Software Engineering

Course 2015/2016

Lecturer: Prof. Henry Muccini – henry.muccini@univaq.it

Planner Path Calculator Deliverable

Date	10/02/2016
Deliverable	4
Team	SharpNado

Team Members		
Nome	**Matricola**	**Email**
Francesco Manfredi	230207	francescomanfrediwd@gmail.com
Daniele Campli	230195	sharpnado@cx.33mail.com
Gianluca Scatena	227436	gianluca.scatena@student.univaq.it
Vincenzo Stoico	228563	vincenzo.stoico@student.univaq.it
Federico Tersigni	232835	federtersig@gmail.com
Jacopo Cavallo	229412	jacopo.cavallo@protonmail.com
Luca De Paulis	227926	depaulisluca@gmail.com

Indice

List of Challenging/Risky Requirements or Tasks

Challenging Tasks	Date of Task Identification	Date of Task Resolution	Challenge Management	ID
Organizzazione delle prime fasi di lavoro	17NOV15	17NOV15	Divisione dei compiti e imposizione delle scadenze.	CHA-01
Comprensione e disambiguazione dei requisiti forniti dal committente (primo elenco di domande).	17NOV15	18NOV15	Abbiamo posto le relative domande al committente.	CHA-02
Comprensione e disambiguazione dei requisiti e delle funzionalità forniti dal committente (secondo elenco di domande).	28NOV15	01DIC15	Risolti i dubbi chiedendo al committente.	CHA-03
Scelte di rappresentazione di vari concetti tramite Class Diagram	10DIC15	22DIC15	Richiesti chiarimenti al Docente sulla maniera più appropriata per rappresentare i concetti voluti; discussione di gruppo.	CHA-04
Esecuzione calcolo: dbms o engine?	07DIC15	21DIC15	Discussione in gruppo e con il committente per consistenza con i NFR. Vd. Nota 1	CHA-05
Dubbi su Analisys Model e sugli strumenti utili alla sua rappresentazione.	10DIC15	10DIC15	Richiesti chiarimenti al Docente e approfondimento sul libro di testo.	CHA-06
Discussione sul livello di astrazione da adottare per Use Case Diagrams e Component Diagram.	10DIC15	10DIC15	Chiesti consigli al Docente. Vd. Nota 2	CHA-07
Utilizzo del software di progettazione MagicDraw per costruire Component Diagram.	07DIC15	07DIC15	Consultazione del manuale utente di MagicDraw; consultazioni di svariati siti Internet; vari tentativi eseguiti “by chance”.	CHA-08
Comprensione della semantica intrinseca al Boundary Object.	17DIC15	22DIC15	Discussione con il Docente.	CHA-09

Mappatura tra i diversi diagrammi del progetto	14GEN16	15GEN16	Discussione e rielaborazione dei diagrammi	CHA-10
Definizione delle interfacce tra i diversi moduli	18GEN16	21GEN16	Discussioni successive dell'argomento alternate a fasi di sviluppo	CHA-11
Comprensione del funzionamento del server MSSQL e Visual Studio	16GEN16	18GEN16	Studio matto e disperato	CHA-12
Formattazione grafica della GUI	16GEN16	19GEN16	Ricerca da fonti varie sul web	CHA-13
Serializzazione e Deserializzazione del FILE per il passaggio da GUI ad Engine e viceversa	30GEN16	05FEB16	Ricerca da fonti varie sul web	CHA-14
Passaggio di parametri tra le finestre della GUI	30GEN16	06FEB16	Ricerca da fonti varie sul web	CHA-15
Connettività Client/Server	30GEN16	03FEB16	Ricerca da fonti varie sul web	CHA-16
Metodo di invio dei dati tra Client e Server	30GEN16	03FEB16	Ricerca da fonti varie sul web	CHA-17
Organizzazione dei dati da diversi alberi nel database	17GEN16	17GEN16	Creazione di un nuovo database per ogni albero. (Vedere dettagli in Design Decisions). Codice [DDE – 06] Vd.Nota 3	CHA-18
Recuperare lista alberi con dati di ognuno	17GEN16	17GEN16	Vd.Nota4	CHA-19
Manutenibilità del database	02FEB16	03FEB16	Un database per ogni albero comporta una riduzione della manutenibilità. (Maggiori dettagli nelle Design Decisions Database). Codice [DDE - 08]	CHA-20
Operazioni su alberi in database o in memoria	08FEB16	08FEB16	I metodi incaricati delle operazioni su indicate sono stati resi	CHA-21

