2015 - 2016

zioFrank

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 #2]

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

Project Guideline

[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

Index - Content of this Deliverable

In questa session andiamo a mettere l'indice dei contenuti di questo DELIVERABLE.

List of Challenging Tasks Solved

ID	Challenging Task	Date (the Task is identified)	Date (the Challenging is solved)	Know How
S1	Individuare laddove è necessario o meno dividere un sistema in sottosistemi differenti o accorpare due componenti in un'unica componente più grande.	03/12/15	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.
S2	Messa a fuoco delle componenti del Sistema e dei servizi implementati da ognuna di esse ad un alto livello di astrazione	07/12/15	09/12/15	Un ampio dibattito a 7 ci ha permesso di confrontare le differenti visioni del sistema e di darne una nuova rappresentazione ritenuta ottimale.
S3	Individuare la collocazione fisica dei software che andremo a produrre per il sistema ed i vincoli architetturali richiesti dal costumer.	07/12/15	11/12/15	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.
S4	Acquisire cognizione di quale sarà verosimilmente l'ordine di grandezza degli alberi, degli attributi e delle operazioni che Micron si aspetta di poter gestire nel nostro sistema per evitare di proporre un prodotto sottodimensionato rispetto alle esigenze concrete.	21/12/15	23/12/15	Richiesta diretta al committente.
S5	Allineare i differenti punti di vista del Team per evitare discussioni future su decisioni già fissate.	13/12/15	21/12/15	Una serie di discussioni di gruppo, schemi alla lavagna ed interazioni dirette con il committente.

ID	Challenging Task	Date (the Task is identified)	Date (the Challenging is solved)	Know How
S6	Sottomettere le scelte di Design al committente il prima possibile per avere maggior tempo per dedicarci ai rischi che ci preoccupano maggiormente.	13/12/15	In corso…	Anticipare la consegna del Deliverable 2.

List of Challenging Task Unsolved and related Risk

ID	Challenging Task	Date (the Task is identified)	Priority Level	Side Effects	Know How
U1	Arrivare ad un embrione della fase implementativa nella prima metà di Gennaio.	22/12/15	High	Fallire questa Challenge significa ritrovarsi ad affrontare la fase più intensa del lavoro in un periodo di sovraccarico per il Team coinvolto da altri impegni accademici.	Spendere delle sessioni di learning, distribuite nel team, sulle tecnologie da utilizzare.
U2	Operare scelte di Design valide per garantire la scalabilità ed il multiaccesso.	22/12/15	High	Ritardare la risoluzione di questa Challenge significa correre il rischio di incappare in importanti Restyle del sistema in una fase troppo avanzata in cui il costo in termini di tempo sarebbe insostenibile.	Saper leggere, nel processo di risoluzione della Task “U1”, le tecnologie in chiave di Scalabilità e Multiaccesso.

A.1 – Requisiti Funzionali

In questa sessione vengono listati i requisiti funzionali necessari alla realizzazione del nostro sistema, provvedendo per ognuno a devidenziarne 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 facente parte 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 scatenare un sistema esterno che 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 auto incrementato 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.
   Divideremo la complessità di questa operazione in due parti, la prima, eseguita direttamente dal DB che verificherà se l'albero con il suo sottoramo esiste davvero e la seconda, eseguita dal BL, che opererà i calcoli opportuni sul sottoramo estratto.
   [Decideremo in futuro se confermare o smentire questa scelta che ripartisce i calcoli tra Software BL e 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 tre task.
5. Quando un operatore Micron (ESTERNAMENTE AL NOSTRO SISTEMA) andrà a popolare il file contenente l'albero lo troverà già strutturato.
6. Quando il Software BL andrà a fare upload su DB e non dovrà spendere tempo di esecuzione in operazioni di ristrutturazione dello stesso.
7. Quando il Software BL verrà invocato per calcolare un percorso su un ramo, sarà necessario verificare l'esistenza dell'albero su DB e del sottoramo su cui voler eseguire i calcoli.
8. Il Software GUI provvede solo all'interfaccia Grafica da offrire all'utente. Tutta la parte operazionale è affidata ad un motore esterno che verrà invocato all'occorrenza.
9. Il Sistema offre 3 servizi: Create Tree, Upload Tree e Calculate Path. Di questi 3 il primo sarà l'unico invocabile esclusivamente dalla GUI. Gli altri 2 saranno all'occorrenza sfruttabili anche da sistemi esterni.
   Non sarà tuttavia mansione del nostro Team procedere all'implementazione di questi sistemi esterni.
10. Ancora non è noto se il Sistema deve preoccuparsi di eliminare il file successivamente all'upload o preferiamo che resti salvato nella directory. Prenderemo questa decisione solo dopo che ci saremo resi conto di come sarà effettivamente strutturato il file internamente. Ci piacerebbe che i file restassero sulla directory anche dopo l'upoload, ma prima di permettere questo dobbiamo essere certi che a livello implementativo non si possa incappare in un doppio inserimento dello stesso albero su DB.

