in corso di esecuzione (soprattutto operazioni di creazione, le quali richiedono più tempo), viene comunque lanciata l’operazione ma le query verranno messe automaticamente in coda dal DB.

Per rispettare i requisiti non funzionali sul tempo di calcolo, abbiamo strutturato l’albero in modo tale che ogni nodo possieda un indice di riferimento al padre. Dato che, nel caso peggiore, un albero da due milioni di nodi con Split uguale a 2 (albero binario) ha altezza pari a venti (con altezza della root uguale a zero), si dovranno effettuare al massimo ventuno somme con complessità lineare. Ciò ci permette di rimanere ampiamente sotto ai tempi imposti dai requisiti non funzionali.

I tempi di calcolo e le varie ottimizzazioni in base ai requisiti hardware verranno trattati in modo approfondito nella sezione “Appendix Code”.

## C.2 Sequence Diagram

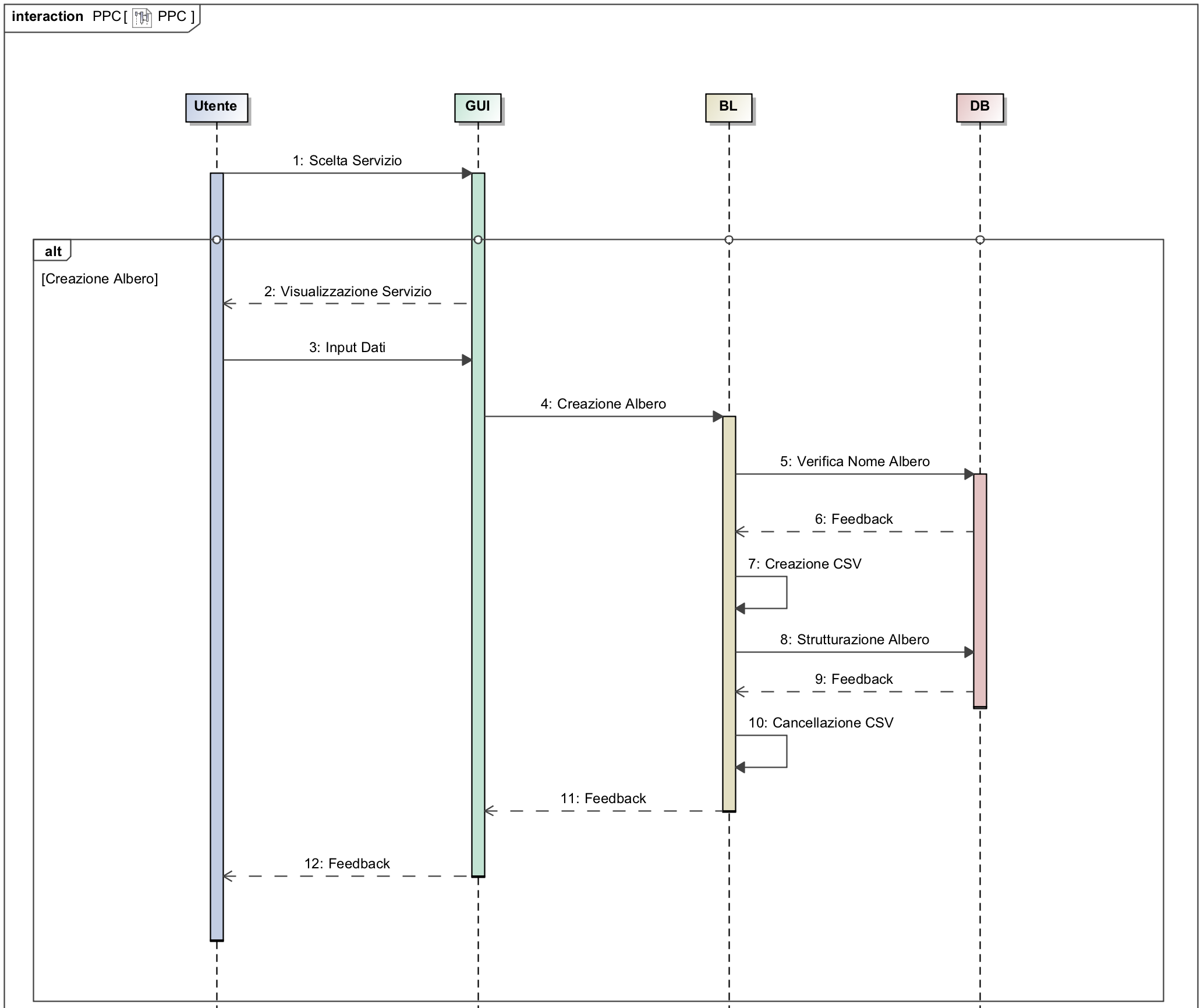
Prima di introdurre i Sequence Diagram riguardanti i tre principali servizi offerti dal PPC, mostriamo il Sequence Diagram relativo alla feature che permette al PPC di visualizzare la lista degli alberi disponibili per effettuare operazioni di eliminazione e calcolo del path.



