2015 - 2016

zioFrank Team

Andrea Giuliani	194352	p90.cc@hotmail.it
Gabriele Di Rocco	195037	squall_l@live.it
Daniele Evangelista	151125	vandaniel83@libero.it
Valerio Piccioni	223000	vesparo@gmail.com
Francesca Ricci	231669	fra.ticci71@gmail.com
Vincenzo Battisti	204509	vincenzo.battisti@gmail.com
Giovanni D'Agostino	202643	giova23@hotmail.it

[PPC – DELIVERABLE #1]

Analisi dei requisiti e prime assunzioni sul sistema da sviluppare per il committente.

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 not 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

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
Messa a fuoco delle componenti del Sistema e dei servizi implementati da ognuna di esse ad un alto livello di astrazione	07/12	09/12	Un ampio dibattito a 7 ci ha permesso di confrontare le differenti visioni del sistema e di darne una nuova rappresentazione ritenuta ottimale.
Individuare la collocazione fisica dei software che andremo a produrre per il sistema ed i vincoli architetturali richiesti dal costumer.	07/12	11/12	L'interrogazione diretta del committente circa questo tema si è rivelata soddisfacente alla comprensione del requisito. A seguire si sono prese delle decisioni in merito presenti nella sezione ASSUNZIONI.
Individuare laddove è necessario o meno dividere un sistema in sottosistemi differenti o accorpare due componenti in un'unica componente più grande.	03/12	In corso	Ad ogni passo che poniamo in avanti nella modellazione troviamo continue risposte a questa domanda. Stimiamo che scendendo ad un più basso livello di astrazione si chiarirà maggiormente l'utilità di accorpare/dividere due componenti.

A.1 - Requisiti Funzionali

In questa sessione vengono listati i requisiti funzionali necessari alla realizzazione del nostro sistema, provvedendo per ognuno ad evidenziarne il livello di priorità secondo una scala da 1 a 3, con 1 = ALTA PRIORITA' e 3 = BASSA PRIORITA'.

Il sistema PPC è un sottosistema autonomo facenteparte di un macrosistema Micron.
Esso deve permettere all'Utente Micron di generare alberi soddisfacenti determinate caratteristiche, chiamare funzioni che salvino i suddetti alberi in un DataBase ed accedere ai dati presenti sul DB per eseguire letture e calcoli sui suddetti alberi.

Per farlo abbiamo bisogno di individuare 3 sottosistemi di PPC:

A. Un interfaccia grafica (GUI)
B. Un motore che si occupi di eseguire le funzioni richieste (BL)
C. Una base di dati su cui salvare le informazioni che ci interessa mantenere (DB)

A. REQUISITI FUNZIONALI **GUI***(GRAPHIC USER INTERFACE):*

1. La GUI permette di generare un albero partendo da una lista di parametri passati in input e di salvarlo (in uno specifico formato di file) sotto una specifica directory*[p.1]*
2. La GUI permette di scatenare un sistema esterno che apra un file, ne estrapoli l'albero contenuto e lo inserisca opportunamente nel DB*[p.2]*
3. La GUI permette di scatenare un sistema esterno che preso in input un albero e due nodi A e B appartenenti ad esso, sia in grado di ritornare la somma degli attributi dei nodi compresi lungo il cammino tra A e B*[p.2]*

B. REQUISITI FUNZIONALI **BL***(BUSINESS LOGIC) :*

1. Le funzioni implementate nel motore BL devono essere suddivise tra funzioni scatenabili solo dal sistema stesso e funzioni accessibili anche da sistemi esterni che si assume appartengano comunque al macrosistema Micron. *[p.3]*
2. Il motore BL deve implementare una funzione capace di accedere alla directory in cui si trovano i file dell'albero, estrapola le informazioni ed esegue una INSERT nel Data Base *[p.1]*
3. Il motore BL deve implementare una funzione che presi in input due vertici A B, ne ricava l'albero di appartenenza , legge i dati dal data base e ritorna la lista di vertici da AB insieme alla somma di ogni attributo*[p.1]*

C. REQUISITI FUNZIONALI **DB***(DATABASE) :*

1. La struttura del DB deve supportare la struttura dati dell'albero salvato su file.*[p.1]*
2. Ogni tabella del DB deve prevedere un ID autoincrementato per ragioni aziendali. *[p.2]*

A.2 - Requisiti Non Funzionali

In questa sessione vengono listati i requisiti non funzionali necessari alla realizzazione del nostro sistema, provvedendo per ognuno ad evidenziarne il livello di priorità secondo una scala da 1 a 3, con 1 = ALTA PRIORITA' e 3 = BASSA PRIORITA'.