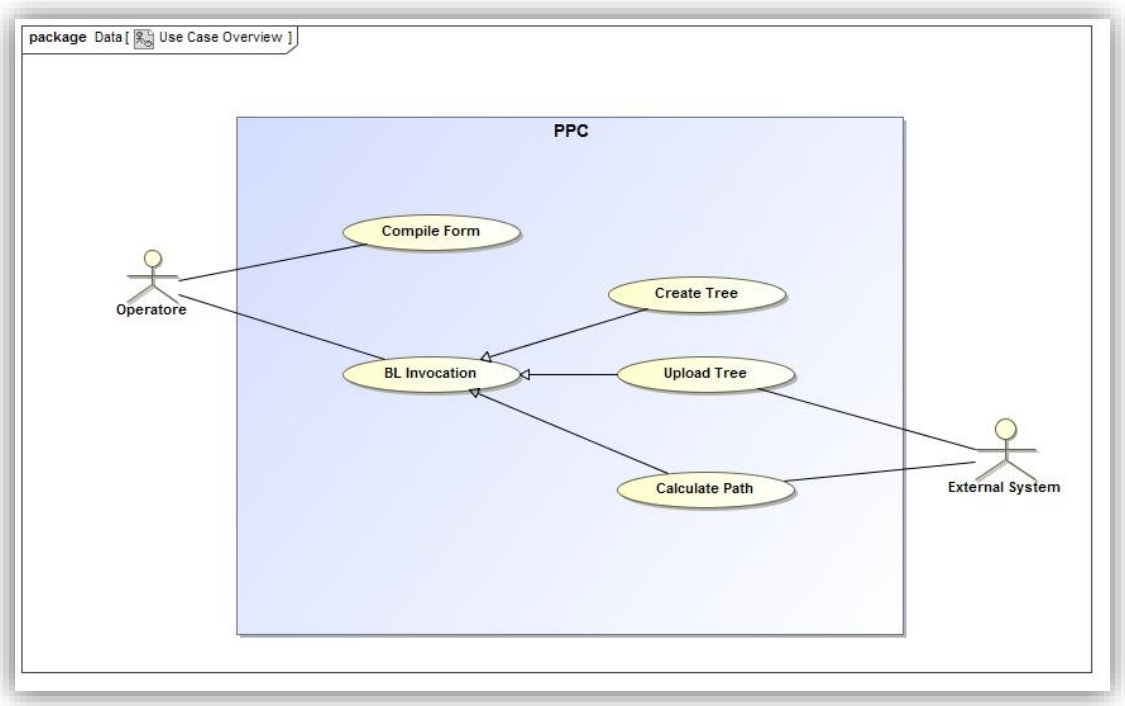
A.5 – Rischi

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. La GUI permetterà, tra i parametri da inserire in input, di selezionare attributi che dovranno fare match con i valori presenti nel DB. Questo prevede un interazione tra GUI e DB che ancora non è stata concretamente argomentata dal Team ma che potrebbe nascondere un rischio insidioso vista la natura multiutente che prevede accessi concorrenti e quindi modifiche al DB potenzialmente in conflitto.
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 notare, il diagramma mostrato [Use Case Overview] è una versione estremamente scarna e rudimentale di quello che sarà il nostro sistema.

La scelta di una visione così ad alto livello risiede in un semplice quanto efficace ragionamento.

A questo punto della modellazione il Team poteva scegliere tra due strade da seguire:

- Uno Use Case ESTREMAMENTE Overview del sistema che si limitasse a mostrare le parti in gioco e le interazioni tra esse
- Uno Use Case ESTREMAMENTE dettagliato che andasse ad analizzare tutti i casi d'uso possibili, anche i più inusuali e che risultasse quindi utile ad un analisi capillare del sistema in tutta la sua completezza.

Il Team ha preferito optare per la prima soluzione in quanto la forma mentis dei membri è stata educata a non abusare degli strumenti di raffinamento del diagramma Use Case e quindi avventurarsi ora in un diagramma estremamente a basso livello di astrazione avrebbe comportato un aumento considerevole dei rischi e soprattutto una spesa in termini di tempo che il Team non potrebbe permettersi.

Gli attori mostrati nel diagramma sono quindi soltanto due:

- **Operatore:** L'utente che utilizza l'interfaccia grafica per usufruire dei servizi offerti dalla GUI
- **External System:** Qualsiasi utente/ sistema automatizzato che intenda richiedere al nostro sistema l'esecuzione di una funzione.

Come si può evincere quindi l'Operatore ha accesso ad un interfaccia grafica che permette di **compilare delle Form in input** ed **invocare un BL** che adempia alle funzioni.

La generalizzazione ci mostra come esistono 3 diversi tipi di Invocazioni (“A kind of”) a BL.

1. Creazione di un albero
2. Upload di un albero
3. Calcolo di un percorso

Ognuna di queste richiama uno dei servizi offerti dal Sistema già precedentemente dichiarati nel capitolo A.1 Requisiti Funzionali (Requisiti GUI 1-2-3).