**FIGURA 4: SEQUENCE DIAGRAM (treeList)**

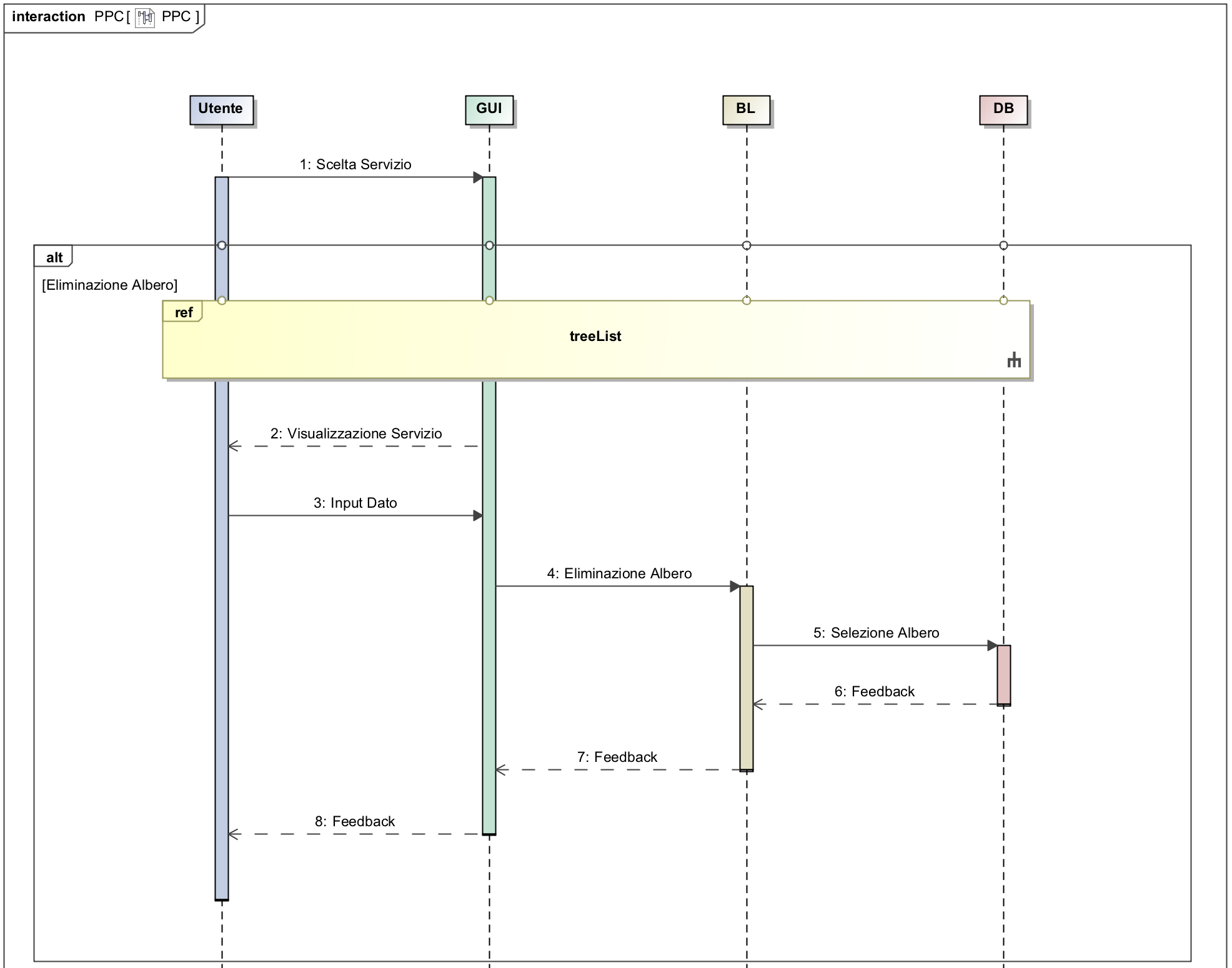
La GUI richiede la lista degli alberi alla BL (1), la quale accede al DB e seleziona, tramite una query, gli alberi con attributo “usabile” uguale a true (2). Il DB restituisce tali alberi alla BL (3) e quest’ultima li restituisce a sua volta alla GUI (4).

In seguito verranno mostrati singolarmente i 3 Sequence Diagram relativi ai tre principali servizi offerti dal PPC: “Creazione Albero”, “Eliminazione Albero” e “Calcolo Path” (per mancanza di spazio verranno suddivisi in 3 diagrammi differenti ma fanno comunque parte di un unico Sequence Diagram “PPC” nel quale sono presenti le tre alternative alla scelta del servizio da parte dell’utente).



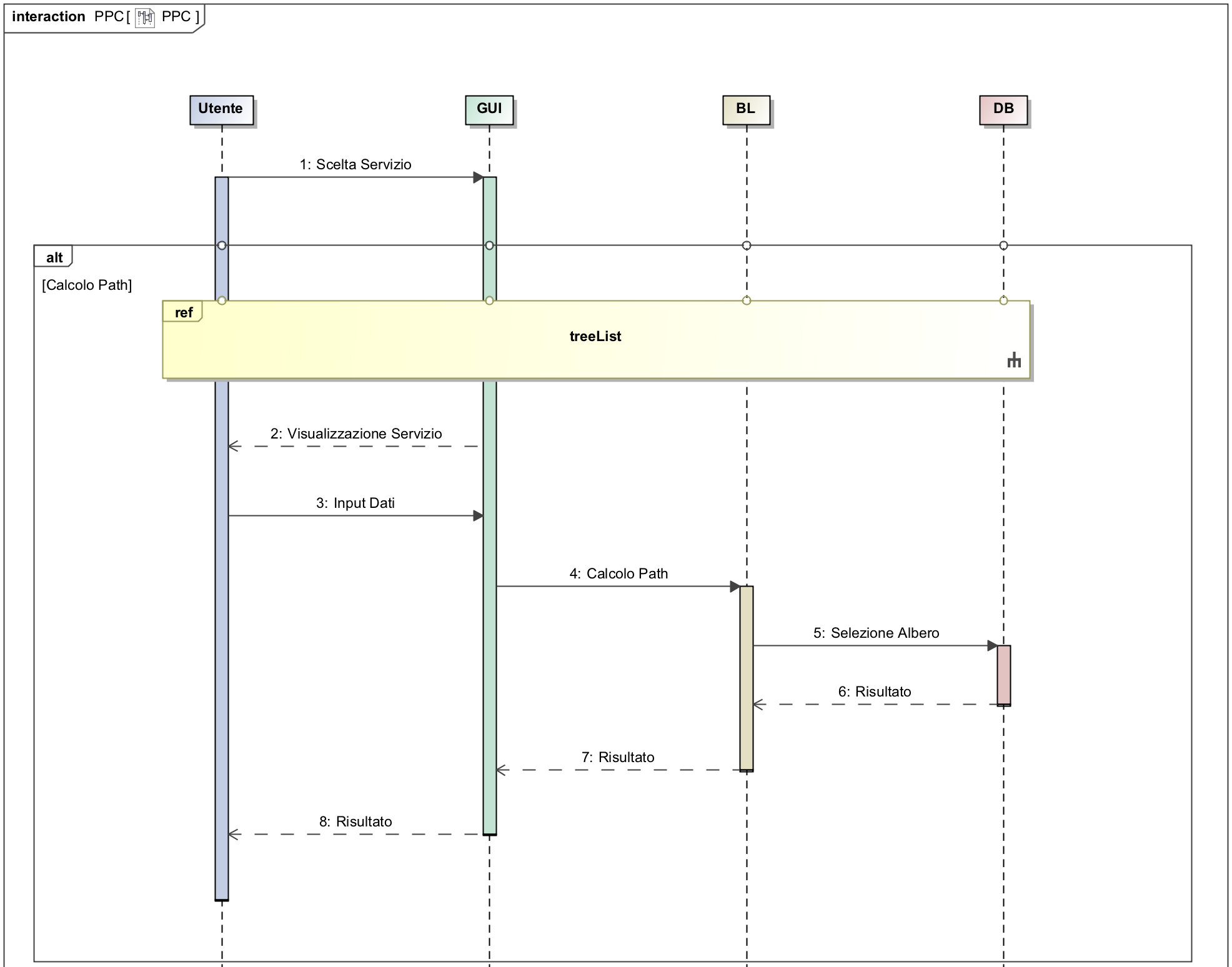
**FIGURA 5: SEQUENCE DIAGRAM (Creazione Albero)**

Dopo che l’utente richiede il servizio di “Creazione Albero” (1), gli viene mostrata la rispettiva form nella quale inserire i dati (2). Una volta inseriti (3), la GUI li manda alla BL (4). Quest’ultima verifica innanzitutto che il nome dell’albero non sia già in uso (5). Nel caso non lo sia (6), la BL crea i tre file CSV (7) e li invia al DB per strutturarli al suo interno (8). Una volta finita tale operazione (9), la BL elimina i tre file CSV (10) e restituisce un messaggio di avvenuta creazione alla GUI (11) che, a sua volta, lo comunica all’utente (12).