			polimorfi. (Maggiori dettagli nelle Design Decisions Database). Codice [DDE -09] Vd.Nota 5	
Costruzione del parser/assembler XML;	30GEN15	06FEB15	Implementazione ad hoc di parser ed assembler, previa studio e documentazione.	CHA-22
Costruzione di compressore/decompressore EXI.	08FEB15	09FEB15	Analisi della documentazione ufficiale, test e debug estensivi.	CHA-23
Fase di testing a causa delle limitate macchine predisposte ad eseguire un ambiente di	30GEN15	10FEB15	Organizzazione per effettuare testing in modalità "best effort"; mai risolto del tutto.	CHA-24
Implementazione di metodi ausiliari per eseguire testing e debugging.	03FEB15	10FEB15	Non esiste una soluzione, soltanto pazienza	CHA-25
Ricerca di siti alternativi al portale msdn per poter utilizzare le risorse di .NET in maniera	03FEB15	10FEB15	Ricerca di siti alternativi al portale msdn per poter utilizzare le risorse di .NET in maniera	CHA-26
Analisi degli strumenti atti a sviluppare in maniera più efficiente (Git in primis).	03FEB15	08FEB15	La parte difficile è stata inserire altro materiale da leggere e studiare (che d'altronde fosse di "contorno" al codice, ovvero la sua gestione). Anche qui risolto con pazienza e forza d'animo.	CHA-27

NOTE

1. Questa scelta e' stata particolarmente ardua poichè è stato necessario uno studio preliminare su come engine e database dovessero effettivamente comunicare tra loro; a seguito di progressive valutazioni dei NFR, abbiamo congiuntamente scelto di applicare le scelte di design descritte nelle sez. E.
2. Per evitare di arricchire con un livello di dettaglio eccessivo i diagrammi, abbiamo proceduto per iterazioni (come consigliato a lezione d'altronde); nel caso degli Use Case Diagrams ogni iterazione ha richiesto una dose di attenzione consistente soprattutto per ricercare e semplificare le parti che non aiutavano la leggibilità o deterioravano la

comprensione con dettagli di poca rilevanza. Per quanto riguarda il Component Diagram, maggiore sforzo è stato impiegato per avere una visione di insieme del sistema tale da mantenere delle linee guida ma senza specificare ogni singola componente nella sua interezza (rischiando cosi di ricadere in una progettazione di tipo upfront/waterfall). E' stato dunque necessario investire del tempo per fare chiarezza su tali questioni, e, per riuscire nel nostro obiettivo, abbiamo dapprima consultato il libro di testo; a questo è stata unita la ricerca di esempi ausiliari sul web che esemplificassero una situazione simile alla nostra, e, infine, e' stato fondamentale avere un confronto con il Docente in modo da confermare quanto appreso lungo il processo di studio.

3. Creazione di un nuovo database per ogni albero salvato sul database server. In questo modo si ottengono tabelle di dimensioni minori e si tengono ben separate entità logicamente non connesse (i diversi alberi). (Vedere dettagli in Design Decisions Database).

4. Per il funzionamento del nostro sistema è necessario fornire all'utente in tempi rapidi una lista degli alberi disponibili sul database server completa delle informazioni ad essi legate (nome, tipo, lista attributi dei nodi, lista attributi degli archi). Avendo optato per salvare ogni albero in un database separato il recupero di queste informazioni diventa difficoltoso. (Vedere dettagli in Design Decisions Database).

5. Avevamo inizialmente trascurato un aspetto importante relativo alle operazioni sugli alberi come selezione di un percorso sull'albero, recupero e modifica dei valori su di essi: queste operazioni prevedevano di lavorare caricando ogni volta l'albero dal database e non contemplavano la possibilità di lavorare con alberi già tenuti in memoria dal componente di business logic.(Vedere dettagli in Design Decisions Database).