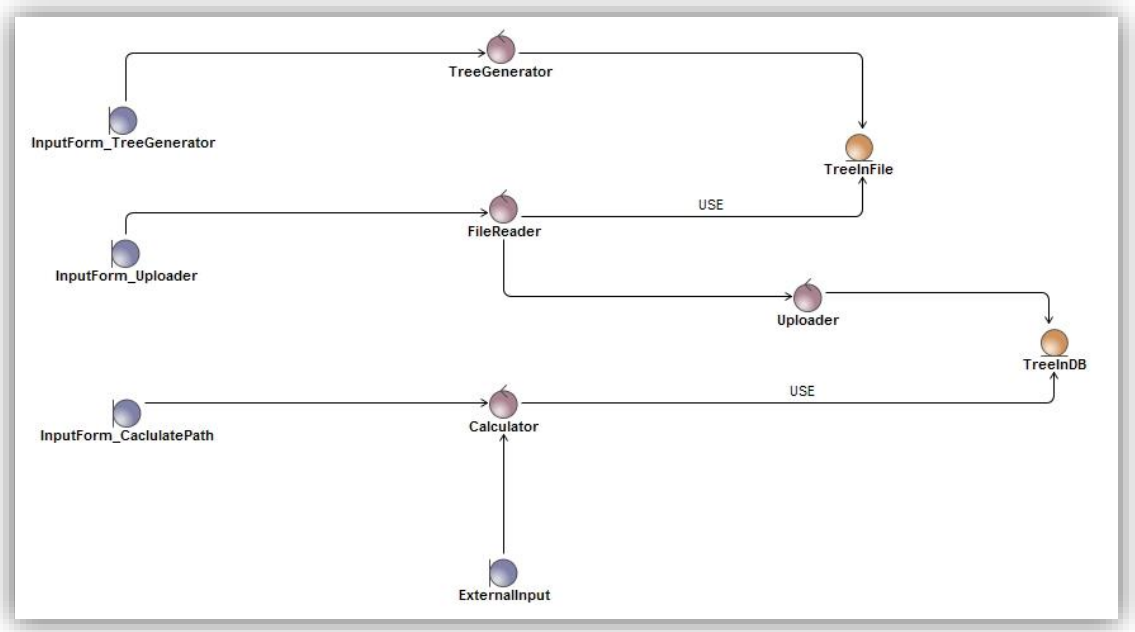
Approfondiamo i casi d’uso più importanti con informazioni in forma tabellare:

Name	Create Tree
Actors	Operatore
Pre Condition	Nessuna
Post Condition	Un nuovo file contentente un Albero è stato creato
ToDo	Crea un nuovo file contente un Albero
Note	La Directory dove viene salvato l’albero deve essere accessibili dall’esterno per favorire la modifica del file

Name	UploadTree
Actors	Operatore, Sistema Esterno
Pre Condition	Esiste il file nella directory ed è stato opportunamente modificato per essere uppato su DB
Post Condition	L’albero è stato caricato su DB
ToDo	Carica un albero da un file a DB
Note	Si assume che un sistema esterno si sia preoccupato di accedere al file e modificarlo prima che esso venga caricato. In seguito verranno esposte le features che soddisferanno questo constraint. Tale funzionalità, come mostrato, può essere invocata anche dall’esterno

Name	Calculate Path
Actors	Operatore, Sistema Esterno
Pre Condition	Esiste l’albero nel DB ed esiste il percorso ricercato nel suddetto albero
Post Condition	Nessuna
ToDo	Dato un albero già esistente, ne individua un ramo e su tale ramo esegue un calcolo matematico sugli attributi.
Note	Tale funzionalità, come mostrato, può essere invocata anche dall’esterno

B.2 – Analisys Obj Model



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

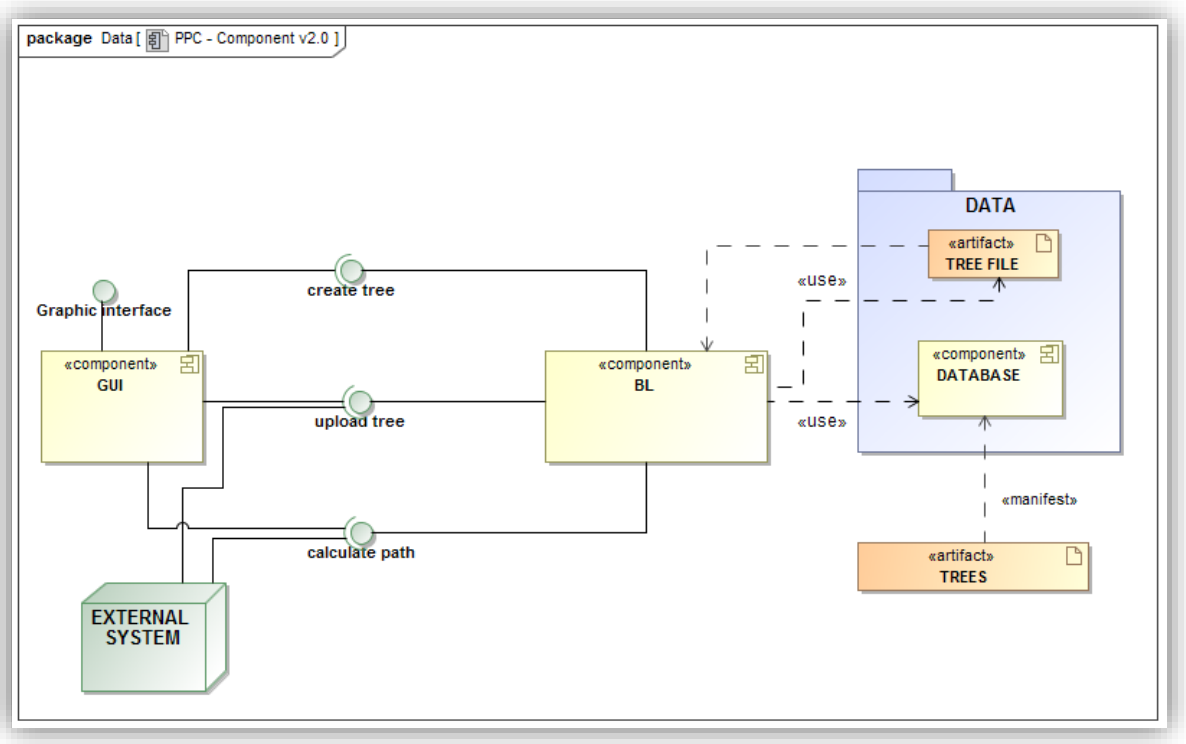
L'operatore colloquia con l'interfaccia **InputForm_TreeGenerator**, la quale richiama il controller **TreeGenerator**, preposto appunto alla funzione di generazione di un nuovo albero.