**FIGURA 5: SEQUENCE DIAGRAM (Eliminazione Albero)**

Dopo che l’utente richiede il servizio di “Eliminazione Albero” (1), viene eseguito il riferimento alla feature “treeList” (ref). La GUI mostra il servizio all’utente (2) comprensivo di lista in modo tale da permettergli di selezionare l’albero (3). La GUI invia tale dato alla BL (4) e quest’ultima, tramite una query, seleziona l’albero da eliminare (5). Il DB setta l’attributo “usabile” di tale albero uguale a false e restituisce un messaggio di avvenuta eliminazione alla BL (6), la quale lo comunicherà alla GUI (7) che a sua volta lo mostrerà all’utente (8).



**FIGURA 6: SEQUENCE DIAGRAM (Calcolo Path)**

Dopo che l’utente richiede il servizio di “Calcolo Path” (1), viene eseguito il riferimento alla feature “treeList” (ref). La GUI mostra il servizio all’utente (2) comprensivo di lista in modo tale da permettergli di selezionare l’albero e poter inserire i vertici (3). La GUI invia tali dati alla BL (4) che, tramite una query, seleziona l’albero (5). Il DB effettua il calcolo del path e restituisce un messaggio di successo (comprensivo di risultato) alla BL (6), la quale lo comunicherà alla GUI (7) che a sua volta lo mostrerà all’utente (8).

**Assunzione:** si è deciso di utilizzare esclusivamente chiamate SINCRONE. Quando dunque un oggetto invia un messaggio a un secondo, il mandante rimane IN ATTESA che l’oggetto ricevente ritorni senza poter proseguire in parallelo nella propria elaborazione. Ad esempio la GUI, una volta inseriti i dati dall’utente, blocca quest’ultimo impedendogli di usare altri servizi fino al totale completamento del processo. Ovviamente tale assunzione riguarda solamente un singolo utente.

# D. E-R Design



**FIGURA 7: SCHEMA E-R**

**Premessa:** si è deciso di non seguire lo schema presente all’interno del documento del progetto a favore di una migliore ottimizzazione del DB. Le entità “Attributo Nodo” e “Attributo Arco”, pur essendo uguali, non sono state unite per facilitare l’importazione da parte del DB di due file CSV distinti, per un totale di tre file CSV.

La prima entità che risulta fondamentale da definire è “Albero”. Tramite essa è possibile salvare all’interno del DB gli attributi che ne descrivono l’anagrafica e la struttura: “ID Albero” (chiave primaria), “Nome Albero”, “Usabilità” (booleano che specifica la possibilità o meno di utilizzare tale albero da parte di un utente per eseguire servizi e gestire la concorrenza), “Split Size”, “Depth” e “Data Creazione”.

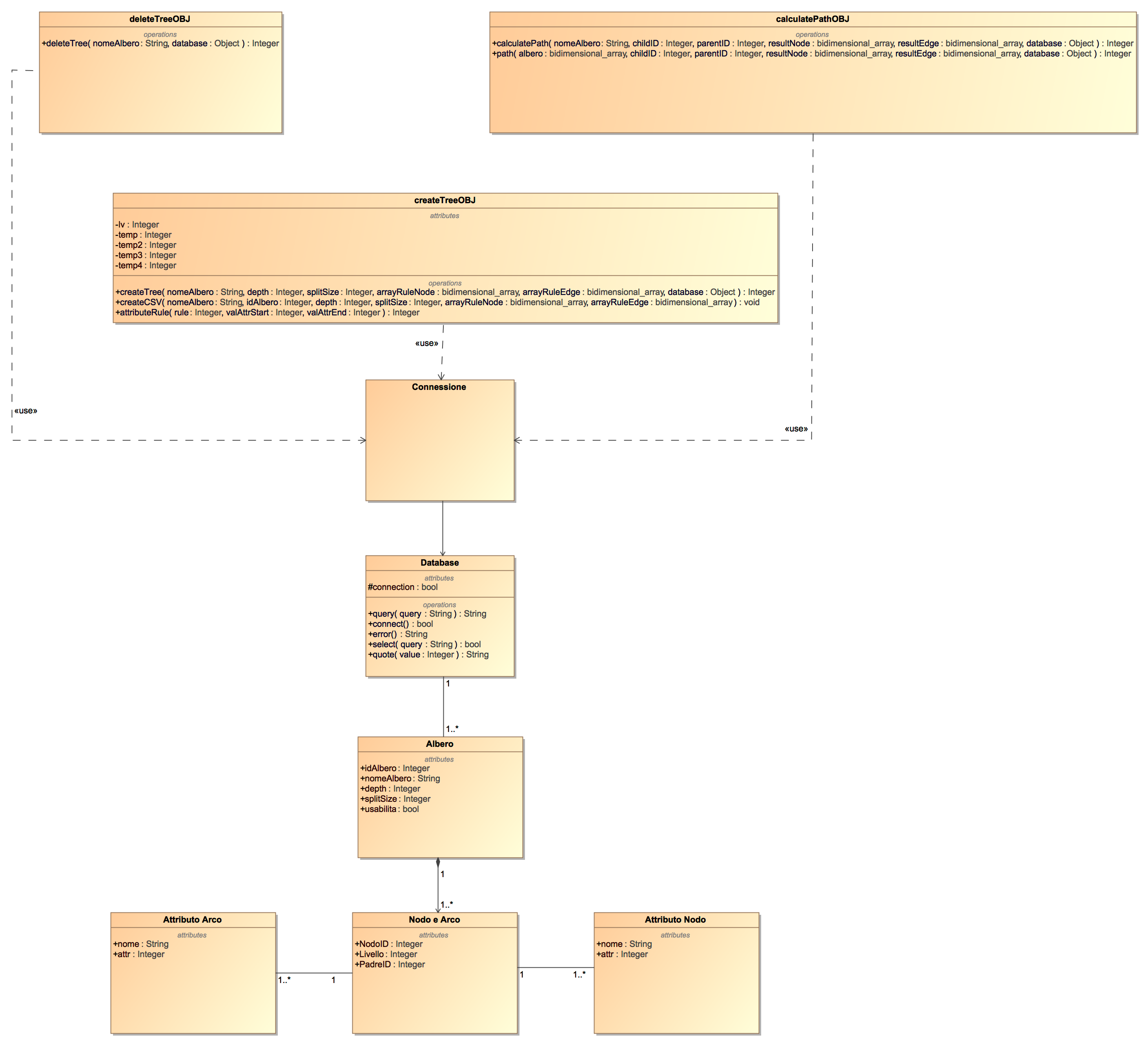
Tale entità risulta essere legata tramite la relazione “Comporre” all’entità “Nodo e Arco”, a sua volta caratterizzata da una relazione ricorsiva (“Gerarchia”) che permette di conoscere la relazione padre-figlio che caratterizza tutti i nodi di un generico albero.