1. Sono richieste buone performance in termini di velocità da parte della funzione che calcola la somma degli attributi di un particolare cammino in un albero. (vediReq Funzionale n.3)*[p.1]*
2. E' necessario garantire una buona scalabilità del sistema che verosimilmente vedrà crescere, insieme alle dimensioni dei dati gestiti, anche il numero di accessi concorrenti alle risorse. *[p.1]*

A.3 - Contents

Di seguito esponiamo i contenuti scambiati dal nostro sistema con l'esterno, sia in ingresso che in uscita.
(IN) – I contenuti acquisiti all'interno del nostro sistema dall'esterno.
(OUT) – I contenuti offerti all'esterno dal nostro sistema.

1. Permettiamo l'accesso ai file salvati da sistemi esterni ***(OUT)***
2. Permettiamo l'accesso alle funzioni del BL da sistemi esterni ***(OUT)***
3. Sebbene sia parte del nostro lavoro di progettazione, modellare un DB consono al nostro sistema, siamo perfettamente consci del fatto che la realizzazione effettiva del nostro sistema avverrà ricavando i dati dal DB ufficiale di Micron. ***(IN)***

A.4 - Assunzioni

1. La corrente documentazione si riferisce ad una modellazione ad alto livello di astrazione pertanto prende in considerazione solo la casistica migliore, le fasi successive della progettazione provvederanno a soddisfare anche altre diramazioni.
2. Assumiamo che tutte le unità software del nostro sistema girano sulla stessa macchina, mentre i dati vengono memorizzati in un DB server proprietario di Micron.
3. Il software che interroga il DB per calcolare gli attributi di un ramo di un dato albero, dovrà preoccuparsi di verificare se esiste l'albero nel DB e se esiste il PATH richiesto. Decideremo in futuro se affidare questi calcoli al Software BL o al DB (quando sarà più chiara la complessità implementativa).
4. Il software che genera l'albero andrà a salvare il suddetto su un file garantendone una struttura dei dati conforme a quella delle tabelle del DB. Assumiamo che questo torni utile in due task.
    a. Quando un operatore Micron (ESTERNAMENTE AL NOSTRO SISTEMA) andrà a popolare il file contenente l'albero e lo troverà già strutturato.
    b. Quando il Software BL andrà a fare upload su DB e non dovrà spendere tempo di esecuzione in operazioni di ristrutturazione dello stesso.

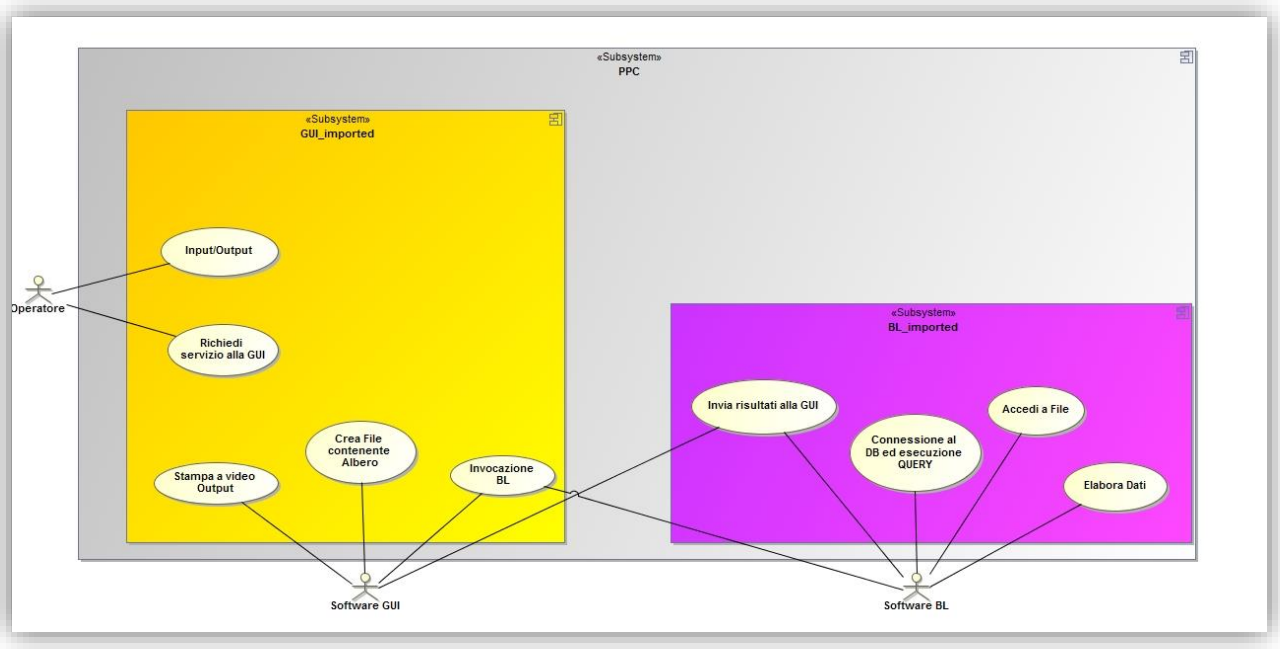
A.5 - Rischi

Allo stato attuale la progettazione esplora con poco dettaglio alcune porzioni di sistema. Le stesse rappresentano i passi più delicati da compiere nelle fasi successive della modellazione.