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



Alla luce delle scelte di Design evidenziate nel capitolo F di questo documento il component Diagram del nostro sistema risulta considerevolmente ritoccato rispetto alla precedente versione.
Anche in questo caso, come per lo Use Case Diagram, il Team si è diretto verso un percorso di reverse engeneering che riportasse la visuale ad un più alto livello di astrazione.

Questa scelta si è rivelata determinante in quanto ci ha aiutato a mettere ordine nella nostra visione delle componenti del sistema in maniera meno dettagliata e più concettuale; da qui nasce la decisione di rendere la GUI una semplice interfaccia grafica e non un unità operativa (vedi Cap.F scelta di design n.1).
Notiamo infatti come l'unica interfaccia implementata dalla componente GUI sia l'interfaccia grafica, mentre si adopera per utilizzare le tre interfacce offerte dalla componente BL.

Abbiamo inoltre operato in questa sede la decisione di non far affidamento ad un architettura client server e di far girare quindi tutte le 3 componenti del sistema in un'unica macchina (vedi Cap F scelta di design n.2)

Il nostro sistema PPC può essere quindi descritto individuando delle macro componenti:

1. **GUI:**
   Si tratta dell'unica componente Sofware che offre il servizio di gestione diretta delle interazioni con l'utente del sistema (input/output).
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.
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 è quella che offre i servizi di creazione

albero, upload dell'albero e calcolo del percorso.
Tali servizi come possiamo vedere vengono utilizzati dalla GUI e da eventuali sistemi esterni. E' facilmente deducibile come questa componente abbia delle dipendenze con gli elementi di DATA quali file e Database.

Esponiamo adesso in forma tabellare qualche informazione in più sui servizi offerti dalle componenti:

Nome	**Graphic Interface**
Accessibilità	Chiunque può avere accesso all'interfaccia grafica
Parametri	Nessuno
Vincoli	Devono essere riempiti tutti i campi di input. Prevediamo di imporre un limite (non ancora deciso) ad alcuni attributi per garantire un margine di efficienza nel nostro sistema.
Funzione	E' il servizio che offre una grafica che permetta l'inserimento in input dei dati e la visualizzazione degli output