L’entità “Nodo e Arco” ha i seguenti attributi che la caratterizzano: “ID Nodo Arco” e “Livello” (altezza di un nodo in un albero, con l’altezza della root uguale a zero). La chiave primaria multipla è rappresentata dalla coppia di attributi “ID Nodo Arco” e “ID Albero”, ovvero l’albero di appartenenza. Risulta chiaro che ad un albero possono corrispondere da 1 a N nodi (1 è il numero minimo per poterlo definire tale), mentre un generico nodo ha sempre uno e un solo padre (fatta eccezione per la root) cosi come può avere da zero (nel caso di una foglia) a N figli.

L’entità “Nodo e Arco”, oltre alla relazione con l’entità “Albero”, è caratterizzata anche da altre due relazioni, entrambe chiamate “Caratterizzare”, l’una con “Attributo Nodo” e l’altra con “Attributo Arco”.

Entrambe le entità “Attributo Nodo” e “Attributo Arco” sono costituite dai seguenti due attributi: “Valore” (che specifica qual è il valore numerico di un determinato attributo di uno specifico nodo o arco), e rispettivamente “Nome Attr. Nodo” per l’entità “Attributo Nodo” e “Nome Attr. Arco” per l’entità “Attributo Arco”. In più, l’entità “Attributo Arco” presente un ulteriore attributo: “Val Padre”, che risulta essere una ridondanza utile alla semplificazione della complessità dell’operazione di calcolo del path; in alternativa, si sarebbe dovuta effettuare una join che avrebbe inciso negativamente sui tempi di calcolo. Entrambe le entità sono caratterizzate dalle chiavi esterne, nonché chiavi primarie stesse, con l’ID di “Nodo e Arco”. È evidente che un generico nodo o arco può essere caratterizzato da 1 a N attributi cosi come un attributo può corrispondere ad uno ed un solo arco o nodo.

# E. Class Diagram



**FIGURA 8: CLASS DIAGRAM**

Descriviamo, in seguito, i componenti del Class Diagram:

**Classe “Albero”**

Attributi**:**

* **IDAlbero**: codice intero positivo identificativo dell’albero, assegnato automaticamente seguendo un ordine crescente al momento della sua creazione.
* **nomeAlbero**: nome, di tipo stringa, assegnato all’albero dall’utente in fase di creazione dello stesso.
* **depth:** numero intero positivo rappresentante l’altezza dell’albero.
* **splitSize:** numero intero positivo che rappresenta il numero di figli di ciascun nodo (ad eccezione delle foglie).
* **usabilita:** booleano indicante la disponibilità dell’albero (ad esempio un utente non può effettuare operazioni di cancellazione o di calcolo del path di un albero in creazione poiché la sua usabilità è settata a “false”).

Relazioni con altre classi:

La classe “Albero” è caratterizzata da una relazione (1,N) con la classe “Nodo e Arco” poiché, banalmente, un albero può essere composto da un nodo (minimo) fino a N nodi. Tale classe presenta inoltre una relazione (1,1) con “Database” dato che ogni albero è salvato in un solo database.

**Classe “Nodo e Arco”**

Attributi:

* **IDNodoArco:** codice intero positivo identificativo del nodo/arco che viene assegnato automaticamente secondo un ordine crescente al momento della creazione dell’albero**.**
* **livello:** numero intero positivo che indica il livello (altezza) in cui si trova il nodo.

Relazioni con altre classi:

La classe “Nodo e Arco” ha una relazione (1,N) con “Attributo nodo” e “Attributo arco”, poiché per ogni nodo/arco esiste da un minimo di uno ad un massimo di N attributi per nodo/arco.

La classe “Nodo e Arco” ha inoltre una relazione (1,1) con “Albero”, poiché ciascun nodo e ciascun arco fanno parte di uno e un solo albero.

**Classe “Attributo Nodo”**

Attributi:

* **nomeAttributoNodo:** nome, di tipo stringa, assegnato all’attributo dall’utente al momento della creazione dell’albero.
* **valore:** numero intero positivo generato a seguito della scelta da parte dell’utente di una delle funzioni per la generazione di valore messe a disposizione dal sistema.

Relazioni con altre classi:

La classe “Attributo Nodo” ha una relazione (1,1) con “Nodo e Arco” poiché ogni attributo fa parte di uno e un solo nodo.

**Classe “Attributo Arco”**

Attributi:

* **nomeAttributoArco:** nome, di tipo stringa, assegnato all’attributo dall’utente al momento della creazione dell’albero.
* **valore:** numero intero positivo generato a seguito della scelta da parte dell’utente di una delle funzioni per la generazione di valore messe a disposizione dal sistema.

Relazioni con altre classi:

La classe “Attributo Arco” ha una relazione (1,1) con “Nodo e Arco”, poiché ogni attributo fa parte di uno e un solo arco.

**Classe “createTreeOBJ”**

Attributi:

* **lv:** variabile di tipo intero che identifica il livello in cui si trova l’albero in un determinato istante.
* **temp:** variabile temporale per l’assegnazione di regole agli attributi di un nodo.
* **temp2:** variabile temporale per l’assegnazione di regole agli attributi di un nodo.
* **temp3:** variabile temporale per l’assegnazione di regole agli attributi di un nodo.
* **temp4:** variabile temporale per l’assegnazione di regole agli attributi di un nodo.