Tra le task che più delle altre potrebbero richiedere un RESTYLING del sistema abbiamo stimato:

1. Il passaggio dei parametri in input sulla GUI.
   Alcuni di questi parametri infatti potrebbero dover trovare un match nel DB e quindi imporci di implementare nuovi accessi ad esso.
2. La strutturazione del file e la scelta del formato con cui esso verrà salvato.
3. Le funzionalità che garantiranno accessi multipli e consistenti al sistema.
4. La corretta ripartizione delle operazioni di calcolo tra le componenti del sistema in maniera tale che si garantiscano performance alte in termini di velocità.

B.1 - Use Case Diagram



Come possiamo vedere nel diagramma overview mostrato, gli attori che interagiranno con il nostro sistema sono:

- **Software GUI:**
  L'attore passivo che implementa un interfaccia grafica e dei servizi offerti all'Operatore.
- **Operatore:**
  L'utente che utilizza l'interfaccia grafica per usufruire dei servizi offerti dalla GUI;
- **Software BL:**
  L'engine che effettua alcune delle operazioni richieste dalla GUI e può essere invocato dalla stessa o da sistemi esterni.

Ciò che ci si aspetta possa fare l'operatore è appunto interfacciarsi con le funzionalità di input/output offerte dalla GUI e scatenare in qualche modo le funzioni richieste al software della stessa.

Il Software GUI, procederà quindi a soddisfare le richieste o autonomamente o invocando l'engine BL.

Vediamo cosa è in grado di fare la GUI autonomamente:

- Creare un file contente un nuovo albero
- Stampare a video l'output delle operazioni svolte
- Invocare l'engine esterno, passandogli opportunamente dei parametri

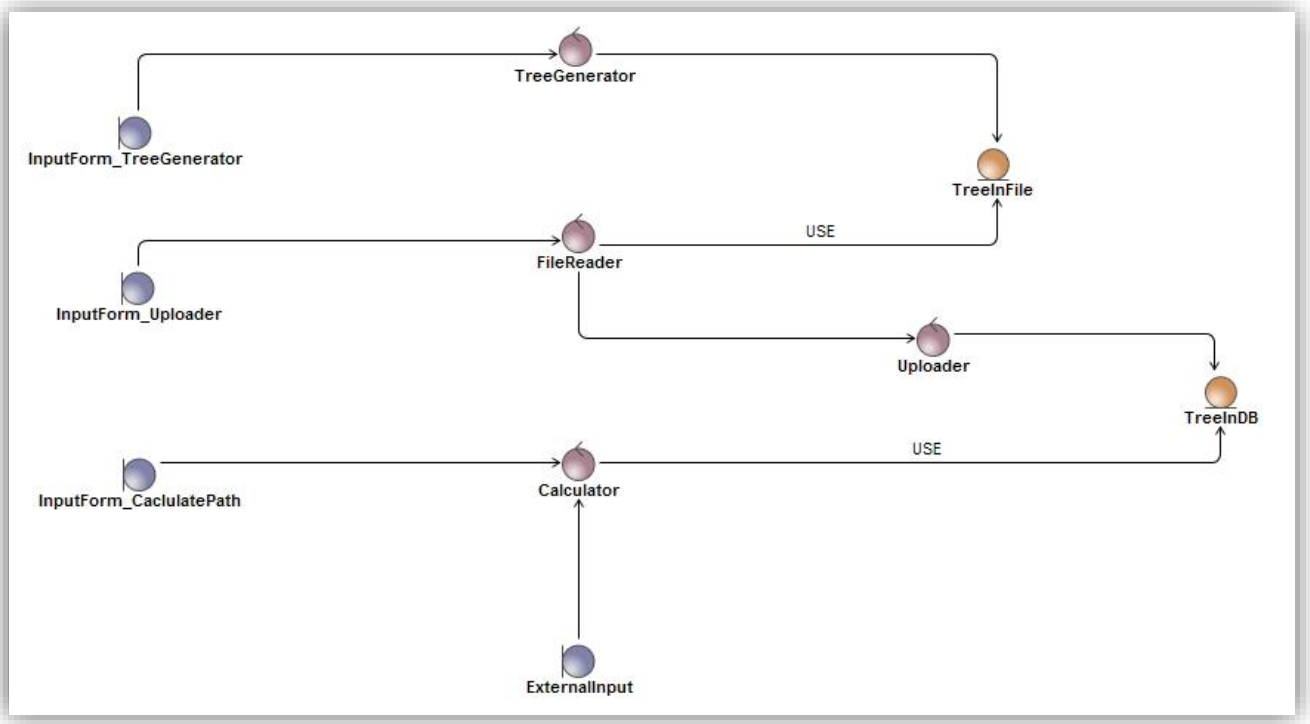
Il Software BL invece svolge le seguenti funzioni:

- Accesso ad un file salvato in una directory visibile dal sistema

- Connessione al DB ed esecuzione di opportune QWERY di lettura e modifica delle informazioni
- Elaborazione dei dati
- Restituzione di un Feedback o di un risultato al Software chiamante

Come è dunque visibile il sistema può essere scomposto in ALMENO due sottosistemi più piccoli in grado di comunicare tra loro laddove necessario.