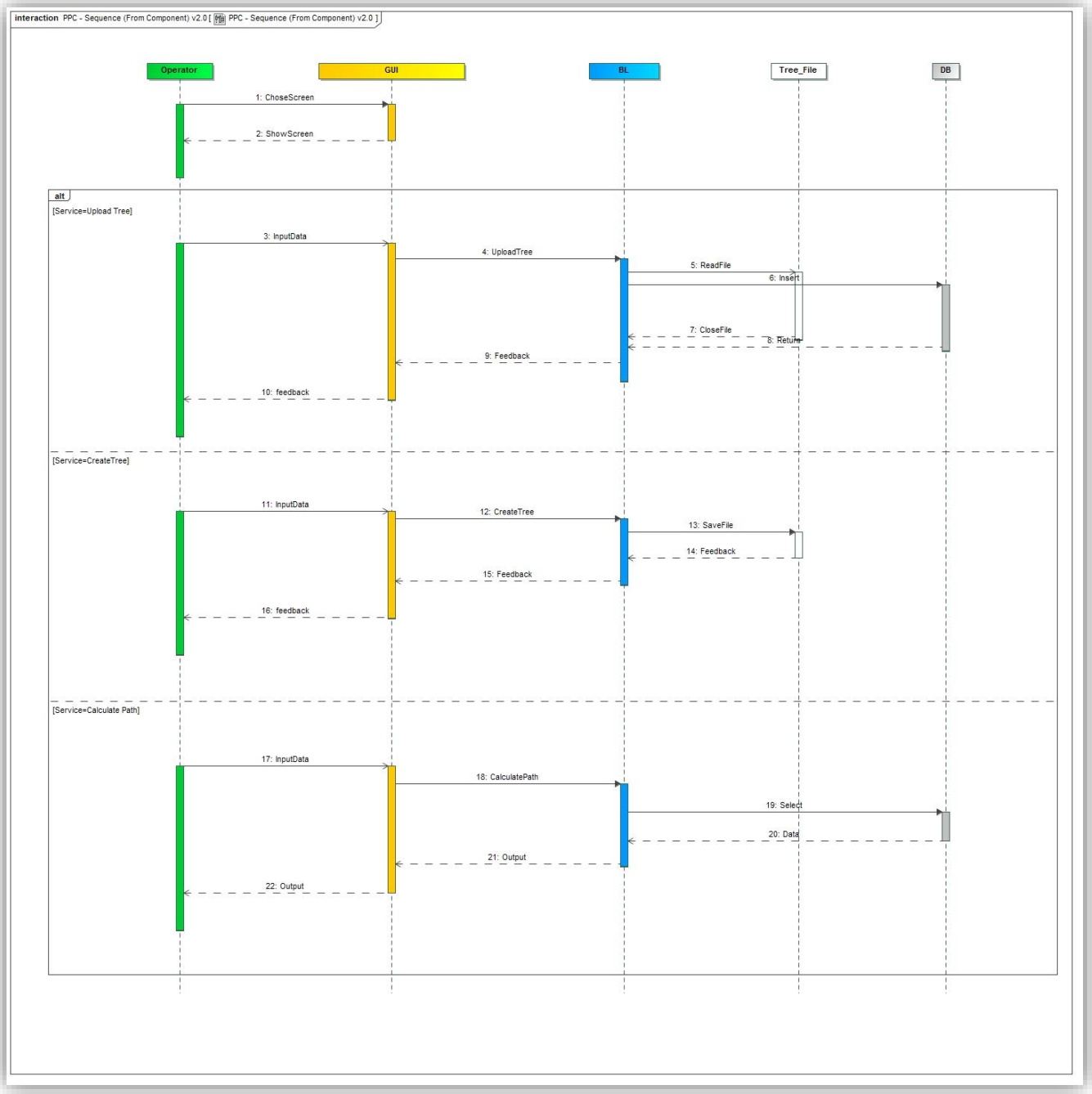
Nome	**Create Tree**
Accessibilità	Si può avere accesso solo tramite GUI
Parametri	SplitSize, Depth, Type, AttributeVertexList, AttributeEdgeList, NomeFile
Vincoli	L'interfaccia dovrà prevedere il salvataggio obbligatorio dell'albero in uno specifico formato di file (non ancora deciso) La directory dove vengono salvati i file sarà unica e predefinita dal sistema
Funzione	Prende l'input, genera l'albero strutturato e lo salva su un file

Nome	**Upload Tree**
Accessibilità	Si può avere accesso sia da GUI che da sistema esterno
Parametri	NomeFile
Vincoli	Il file deve esistere nella directory Il file deve essere già stato modificato da un sistema esterno Il file non deve essere già stato uploadato sul database
Funzione	Prende il file, ne verifica la validità, lo inserisce nel DB

Nome	**Calculate Path**
	Si può avere accesso sia da GUI che da sistema esterno
Parametri	Type, StartVertexA, EndVertexB
Vincoli	L'albero deve esistere nel database Devono esistere i vertici A e B Deve esistere il percorso tra i due vertici
Funzione	Interroga il DB, verifica l'esistenza dell'albero, dei vertici e del path ricercato, estrae i dati e li elabora, ritorna un output

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
- GUI
- BL
- Tree_File
- DB

Operator: Si tratta dell'Utente Micron che andrà ad usufruire del sistema, come si evince egli si interfaccia solo ed esclusivamente con una **GUI** 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, il 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 Upload di un file già esistente sul DB (UploadTree)
2. Servizio di Generazione di un file contenente l'albero (CreateTree)
3. Servizio che permette il calcolo sugli attributi nel percorso tra due nodi di uno specifico albero (CalculatePath)

ALT. [Service= UploadTree]:

L'operatore compila i dati in input tramite la GUI (3), invoca la funzione del BL relativa passandogli i parametri prima inseriti (4).
Il motore BL accede al file le cui coordinate erano contenute nell'input, acquisisce da esso i dati strutturati dell'albero e li inserisce nel DB tramite una Query di INSERT (5)(6)(7).
Come si può notare ci aspettiamo che non sia necessario attendere la completa lettura del file prima di iniziare l'accesso a DB; tuttavia è fondamentale attendere che la lettura sia utlimata prima di decretare la chiusura della connessione con il DB.
Ritornato il feedback positivo dal DB, esso viene propagato per tutte le parti coinvolte fino ad essere riportato alla GUI che lo mostra a schermo (8)(9)(10).