Metodi:

* **createTree (nomeAlbero: String, depth: Integer, splitSize: Integer, arrayRuleNode: array[][], arrayRuleEdge: array[][], database: Object):** funzione di creazione dell’albero che prende in input il suo nome, la sua altezza, il numero di figli e le coppie attributo-valore dei nodi e degli archi. La funzione ritorna uno dei seguenti valori:
* **3:** errore di connessione
* **2:** albero già esistente
* **1:** errore in fase di creazione dell’albero
* **0:** albero creato con successo
* **createCSV (nomeAlbero: String, idAlbero: Integer, depth: Integer, splitSize: Integer, arrayRuleNode: [][], arrayRuleEdge: [][]):** funzione di creazione del file CSV che prende in input il suo nome, la sua altezza, il numero di figli e le coppie attributo-valore dei nodi e degli archi. Il metodo non presenta valori di ritorno.
* **attributeRule (rule: Integer, valAttrStart: Integer, valAttrEnd: Integer):** funzione che permette all’utente di scegliere tra le tre regole fornite di default per la generazione del valore degli attributi. La funzione ritorna un intero, ovvero il valore dell’attributo del nodo o arco.

Relazioni con altre classi:

La classe “createPathOBJ” utilizza la “Connessione” per accedere al DB.

**Classe”calculatePathOBJ”**

Metodi:

* **calculatePath (nomeAlbero: String, childID: Integer, parentID: Integer, resultNode: array[][], resultEdge: array[][], database: Object ):** funzione di calcolo della somma degli attributi in un dato path che prende in input il nome dell’albero, il nodo di destinazione, il nodo di partenza e due variabili passategli per riferimento dove verranno restituite la somma dei nodi e degli archi come array bidimensionali. La funzione, dopo aver controllato la connessione al db e che l’albero sia utilizzabile, ritorna uno dei seguenti valori:
* **3:** errore di connessione
* **2:** albero inesistente
* **1:** nodo inesistente
* **0:** path inesistente

La funzione inoltre modifica le variabili “resultNode” e “resultEdge” passate per riferimento alla funzione.

* **path: (albero: array[][], childID: Integer, parentID: Integer, resultNode: array[][], resultEdge: array[][], database: Object):** funzione di calcolo della somma degli attributi in un dato path che prende in input il nome dell’albero, il nodo di destinazione, il nodo di partenza e due variabili passategli per riferimento dove verranno restituite la somma dei nodi e degli archi come array bidimensionali. La funzione ritorna uno dei seguenti valori:
* **1:** nodo inesistente
* **0:** path inesistente

La funzione inoltre modifica le variabili “resultNode” e “resultEdge” passate per riferimento alla funzione.

Relazioni con altre classi:

La classe “calculatePathOBJ” utilizza la “Connessione” per accedere al DB.

**Classe “deletePathOBJ”**

Metodi:

* **deleteTree (nomeAlbero: String, database: Object):** funzione di cancellazione dell’albero che prende in input il nome dell’albero. La funzione ritorna uno dei seguenti valori:
* **3:** errore di connessione
* **2:** albero inesistente
* **1:** eliminazione avvenuta con successo
* **0:** errore in fase di eliminazione

Relazioni con altre classi:

La classe “deletePathOBJ” utilizza la “Connessione” per accedere al DB.

**Classe “Database”**

Attributi:

* **connection:** variabile booleana per il controllo di connessione con il DB.

Metodi:

* **query (query: String):** funzione che, passando una query al database, ritorna “false” se la connessione al DB non ha avuto successo. In caso contrario, invece, ritorna il risultato della query.
* **select (query: String):** funzione di selezione che ritorna, sotto forma di array, in caso di successo il risultato della query, altrimenti restituisce “false”.
* **error():** funzione che ritorna il messaggio di errore del database.
* **quote (value: Integer):** funzione che skippa i caratteri speciali per la query.

Relazioni con altre classi**:**

La “Connessione” è dipendente da “Database” poiché, senza una connessione avvenuta con successo con quest’ultimo, non è possibile accedere a nessun servizio offerto dal PPC.

“Database” ha inoltre una relazione (1,N) con “Albero” poiché banalmente nel database sono salvati da 1 a N alberi.

# F. Design Decisions

1. **CREAZIONE ALBERO:**

La creazione di un albero viene effettuata dalla BL con l’utilizzo di file .CSV. Tale scelta è giustificata in modo approfondito nel punto “A.3 Content”.

1. **SCELTA DBMS:**

Il DBMS che utilizzeremo è MySQL. Tale scelta è giustificata in modo approfondito nel punto “A.3 Content”.

1. **CALCOLO PATH:**

Il calcolo del path viene eseguito tramite una query sul database. Abbiamo ritenuto troppo complesso utilizzare un linguaggio di programmazione (es. Java) per accedere al DB e selezionare il path corrispondente per copiarlo in locale ed effettuarne la somma. Tale scelta è giustificata dalla semplicità con la quale si possono effettuare query riguardanti la somma del path direttamente in SQL.

1. **ATTRIBUTO “Usabilità”:**

Sull’entità “Albero” è presente l’attributo “Usabilità”. Tale attributo ci permette di gestire la concorrenza in fase di creazione, eliminazione e calcolo del path. Tale scelta è giustificata in modo approfondito nel punto “C.1 Component Diagram”.

1. **ENTITA’ “Nodo e Arco”:**

Nel database non sono presenti due entità “Nodo” e “Arco”, ma solo un’unica entità “Nodo e Arco”. Tale scelta è giustificata in modo approfondito nel punto “D. E-R Design”.

Di seguito vengono elencate le Design Decisions che abbiamo ritenuto meno importanti rispetto alle cinque presenti in questa sezione ma che verranno usate nella sezione G per soddisfare diversi requisiti:

1. **GUI-CHECK:**

La GUI effettuerà un check iniziale sui dati inseriti dall’utente. Tale check potrà però riguardare esclusivamente il valore dei relativi dati (es. caselle lasciate vuote, altezza o numero di figli negativo o decimale, ecc…). Tutto ciò su cui la GUI non può effettuare alcun controllo, verrà riservato al DB che si occuperà di verificare la presenza di eventuali errori (es. verificare che l’albero da voler eliminare non sia stato già eliminato, verificare l’esistenza dei vertici in un determinato albero durante il calcolo del path, verificare l’esistenza di un cammino tra i due vertici, ecc…).

1. **GUI DINAMICA:**

La GUI non è composta da pagine differenti, ma sono tutte su una stessa pagina dinamica. L’utente si troverà di fronte ad un’unica pagina iniziale dove verranno visualizzate tre scelte: “Creazione albero”, “Eliminazione albero” e “Calcolo path”. A seconda della scelta, verrà visualizzata l’interfaccia corrispondente. Utilizzando una pagina dinamica, si riesce ad alleggerire il carico sul server in quanto in essa sono presenti tutte le features necessarie, riducendo così le richieste di pagina.