B.2 – Analisys Obj Model



Una volta analizzato lo use case iniziamo ad individuare le principali entità che comporranno il sistema.

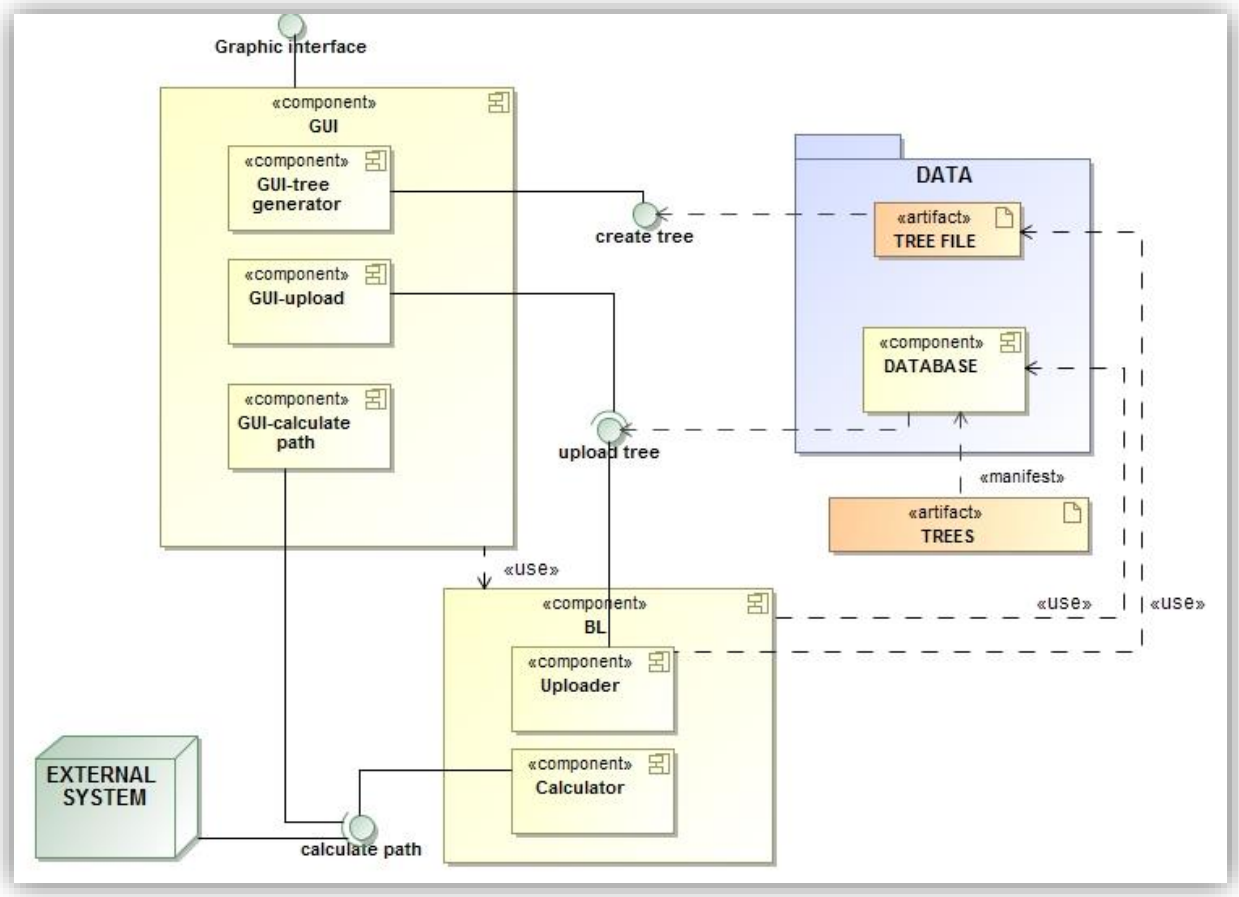
L'operatore colloquia con l'interfaccia **InputForm_TreeGenerator,** la quale richiama il controller **TreeGenerator,** preposto appunto alla funzione di generazione di un nuovo albero.

Il risultato della computazione dei dati genera una entità **TreeInFile,** rappresentante le informazioni relative all'albero, memorizzate su file.

L'operatore ha altresì accesso ad una **InputForm_Uploader,** la quale richiede il servizio di lettura su file tramite il controllore **FileReader,** e l'inserimento su Database tramite il controllore **Uploader,** generando le tuple costituenti l'albero.