ALT. [Service=CreateTree]:

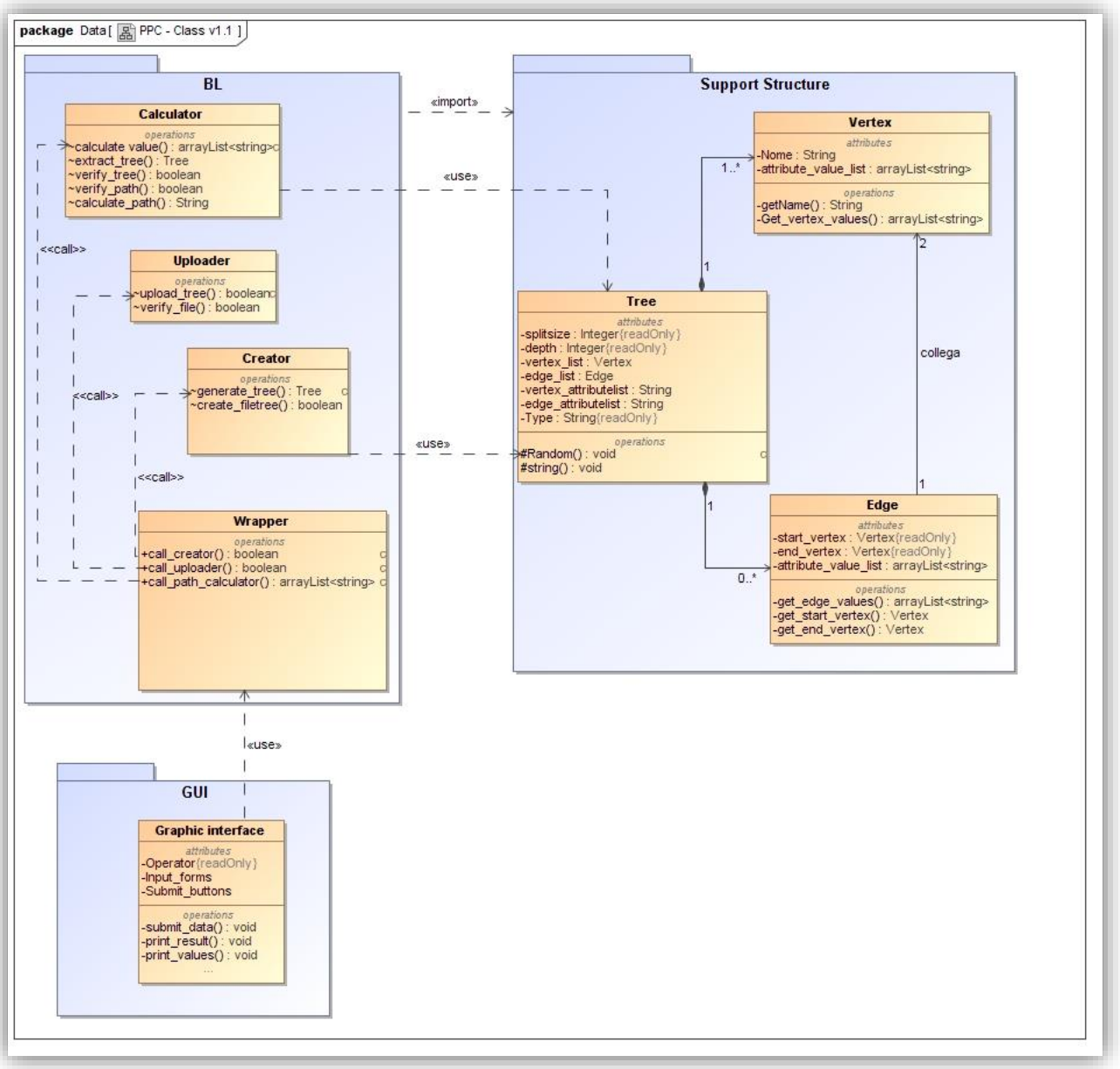
L'operatore compila i dati in input tramite la GUI e li invia al BL (11)(12).
Quest'ultima elabora i dati in modo da generare un albero e li salva in maniera strutturata su un file (13).
Il Feedback positivo, restituito dal salvataggio del file, viene ritornato dapprima al BL e poi mostrato a schermo dalla GUI (14)(15)(16).

ALT. [Service=CalculatePath]

L'operatore compila i dati in input tramite la GUI e li invia al motore BL (17)(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)

Si è scelto di implementare tutte le richieste di servizi tramite chiamate sincrone, in modo da limitare il numero di chiamate e calcoli contemporanei da parte di un singolo utente, in grado di appesantire ulteriormente il sistema, considerando che tali funzionalità sono messe a disposizione di più users nello stesso istante (multi-accesso). Il singolo utente perciò sarà in grado di ottenere un servizio alla volta. Solo dopo aver ricevuto l'output richiesto sarà permesso immettere nuovi dati in input.

C.3 – Class Diagram



Abbiamo individuato le seguenti Classi, raggruppate in package distinti, che rispecchiano la struttura del component diagram e ne mantengono i principi logici, punto di partenza per le future scelte di design.

Package BL, contenente le classi preposte all'implementazione delle operazioni proprie dell'Engine.
Package GUI, che contiene le classi utilizzate per generare la GUI e i relativi servizi.
Package Support Structure, contenente quelle classi artefici dell'astrazione dei dati.
Leggiamo nel dettaglio l'insieme delle classi contenute ed individuate a questo livello di raffinamento (volutamente ancora di alto livello).