# G. Explain how the FRs and the NFRs are satisfied by design

|  |  |
| --- | --- |
| **Requisiti** | **Design Decision** |
| A1.1 GUI Requirements | F.6, F.7 |
| A1.1.3 Calcolo Path | F.3 |
| A1.3 DB Requirements | F.2, F.6 |
| A1.2.1 Creazione Albero | F.1 |
| Tempo calcolo path (NF) | F.3 |
| Scelta del DBMS (NF) | F.2 |

Elenchiamo in seguito i restanti requisiti non funzionali e le relative Design Decisions non elencate nella sezione F del deliverable:

1. “Il range (K-N) deve essere differente per ogni categoria di attributo. I valori possono essere sia interi che decimali.”. Tale requisito non funzionale è soddisfatto dalla GUI, la quale permette all’utente di inserire diverse categorie di attributo ognuna con il proprio valore.
2. “Gli alberi possono essere visualizzati e cancellati da qualsiasi utente, non solo da chi li ha inizialmente generati.”. Tale requisito non funzionale è soddisfatto innanzitutto dalla GUI che permette all’utente di selezionare qualsiasi albero all’interno del DB, e in secondo luogo dal DB stesso il quale non memorizza, all’interno di un generico albero, l’utente che lo ha creato.
3. “Gli alberi, una volta salvati nel DB, non possono essere modificati.”. Tale requisito non funzionale è soddisfatto dalla GUI, la quale non permette all’utente di modificare un albero precedentemente creato.

Non abbiamo assunto nessun’altra Design Decision per i restanti requisiti non funzionali non elencati.

# H. Effort Recording

**GANTT**  
Make a GANTT documenting the tasks and timing you expect to spend on the deliverable. Try to be as precise as possible. Check, after the deliverable deadline, if and how you satisfied (or not) the deadlines.

**Logging**   
As you are working on the assignment, record what you are doing and how long you spent. As a rule of thumb, you should add a log entry every time you switch tasks. For example, if you do something for two hours straight, that can be one log entry. However, if you do two or three things in half an hour, you must have a log entry for each of them. You do not need to include time for logging, but should include the time spent answering the other parts of this question.

For this purpose, please use the **LogTemplate.xls** file.

**Categorization**  
When logging the time spent on the project, please create different sub- categories.Specifically, it is important to clearly distinguish between two main categories: the time spent for “**learning**” (the modeling languages, the tools, etc.) from the time needed for “**doing**” (creating the models, taking the decisions, …). Learning tasks are in fact costs to be paid only once, while doing costs are those that will be repeated through the project.

For each category, please define sub-categories. Examples follow. You may add other sub-categories you find useful.

|  |  |
| --- | --- |
| ***Learning***   * ***Requirements Engineering*** * ***Non Functional Requirements*** * ***Use Case Diagrams*** * ***Tool study*** | ***Doing:***   * ***Requirements discovery*** * ***Requirements Modeling (UC diagrams)*** |

**Summary Statistics**

Based on the attributes defined above, calculate the summary statistics of the time spent for “learning”, the time spent for “doing”, and the total time.

**Note: this Deliverable report shall document only the Summary Statistics for the different deliverables (D1, D2, and Final). Detailed information shall be reported in the Excel file.**

**COPY HERE (computed from the spreadsheet): i) the total number of hours spent by the group (that is, hours per task X number of people working on that task), ii) the time spent for LEARNING and for DOING**

Totale ore spese per il Deliverable v3: 61 ore di cui 5 spese per il learning e 56 per il doing.

# Appendix Code

*Nel seguito viene riportata una breve descrizione del codice. Per maggiori approfondimenti attenersi alla documentazione data nei file tramite commenti. Inoltre un breve tutorial su come usare il sistema è stato collocato all'interno del prototipo come file README.txt*

**CreateTree.php**

Nel file createTree.php viene riportato il codice di creazione dell'albero. All'interno viene dichiarata la classe createTreeOBJ contenente i seguenti metodi:

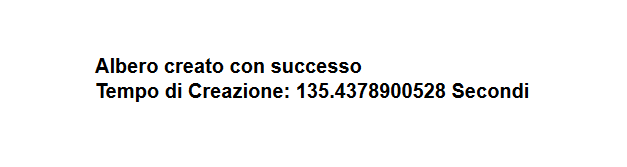
* **createTree()** è il metodo principale dove vengono richiamati gli altri metodi privati. Inizialmente viene fatto un controllo sulla presenza di connessione e sull'esistenza dell'albero e solo in seguito viene richiamato il metodo createCSV(). Conclusa l'esecuzione di quest’ultimo, createTree() si occupa dell'inserimento dei 3 file CSV nel DB. Tale procedura viene fatta in maniera sequenziale in quanto si è notato che con accessi simultanei allo stesso DB non si aveva un miglioramento dei tempi. Il metodo infine restituirà dei valori per informare dell'avvenuta creazione mentre, in caso di errore, cancellerà tutto quello che è stato creato fino a quel momento non lasciando così traccia nel DB di alberi incompleti.
* **createCSV()** è il metodo che si occupa della creazione dei file CSV. Come detto in precedenza, dopo accurate analisi si è scelto di creare 3 differenti file CSV (uno per i nodi, uno per gli attributi degli archi e uno per gli attributi dei nodi). Tali file vengono creati in parallelo nel metodo in modo tale da ottimizzare i tempi e non ripetere le stesse azioni troppe volte. Durante la creazione di tali file viene richiamato ciclicamente il metodo attributeRule() per inserire gli attributi nel modo corretto e tramite regole prestabilite gli attributi nei nodi e nei vertici.
* **attributeRule()** è il metodo che si occupa della gestione degli attributi. L'utente può scegliere tra 4 regole prestabilite (“Random”, “Valore prestabilito”, “Esponenziale per livello” e “Crescente per nodo”). Tale metodo esegue i calcoli in modo corretto e restituisce il valore quando chiamato.

Il sistema così creato è stato frutto di studi accurati sui tempi. Ci era stato detto che i tempi di creazione non avevano rilevanza ma il team ha sostenuto che una lunga attesa da parte degli utenti non era l’ideale. Tutto ciò è scaturito dal fatto che il sistema deve poter gestire 100 operazioni di creazione in contemporanea.

In seguito verranno mostrati i tempi di creazione rispettivamente di 1, 2, 3, 4, 5 e 10 alberi in parallelo. Tale albero è caratterizzato dai seguenti dati:

* altezza = 20
* split = 2
* n° attributi su nodo = 2
* n° attributi su arco = 2

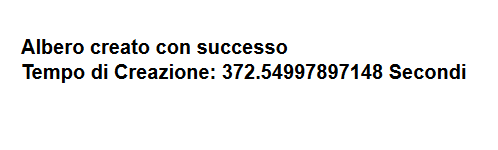
**NOTA:** Tutti i test riportati sono stati effettuati su un ASUS K550VX-DM108T con processore i7-6700HQ 2.6 GHz, 8 GB di RAM DDR3 con velocità 2333MHz, 1TB HDD 5600RPM. Il server è stato emulato con WampServer. Alla fine della sezione verrà trattato un argomento riguardante una soluzione architetturale per ottimizzare di molto tali tempi.