A. Requirements Collection

A.1 Functional Requirements

Ad ogni requisito funzionale e' stato assegnato un codice alfabetico di tre lettere (es.[Codice CRA]) in modo tale che esso sia tracciabile nei diagrammi prodotti e allo scenario corrispondente; tale codice verra' utilizzato come identificativo di riferimento nelle sezioni descriventi le scelte architetturali. Per ogni requisito è stato assegnato un ulteriore identificativo numerato (es.[REQ-01]) che dà un ordine cronologico ai functional requirements.

A.1.1 GUI Requirements

La Graphic User Interface dovrà provvedere a fornire le seguenti funzionalità:

- Generare un albero quando viene premuto un bottone “Build Tree” [REQ-01] *[Codice CRA]*;
- Consentire di invocare l'engine e di avviare tramite questo le seguenti operazioni [REQ-02]:

    1. esportazione di un file presente nel database sul filesystem [Nota 1][REQ-02.A] *[Codice ESF] ;*
    2. importazione di un file albero presente sul filesystem [Nota 1] nel database [REQ-02.B]*[Codice UPF]*;
    3. Creazione di un file contenente un albero costruito secondo i parametri

specificati in input dall'utente (split size, depth, attribute list, attribute value generation rule) [Nota 3] e salvataggio dello stesso sul filesystem [Nota 1] [REQ-02.C]*[Codice CRA]*;
4. calcolo della somma dei valori selezionati in input, dove questi ultimi sono essenzialmente ricavati facendo scegliere all'utente un albero target, un nodo iniziale ed uno finale (che costituiscono dunque un percorso) ed un insieme di attributi relativi ad archi e nodi [REQ-02.D]*[Codice CSA]*;
5. modifica dei valori di attributi previo input dall'utente costituito dalla selezione di uno specifico albero dai nodi e archi e dagli attributi relativi a questi ultimi che dovranno essere coinvolti nel calcolo [REQ-02.E] *[Codice MDV]*;

- deve consentire all'utente di inserire i parametri necessari per eseguire correttamente le operazioni che la GUI stessa e' in grado di avviare [REQ-03] [Codice INP] ;
- visualizzare un messaggio informativo che comunichi lo stato di successo o fallimento delle operazioni eseguite [Nota 2] [REQ-04] [Codice FDB] ;

A.1.2 Business Logic Requirements

La Business Logic deve:

- esportare un albero presente in database sul filesystem [Nota 1][REQ-05][Codice ESF] ;
- supportare l'importazione di un file da filesystem [Nota 1] nel database[REQ-06] [Codice UPF] ;
- creare un albero secondo i parametri specificati in input dall'utente (split size, depth, attribute list, attribute value generation rule) [Nota 3] e salvarlo su un file [REQ-07] [Codice CRA];
- fornito in input dall'utente un percorso (costituito da un nodo iniziale ed uno finale) e selezionato un insieme di attributi relativi agli archi e ai nodi coinvolti , calcolare la somma dei valori contenuti in tali attributi [REQ-08][Codice CAL];
- restituire un feedback attraverso un codice di esito operazione [REQ-09] [Codice FDB] ;
- modificare dei valori di attributi avendo in input dall'utente un albero, un insieme di nodi/archi e gli attributi target delle modifiche volute [REQ-10][Codice MDV];
- poter essere utilizzato da sistemi esterni che lo invocano [REQ-11];

A.1.3 DB Requirements

Il database dovrà:

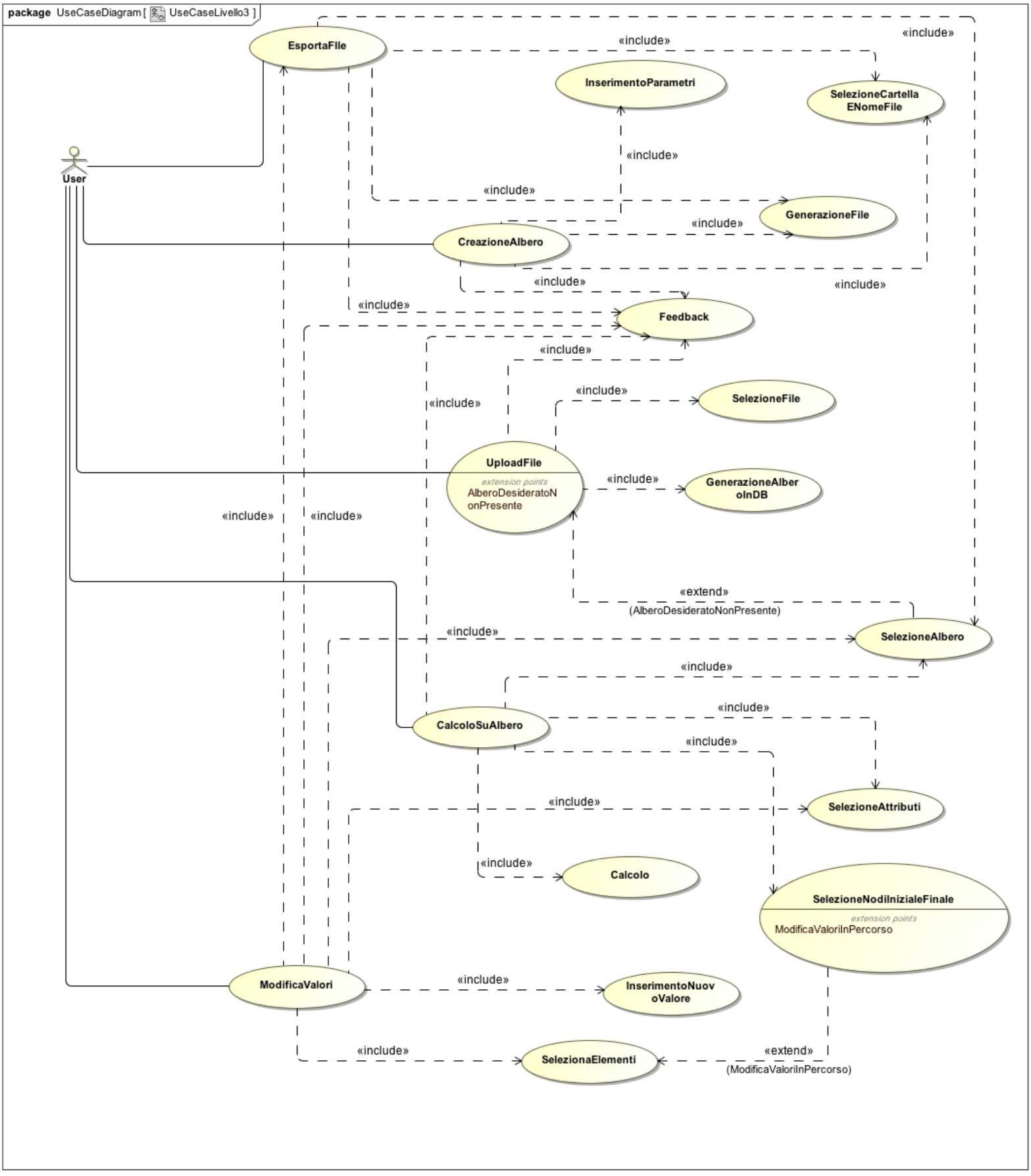
- essere capace di importare file albero salvati sul filesystem [REQ-12][Codice GDB];
- contenere una struttura logica (i.e. una tabella) chiamata AttributeDefinition, la quale racchiuderà tutti i tipi degli attributi degli archi e dei nodi [REQ-13];

Note

1. Si vuole precisare che con il termine "filesystem" si delineano tutti i filesystem che il sistema operativo eseguente l'engine e' in grado di riconoscere e gestire.
2. Lo stato di operazione terminata con successo verra' notificato con il risultato dell'operazione eseguita (eg. conseguentemente ad un calcolo della somma terminato correttamente verrà illustrato il valore ottenuto), mentre, qualora si presentasse un insuccesso, verranno forniti i dettagli noti riguardanti l'errore riscontrato.