Per concludere, si può accedere alla funzionalità di calcolo dei dati di interesse, sia tramite l'interfaccia interna **InputForm_CalculatePath,** sia da interfaccia esterna al nostro sistema, entrambe in grado di richiedere un servizio al controllore **Calculator**. Quest'ultimo svolgerà le sue mansioni utilizzando i dati memorizzati precedentemente su database.

C.1 – Component Diagram



Il nostro sistema PPC può essere descritto individuando delle macrocomponenti:

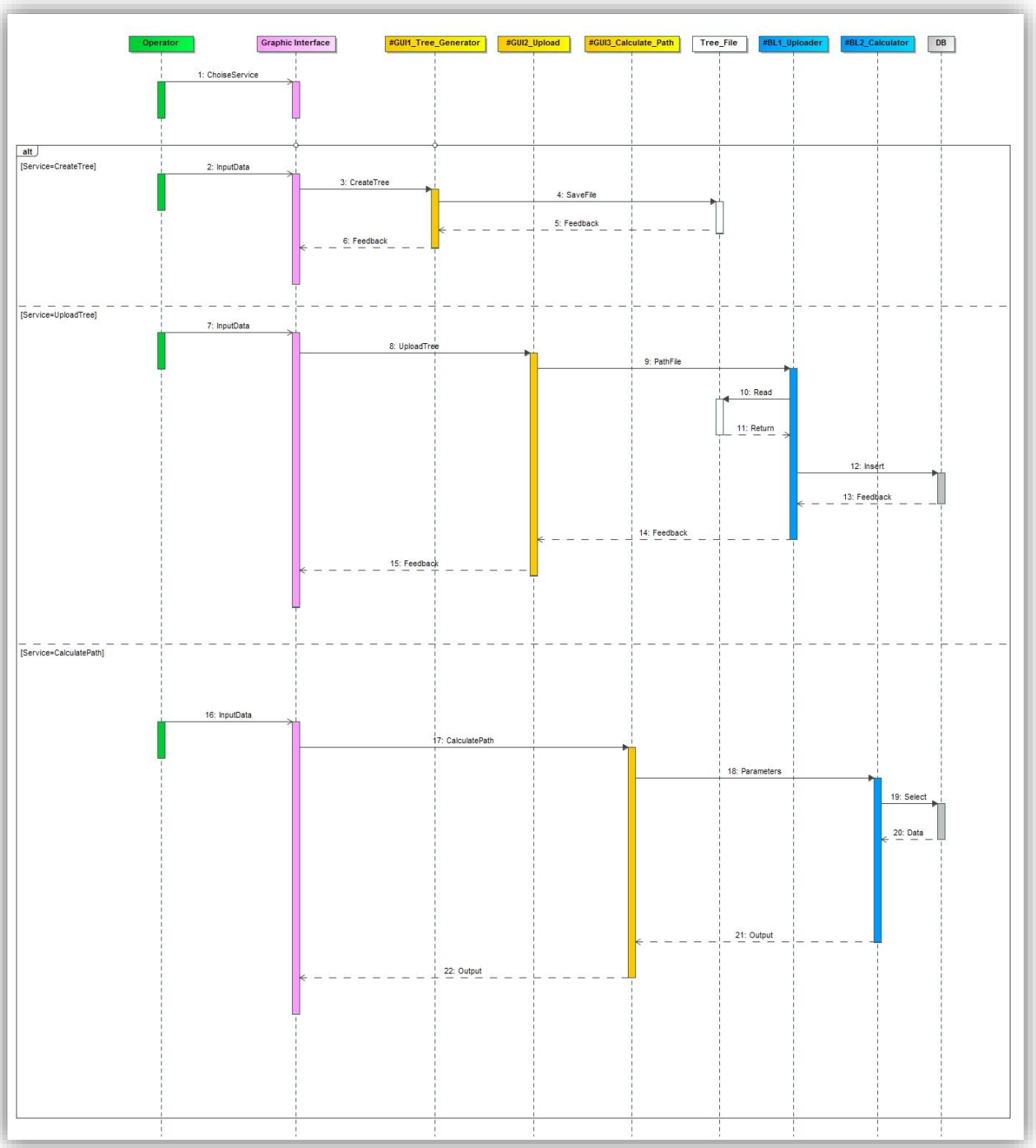
1. **GUI:**
   Si tratta dell'unica componente Sofware che offre il servizio di gestione diretta delle interazioni con l'utente del sistema (input/output).
   Essa verrà impiegata in 3 differenti operazioni distinte tra loro, pertanto possiamo individuare al suo interno 3 sottocomponenti: GUI-tree generator, GUI-upload , GUI-calculate path.
2. **DATA:**
   Si tratta dello spazio astratto di archiviazione delle informazioni, esso comprende al suo interno la componente Database e la porzione di memoria dove viene salvato il file dalla GUI.
3. **BL:**
   Si tratta della componente Software che viene invocata per eseguire delle operazioni di lettura/scrittura su DB e di elaborazione dei dati. Essa verrà impiegata per 2 differenti operazioni distinte tra loro, pertanto possiamo individuare al suo interno 2 sottocomponenti: BL-Uploader, BL-Calculator.