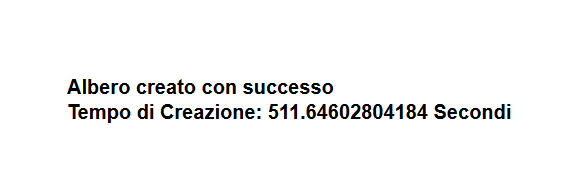
**TEST 1**: Creazione di 1 albero

/Users/Mario/Desktop/IMG-20170210-WA0046.jpg

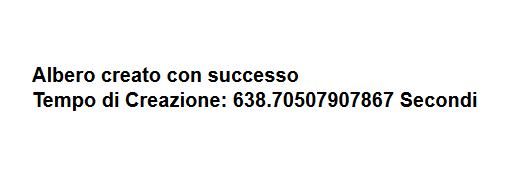
**TEST 2**: Creazione di 2 alberi simultaneamente



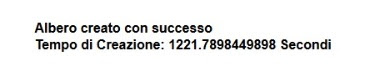
**TEST 3**: Creazione di 3 alberi simultaneamente



**TEST 4**: Creazione di 4 alberi simultaneamente



**TEST 5**: Creazione di 5 alberi simultaneamente



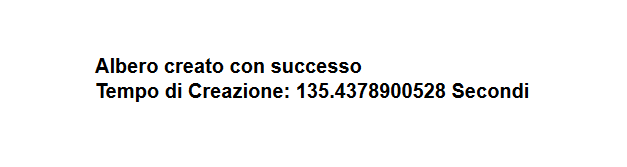
**TEST 6**: Creazione di 10 alberi simultaneamente

Sulla base dei test svolti, abbiamo calcolato una regressione lineare. Si può notare come la creazione di 10 alberi in parallelo sia perfettamente corrispondente alla funzione. Per poter stimare il tempo atteso per la creazione di 100 alberi in parallelo applichiamo la funzione, ottenendo:

Ora invece verranno riportati i test riguardanti la creazione di un albero ma al variare del numero di attributi su nodi e archi:

/Users/Mario/Desktop/IMG-20170210-WA0053.jpg

**TEST 8**: Creazione di 1 albero, n° attributi su nodi e archi = 1



**TEST 8**: Creazione di 1 albero, n° attributi su nodi e archi = 2

/Users/Mario/Desktop/IMG-20170210-WA0049.jpg

**TEST 9**: Creazione di 1 albero, n° attributi su nodi e archi = 3

/Users/Mario/Desktop/IMG-20170210-WA0050.jpg

**TEST 10**: Creazione di 1 albero, n° attributi su nodi e archi = 4

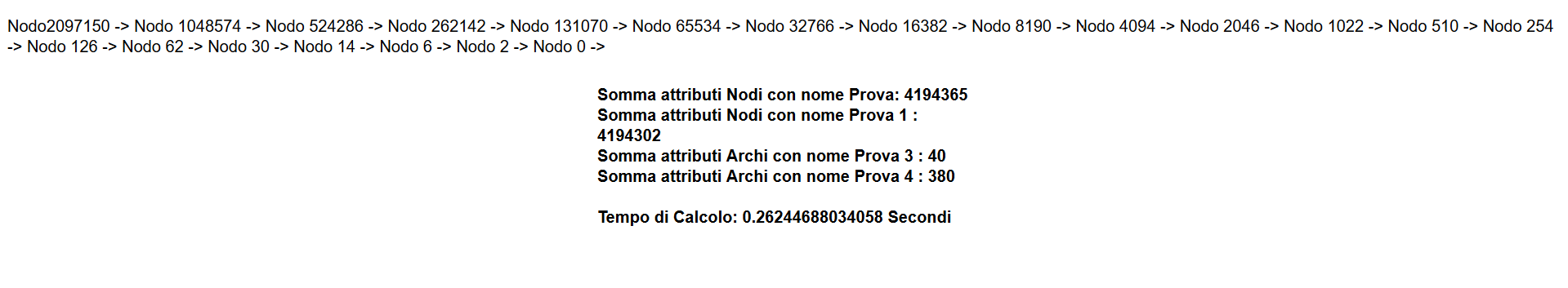
Anche questo caso, sulla base dei test svolti, abbiamo calcolato una regressione lineare per visualizzare l’andamento del tempo di creazione in base al numero di attributi su nodi e archi.

**CalculatePath.php**

Nel file calculatePath.php viene riportato il codice di calcolo della somma degli attributi dato un cammino tra due vertici. All'interno di tale file viene dichiarata la classe calculatePathOBJ contenente i seguenti metodi:

* **CalculatePath()** è il metodo principale da poter richiamare in quanto metodo pubblico ma non è il cuore dell'esecuzione del calcolo. In effetti il metodo controlla solo la connessione e l'esistenza dell'albero scelto dall'utente per poi chiamare il metodo Path().
* **Path()** è il metodo privato che esegue il vero e proprio calcolo del cammino. In primo luogo recupera il figlio scelto dall'utente; successivamente risale fino al padre tramite un ciclo nel quale recupera il PadreID di ogni nodo e lo setta come nodo successivo da visitare. Tutto ciò viene eseguito tramite SELECT consecutive sul database. Nel risalire, somma tutti gli attributi degli archi e dei nodi.

Con questo metodo riusciamo a rimanere molto al di sotto dei tempi prestabiliti. Il test è stato effettuato su un albero da 2.097.150 di nodi (Altezza = 20, Split = 2, n° Attributi Nodo = 2, n° Attributi Arco = 2). Il path è stato calcolato dal nodo 2.097.150 al nodo 0.



**Test 1**: Calcolo del path dal nodo 2.097.150 al nodo 0

**DeleteTree.php**

Nel file deleteTree.php viene riportato il codice di eliminazione dell'albero selezionato dall'utente. All'interno viene dichiarata la classe deleteTreeOBJ contenente il solo metodo:

* **deleteTree()** controlla inizialmente la connessione al DB e l'esistenza dell'albero da eliminare per poi settare il valore nella colonna “usabile” della tabella “trees” uguale a 0. Successivamente modifica il nome dell’albero da eliminare con il suo md5. Ciò permette ad un nuovo utente di creare un albero con il nome dell’albero eliminato (senza md5 la creazione fallirebbe poiché è presente un albero con quel nome, anche se già eliminato). Sarà poi il DB ad occuparsi giornalmente dell'eliminazione (tramite un evento alle 1:00 di notte come descritto in precedenza nelle altre sezioni).