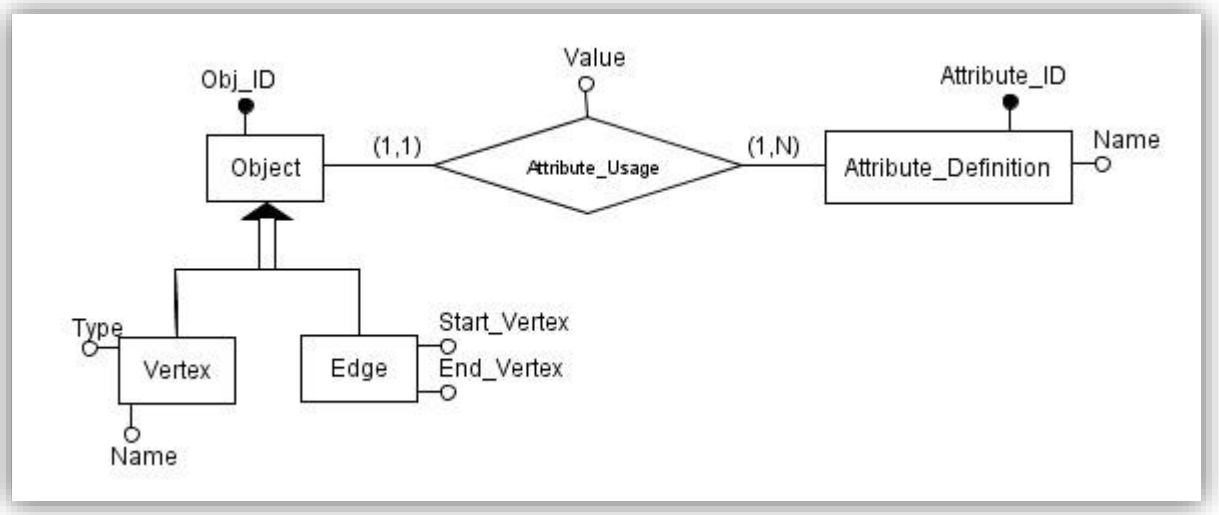
La classe GUI INTERFACE è, contestualmente a questo livello di dettaglio, ancora l'unica contenuta nel Package relativo. Essa realizza ovviamente l'interfaccia utente e la chiamata dei servizi richiesti.

Le classi del package BL sono Creator, Uploader e Calculator, che realizzano le corrispondenti funzioni di generazione di file albero, caricamento di quest'ultimo su Database e Calcolatore di Path in base ai parametri forniti, da operatore Micron che faccia uso di GUI, oppure da terze parti tramite sistema esterno.

Riteniamo utile demandare l'invocazione di queste classi e relativi metodi ad una classe Wrapper, che ha un ruolo di interfaccia tra il package stesso e la GUI o gli eventuali sistemi esterni che richiedono i servizi offerti.

Le classi contenute in Support Structure sono ovviamente Vertex e Edge, che assieme compongono la classe Tree.
Conterranno attributi e metodi che definiscono alberi, vertici ed archi, permettendone l'elaborazione da parte delle classi di BL.
Suddividendo in questo modo le classi e i package riteniamo soddisfacente il livello di dettaglio raggiunto per questa fase di raffinamento.

D.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, in tale relazione è possibile trovare il campo Value che conterrà il valore attribuito a quell'attributo assegnato a quello specifico object.

F.1 – Design Decision

Tutte le Decisioni di Design mostrate in questo Capitolo sono frutto di riflessioni e stime elaborate dal gruppo.
In alcuni casi prendere una decisione definitiva è risultato prematuro, tuttavia il Team ha preferito formalizzare comunque quelle che erano le idee a riguardo compiendo delle piccole scelte, al fine di allineare lo sviluppo futuro verso una strada condivisa che all'occorrenza potrebbe essere revisionata coerentemente allo scioglimento dei nodi più spinosi già mostrati nell'analisi dei Rischi (vedi tabella e paragrafo A.5).

1. RIPARTIZIONE LOGICA DELLE FUNZIONALITA'
   La specifica proposta dal committente, oltre a descrivere COSA il sistema dovesse essere in grado di offrire, dava anche dei suggerimenti più o meno espliciti su COME ripartire in diverse unità software (fino ad ora chiamate GUI e BL) le funzionalità da implementare.
   Se in un primo momento il team aveva preferito rimanere adeso alla specifica, ad un certo punto è risultato cruciale dover dare un interpretazione personale delle richieste, proponendo una nostra ristrutturazione che segue una semantica ben precisa.

   La GUI è stata quindi interpretata come un unità che faccia da tramite tra l'operatore ed il sistema; ad essa non saranno affidati compiti operativi, bensì si limiterà a raccogliere l'input dell'utente e ad inviarlo al motore BL.
   Motiviamo questa scelta da un doppio punto di vista, quello di fluidificare la complessità di progettazione (in questo modo lo sviluppo delle funzioni operative può procedere in parallelo con lo sviluppo della vera e propria interfaccia) e quello di realizzare un sistema che abbia una maggiore scalabilità futura. Supponiamo infatti che un domani si volessero implementare nuovi accessi alle funzionalità del BL, magari da altri sistemi, questo sarebbe possibile semplicemente intervenendo sul codice di BL senza toccare affatto la componente GUI.