Diamo una spiegazione delle sottocomponenti prima citate:

1. **GUI-tree generator:**
   E' la sottocomponente della GUI che si occupa di offrire il servizio di generazione di un nuovo albero (create tree) e salvataggio su file dello stesso.
2. **GUI-upload:**
   E' la sottocomponente della GUI che si occupa di usufruire del servizio offerto da BL-Uploader tramite opportuno passaggio di parametri.
3. **GUI-calculate path:**
   E' la sottocomponente della GUI che si occupa di usufruire del servizio offerto da BL-Calculator tramite opportuno passaggio di parametri.
4. **BL-Uploader:**
   E' la sottocomponente della BL che si occupa di offrire il servizio di lettura del file e di inserimento delle informazioni estrapolate all'interno del DB.
5. **BL-Calculator:**
   E' la sottocomponente della BL che si occupa di offrire il servizio di lettura delle informazioni salvate su DB e di calcolo delle stesse.
   Questo servizio è usufruibile anche da sistema esterno.

C.2 – Sequence Diagram

Di seguito mostriamo il Sequence Diagram ricavato dal Component Diagram che mette in evidenza la dinamicità del sistema analizzandone uno specifico caso felice (Happy Path) in cui supponiamo che il flusso di ogni operazione possibile sul nostro sistema segua un esecuzione corretta e priva di anomalie.



Le LifeLine coinvolte nel nostro sistema sono le seguenti:

- Operator
- Graphic Interface
- #GUI1_Tree_Generator
- #GUI2_Upload
- #GUI3_Calculate_Path
- Tree_File
- #BL1_Uploader
- #BL2_Calculator
- DB

Operator: Si tratta dell'Utente Micron che andrà ad usufruire del sistema, come si evince egli si interfaccia solo ed esclusivamente con una **Graphic Interface** che permette l'input/output del sistema.

Tali interazioni attiveranno l'esecuzione di una parte specifica di software della **GUI** adibita alla risoluzione di un servizio.

Tali software potranno interagire con altre componenti quali, parti del motore esterno **BL,** uno spazio di storage fisico per il salvataggio dei file (**Tree_File**) ed una parte di storage strutturata in un **DB**.

Vediamo l'esecuzione dell'Happy Path.

L'operatore è chiamato dapprima a selezionare il servizio della GUI di cui vuole usufruire interagendo con l'interfaccia grafica(1).

A questo punto il flusso si dirama in 3 sottoflussi alternativi, ognuno adibito a soddisfare servizi differenti:

1. Servizio di Generazione di un file contenente l'albero (CreateTree)
2. Servizio di Upload di un file già esistente sul DB (UploadTree)
3. Servizio che permette il calcolo sugli attributi nel percorso tra due nodi di uno specifico albero (CalculatePath)

ALT. [Service=CreateTree]:

> L'operatore compila i dati in input tramite la Graphic Interface e li invia alla parte della GUI adibita a fornire all'utente il servizio di Creazione di un nuovo Albero (2)(3).
> Quest'ultima elabora i dati in modo da generare un albero e li salva in maniera strutturata su un file (4).
> Il Feedback positivo, restituito dal salvataggio del file, viene ritornato dapprima alla GUI e poi mostrato a schermo (5)(6).

ALT. [Service=UploadTree]