Abbiamo ritenuto non necessario effettuare specifici test sull’eliminazione di più alberi in parallelo poiché tale evento avviene all’1:00 di notte. Il DB ha quindi 6 ore di tempo per eliminare tutti gli alberi; ciò, visti i tempi in fase di creazione, non crea preoccupazione su tale aspetto

**Db.php**

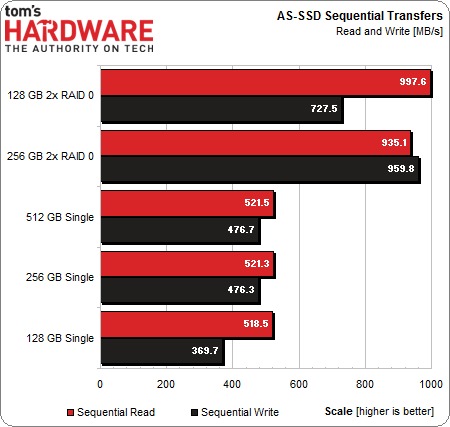
Nel seguente file viene riportato il codice utilizzato dagli altri file PHP per interfacciarsi con il DB. Al suo interno sono presenti 5 metodi:

* **connection()** è il metodo che si occupa della connessione al DB. Al suo interno viene richiamato il file config.ini nel quale sono memorizzati l'username, la password e il nome del DB. In caso di esecuzione andata a buon fine viene restituita la connessione con il DB, altrimenti viene restituito false.
* **query()** è il metodo che si occupa di effettuare la query sul database. Al suo interno viene richiamato il metodo connection() per effettuare la connessione con il DB per poi eseguire la query.
* **select()** è il metodo assegnato per l'esecuzione delle query di tipo SELECT. Al suo interno viene richiamato il metodo query(), il quale restituisce false nel caso di esecuzione non andata a buon fine, altrimenti un array doppio con tutti i risultati restituiti dalla query.
* **error()** è il metodo che controlla e restituisce gli eventuali errori forniti dal database.
* **quote()** è il metodo che inserisce tra virgolette le stringhe passategli ed elimina tutti i caratteri non accettati dal DB.

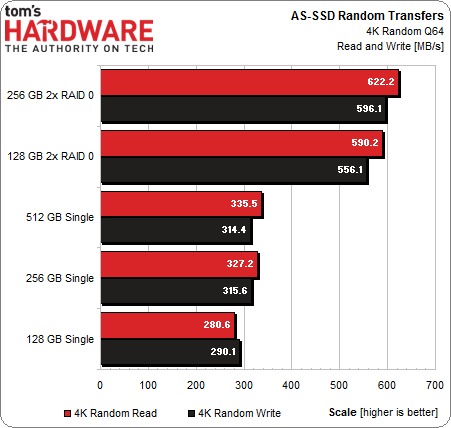
**Soluzione Architetturale v2**

Dopo esserci resi conto che i tempi di creazione di 100 alberi in modo concorrente superavano i nostri margini di tollerabilità, abbiamo pensato di adottare una soluzione alternativa a quella consentita dai nostri limiti fisici, ovvero il possesso di soli PC o portatili. Analizzando una ad una tutte le fasi che portano alla creazione di un albero, ci siamo accorti che la più lenta, ovvero il “collo di bottiglia”, è la strutturazione dei 3 file CSV all’interno del DB. Una prima soluzione quindi è stata quella di far girare il DB del nostro PPC su di un SSD, per velocizzare di molto le operazioni di scrittura e lettura. Ciò le rende sicuramente molto più veloci rispetto ad un HDD da 5.6k RPM (ciò che abbiamo in possesso), ma non ci da comunque la sicurezza di riuscire a rimanere sotto un’ora di tempo. Abbiamo quindi pensato di adottare l’utilizzo di SDD in RAID. Dopo aver effettuato diverse ricerche in rete, abbiamo notato come tale tecnologia consenta di ottimizzare in modo elevato un DB e rendere parallele le varie operazioni di scrittura all’interno di quest’ultimo. Ciò ci permette di abbassare drasticamente i tempi di creazione. In particolare ci siamo soffermati su due livelli di RAID che risultano essere i più utilizzati seppur in environment differenti: il RAID 0 e il RAID 10.

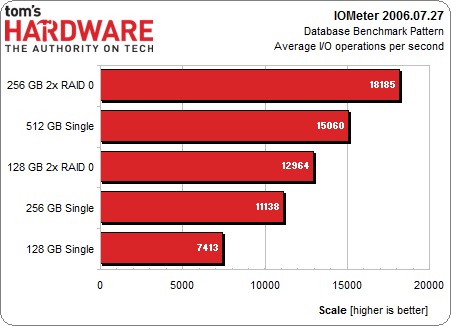
Il primo presenta un overhead praticamente nullo e performance davvero interessanti come si evince dai benchmark sottostanti:



**FIGURA 1**: Read e Write (Mb/s)



**FIGURA 2**: 4K Random Q64, Read e Write (Mb/s)



**FIGURA 3**: DB Benchmark Pattern, Average I/O Operation Per Second

Il RAID 0 ha però uno svantaggio potenzialmente pericoloso: se anche solo uno dei dischi del RAID ha un malfunzionamento, l’intero RAID diventa inutilizzabile.

Il secondo, al contrario, è largamente utilizzato nei database per la sua affidabilità poiché tollerante ai guasti, ma non raggiunge gli stessi livelli di performance di un RAID 0, anche se fornisce ottime performance in sistemi write intensive.

Seppure l’affidabilità ne risentirebbe, per abbattere i tempi prefissatici il team sarebbe propenso nell’utilizzo del RAID 0 che, data la facilità di applicazione e l’enorme vantaggio in termini di prestazioni sarebbe il più indicato per il nostro scopo.

Avendo effettuato i nostri test su un portatile con le caratteristiche sopra citate ci è stato fin da subito chiaro che passare da HDD a 5.7k RPM ad un SSD ci avrebbe consentito di avere tempi di creazione di 100 alberi in parallelo che sarebbero stati ben lontani dalle precedenti 3 ore e 20 (molto probabilmente al di sotto delle 3 ore) ma che sarebbero comunque insoddisfacenti per poter abbattere il muro di un’ora prefissatoci. La tecnologia sopra descritta, invece, ci permetterebbe di abbattere ulteriormente i tempi e di soddisfare quindi i nostri pre-requisiti impiegando meno di un'ora di tempo per la creazione di 100 alberi in parallelo. Il team ha quindi identificato come architettura ideale un server professionale che abbia almeno 4 SSD in RAID 0.