2. DOVE GIRA IL NOSTRO SISTEMA
   Micron in tal senso si è espressa chiaramente dandoci carta bianca. Abbiamo pertanto cercato la scelta a più basso impatto economico tra quelle che abbiamo preso in analisi, e da quanto suggeritoci dalla nostra seppur scarsa esperienza a riguardo, l'alternativa migliore è quella di far girare tutto su un unico sistema multiutente.
   Scartiamo pertanto l'implementazione di un architettura client-server che poteva risultare più dispendiosa in termini di risorse e meno performante sui tempi di risposta.

3. A CHI AFFIDARE I CALCOLI PER MIGLIORARE L'EFFORT
   Sappiamo benissimo che ci sono operazioni che richiedono rapidi tempi di risposta e accessi concorrenti.
   Una di queste è proprio quella che si occupa di adempiere alla funzionalità di CALCULATE PATH.
   Per adempiere alla realizzazione di questa funzionalità sarà necessaria una lettura su DB, un controllo di coerenza del percorso (verificare se esiste l'albero e se esiste al suo interno il ramo selezionato) e l'esecuzione effettiva dei calcoli richiesti.

   Per adempiere al meglio a queste operazioni abbiamo pensato di dividere il "controllo di coerenza" dall'esecuzione effettiva dei calcoli.
   Il primo verrà effettuato direttamente su DB (poiché abbiamo in mente una strutturazione dell'albero tale che dovrebbe favorire tale operazione), mentre la seconda parte verrà eseguita dal motore BL.
   Questa strategia dovrebbe permetterci di ridurre la mole di calcoli superflui sull'albero, in quanto saremmo capaci di estrarre da DB soltanto il sottoramo che ci interessa, abbattendo enormemente la quantità di dati da dover maneggiare.

4. COME PRESENTARE LA LISTA DI ATTRIBUTI NELL'INTERFACCIA GRAFICA
   Sappiamo che un utente che genera un albero dovrà selezionare anche degli attributi da assegnare a vertici ed archi, sappiamo che tali attributi esistono già in una lista salvata su DB ma sappiamo anche che sarà possibile aggiungerne di nuovi all'occorrenza.

   Se inizialmente eravamo perplessi su come gestire questa cosa, dopo un contatto diretto con il committente, abbiamo appurato che la quantità di attributi che si stima verranno salvati nella lista su DB è piuttosto esigua e pertanto supponiamo di poter gestire tutto con un semplicissimo menu a tendina mostrato nell'interfaccia grafica dell'utente al quale daremo la possibilità di aggiungere nuove voci ove necessario.
5. DIRECTORY FILE
   Stando alla specifica, il file contenente l'albero deve essere salvato su una directory non meglio specificata.
   Inizialmente pensavamo fosse utile per ogni utente poter selezionare una directory diversa, andando avanti nella progettazione tale soluzione si è rivelata superflua e controproducente.
   Preferiamo quindi impostare noi una directory di default rendendo più semplice il lavoro anche a chi dovrà editare i suddetti file, potendoli trovare tutti insieme.
6. COME ASSICURARCI CHE NON VENGANO UPPATI FILE INCOMPLETI
   L'utente che editerà il file si dovrà anche preoccupare di settare un booleano "UPPABILE" a true.
   Il Bl, prima di procedere all'Upload controllerà quindi tale booleano e scarterà quei file che non sono idonei ad esser caricati.

G.1 – Requirements through Design

In questo capitolo ci preoccupiamo di offrire una mappatura esplicita di come le nostre scelte di Design vadano a soddisfare i Requisiti esposti nel capitolo A.
La tabella ci mostra due colonne:
nella colonna di sinistra abbiamo i requisiti con riferimenti al capitolo A di questo documento, mentre nella colonna di destra abbiamo le Design Decision che rispondono a questi requisiti con riferimenti al capitolo F di questo documento.

Requirements (Cap. A)	Design Decision (Cap. F)
A.1_GUI n.1 A.1_GUI n.2 A.1_GUI n.3	F.1_DD n.1 F.1_DD n.4
A.1_BL n.1	F.1_DD n.1
A.1_BL n.2	F.1_DD n.5
A.1_BL n.3	F.1_DD n.3
A.1_DB n.1 A.1_DB n.2	Vedi cap.D
A.2 n.1	F.1_DD n.2 F.1_DD n.3
A.2 n.2	F.1_DD n.1 F.1_DD n.5 F.1_DD n.6

Alla luce di quanto evidenziato da questa tabella notiamo come soprattutto i req NON funzionali (le ultime due entry della tab) siano stati soddisfatti da scelte mirate e ponderate.

Riteniamo interessante soffermarci sul requisito relativo alla scalabilità ed al multiaccesso (A.2 n.2).
Come facilmente intuibile si tratta di un requisito estremamente ampio e trasversale da soddisfare. Le scelte del Team in merito, infatti, sono ricadute sulla modularità del sistema (F.1_DD n.1), sulla semplificazione dello stesso (F.1_DD n.5) e sull'implementazione di alcune misure di sicurezza atte a garantire la consistenza delle informazioni salvate nel sistema.

Il Team pensa di essere sulla buona strada ma, come è naturale che sia, si riserva la possibilità di incrementare il design completo del sistema con nuove scelte complementari (o se necessario sostitutive) al fine di soddisfare ancor meglio i requisiti individuati.