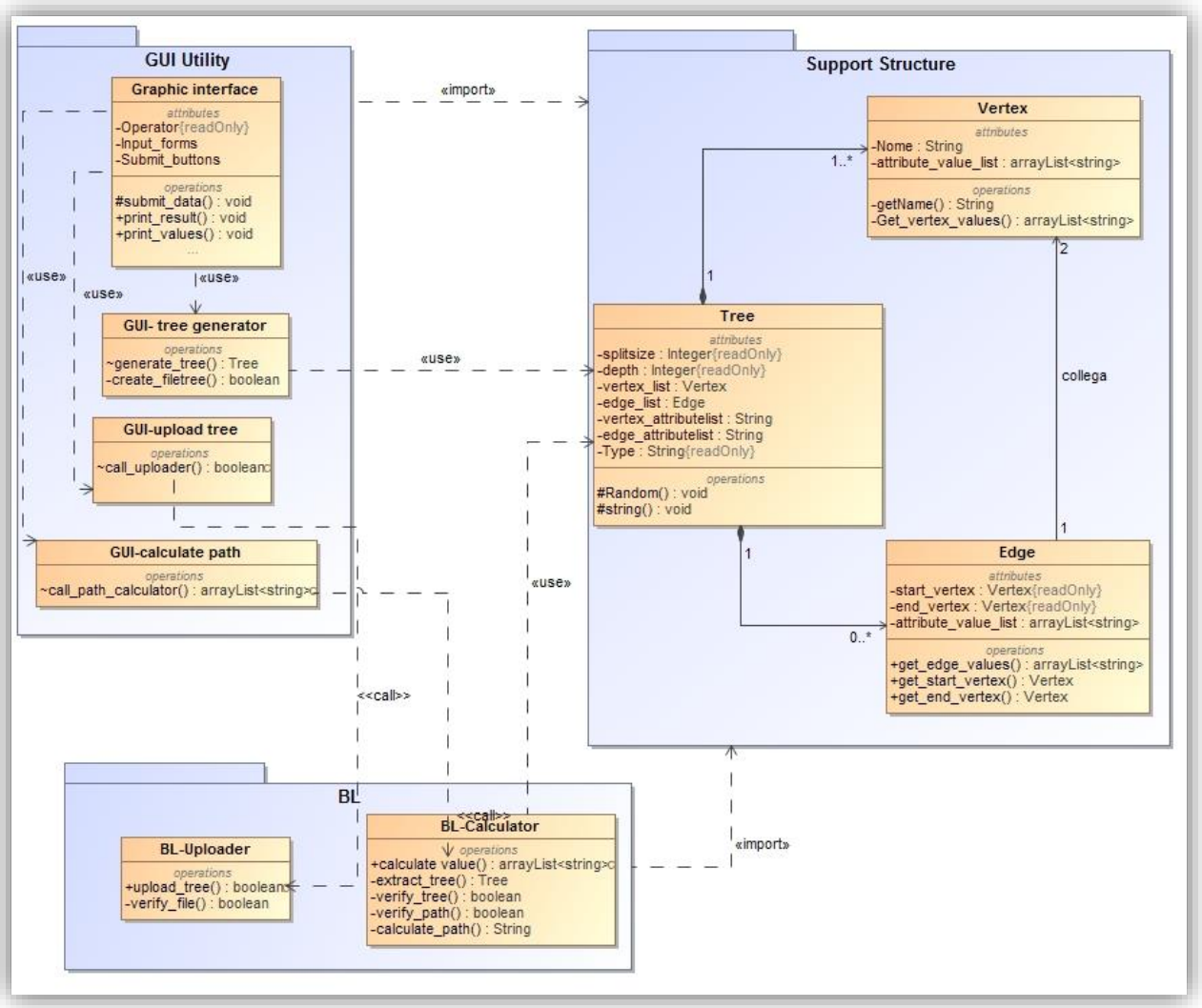
> L'operatore compila i dati in input tramite la Graphic Interface e li invia alla parte della GUI adibita a fornire all'utente il servizio di Upload del file Albero nel database (7)(8).
> Quest'ultima invoca un motore esterno #BL1_Uploader inviandogli i riferimenti del file da inserire su DB (9).

Il motore BL accede al file le cui cordinate erano contenute nell'input, acquisisce da esso i dati strutturati dell'albero e li inserisce nel DB tramite una Query di INSERT (10)(11)(12).
Ritornato il feedback positivo dal DB, esso viene propagato per tutte le parti coinvolte fino ad essere riportato alla Graphic Interface che lo mostra a schermo (13)(14)(15).

ALT. [Service=CalculatePath]

L'operatore compila i dati in input tramite la Graphic Interface e li invia alla parte della GUI adibita al servizio che calcola la somma degli attributi su un ramo specifico di un dato albero(16)(17).
Quest'ultima invoca un motore esterno #BL2_ Calculator inviandogli le informazioni utili ad identificare l'albero ed il relativo ramo su cui voler eseguire il calcolo degli attributi(18).
Il motore estrae l'albero da DataBase, esegue su di esso i relativi calcoli e li ritorna in output alla GUI che a sua volta li propaga a schermo sulla Graphic Interface (19)(20)(21)(22).

C.3 – Class Diagram



PREMESSA:

Assumiamo che alcune classi che all'apparenza sembrano superflue potrebbero in seguito a successivi raffinamenti, incrementare di complessità e grandezza. Riteniamo dunque per una migliore esposizione e futura analisi tenere classi che implementano pochi metodi.

DESCRIZIONE:

Questa prima versione del sistema consiste in tre package principali, ognuno dei quali assolve specifiche funzioni.
Il package GUI Utility conterrà tutte le classi necessarie alla realizzazione di un'interfaccia e i relativi servizi: tree generator, upload tree e calculatepath.
L'interfaccia grafica dovrà mostrare a video le schermate con le relative form e bottoni, necessari per specificare un servizio.
Le classi che realizzano l'interfaccia faranno uso di altre classi per l'implementazione dei servizi.

Più nel particolare:

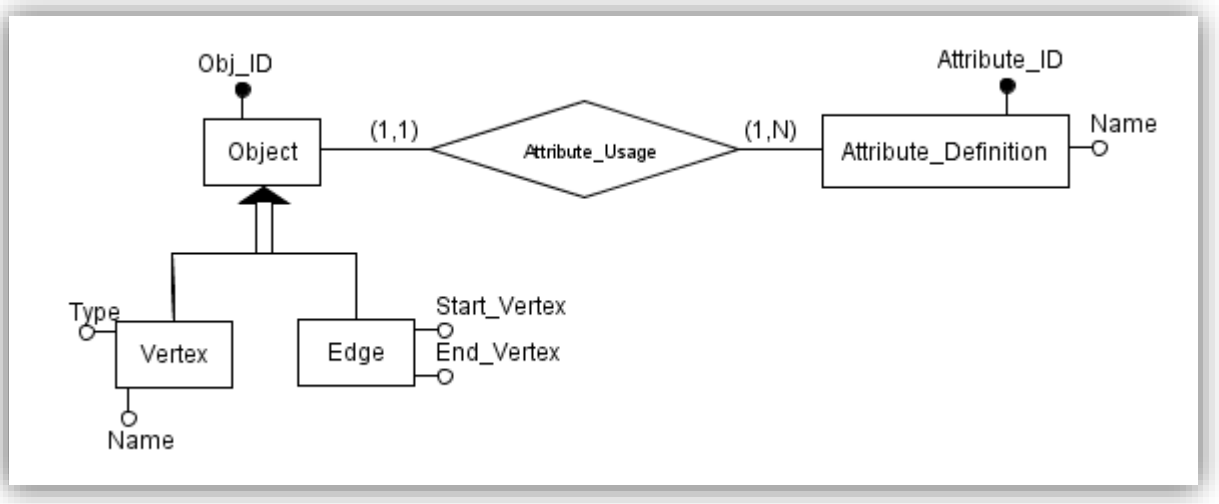
1. **Tree Generator**
   Utilizza metodi che istanziano un oggetto della classe tree che servirà come struttura di supporto e contenitore per i dati, che verranno poi salvati in maniera sistematica all'interno di uno specifico file
2. **Upload Tree**
   Richiama un metodo all'interno della classe BL-Uploader del package BL
3. **CalculatePath**
   Richiama un metodo all'interno della classe BL-Calculator del package BL

In particolare inoltre l interfaccia grafica utilizzerà classi/metodi per stampare a video feedback e risultati ottenuti dal package BL. Il package BL contiene al momento due classi che implementano le due componenti dell'engine.

1. **BL-Uploader**
   Questa classe si occuperà della verifica dell'idoneità del file ed effettuerà la insert dei dati formattati nel file sul database.
2. **BL-Calculator**
   Questa classe dopo aver verificato l'idoneità dell'albero e del path richiesto tramite GUI, accederà al database per estrarre il sottoalbero richiesto ed effettuerà le opportune operazioni di calcolo utilizzando la struttura di appoggio Tree.

Il package SupportStructure conterrà le classi che descriveranno la struttura di supporto tree. Tali classi dovranno permettere di creare oggetti che siano conformi alle specifiche dell'albero richiesto. Ovvero tipo, splitsize, depth, vertexattribute list, edgeattribute list etc. e le funzioni per la generazione di contenuti random.

E.1 – DataBase – ER diagram



PREMESSA:

Il DataBase è ad una sua primissima versione. Tale modello ER serve a darne una prima traccia che verrà poi raffinata e ristrutturata nelle fasi immediatamente successive.

DESCRIZIONE:

La prima entità che è importante descrivere è l'entità OBJECT.
Con essa andiamo ad identificare tutti gli elementi di un albero, quindi Vertici ed Archi.

Ogni object può essere identificato grazie ad un ID numerico autoincrementale.

Tra gli oggetti dell'albero distingueremo due diverse tipologie di oggetto che andremo a collocare nelle due entità distinte: VERTEX ed EDGE.

La prima, riferita ai vertici del grafo che andremo a generare, comprenderà al suo interno un attributo Type per identificare a QUALE albero appartenga un vertice ed un attributo name che aiuterà il vertice ad essere più riconoscibile.

Al contrario la seconda entità figlia di OBJECT, l'entità EDGE, mostra come attributi Start_Vertex ed End_Vertex, che ne indicano, come intuibile, a quali vertici si lega tale arco (sappiamo per definizione che un arco per esistere deve essere legato a due vertici).

Gli elementi dell'entità Object si legano a degli Attributi (questo perché sappiamo che un vertice o un arco possono avere uno o più attributi entro i quali inserire dei valori), tali Attributi saranno memorizzati in un apposita tabella definita dall'entità Attribute_Definition.In quest'entità teniamo traccia anche dell'identificativo numerico di ogni attributo e di un nome che ne faciliti il riconoscimento.

La relazione che lega un oggetto dell'albero ad ogni attributo è detta Attribute_Usage.