3. Definizione dei parametri immissibili come input:
    - SplitSize: indica quanti vertici devono essere generati partendo dalla radice;
    - Depth: esprime la profondità dell’albero da creare;
    - AttributeList: fornisce una lista di attributi da associare separatamente a vertici ed archi;
    - AttributeValueGenerationRule: specifica se inizializzare il valore degli attributi tramite numeri randomici compresi in un range selezionabile, oppure tramite una stringa unica definibile dall’utente.

A.1.4 Requirements Modeling Through Use Case Diagrams



Use Case Name	**CreazioneAlbero**
Actors	User
Entry Conditions	-
Description	Il sistema offre all'utente la possibilità di creare un albero.
Codice	CRA
Reference Scenario	L'utente specifica i parametri per il nuovo albero (Nome, Tipo, SplitSize, Depth, Attribute Definition, Vertex Attribute List, Edge Attribute List, Attribute Value Generation Rule). L'utente specifica la directory di destinazione del file albero. Il sistema genera un file albero. Il sistema da un feedback all'utente per il successo o il fallimento dell'operazione.
Exit Conditions	Un nuovo file albero è stato generato.
Exceptions	
Include	InserimentoParametri, SelezioneCartellaENomeFile, GenerazioneFile, Feedback
Included in	-
Extends	-
Extended By	-
Notes	-

Use Case Name	**UploadFile**
Actors	User
Entry Conditions	Deve esistere file albero da caricare
Description	Il sistema permette all'utente di caricare un file albero nel database per effettuare calcoli e modifiche su di esso.
Codice	UPF
Reference Scenario	L'utente seleziona un file albero esistente. Il sistema genera un albero nel database a partire dal file indicato a condizione che non esista già un albero con stesso nome e tipo. Il sistema da un feedback all'utente per il successo o il fallimento dell'operazione.
Exit Conditions	Un nuovo albero è presente nel database e disponibile per calcoli e modifiche.
Exceptions	La creazione fallisce con avviso all'utente se esiste già un albero con lo stesso Nome o Tipo nel database;
Include	SelezioneFile, GenerazioneAlberoInDB, Feedback
Included in	-
Extends	SelezioneAlbero
Extended By	-
Notes	-

Use Case Name	**CalcoloSuAlbero**
Actors	User

Entry Conditions	L'albero su cui fare calcoli è presente nel database o esiste file albero da caricare nel database.
Description	Il sistema permette all'utente di effettuare calcoli su alberi (solo somma di attributi numerici).
Codice	CSA
Reference Scenario	L'utente seleziona un albero già presente nel Database. L'utente seleziona gli attributi dei nodi o dei vertici su cui vuole effettuare i calcoli. Sceglie due nodi dell'albero che definiranno un percorso quindi sceglie un nodo iniziale ed uno finale. Viene effettuato il calcolo. Viene rilasciato un Feedback sull'avvenuta operazione.
Exit Conditions	Le informazioni richieste sono presentate all'utente
Exceptions	Non esistono nodi o archi su cui si vuole effettuare il calcolo; Non è possibile effettuare il calcolo richiesto;
Include	SelezioneAlbero, SelezioneAttributi, SelezioneNodiInizialeFinale, Calcolo, Feedback
Included in	-
Extends	-
Extended By	-
Notes	-

Use Case Name	**SelezioneAlbero**
Actors	User
Entry Conditions	L'albero deve esistere nel DB o sottoforma di file
Description	Funzionalità che permette di selezionare un albero dal DB o dal filesystem per eseguire varie operazioni su di esso.
Codice	SAL
Reference Scenario	L'utente sceglie un file albero oppure un albero già presente nel Database.
Exit Conditions	L'albero è indicato come oggetto delle operazioni da eseguire.
Exceptions	-
Include	-
Included in	EsportaFile, CalcoloSuAlbero, ModificaValori
Extends	-
Extended By	UploadFile
Notes	-

Use Case Name	**EsportaFile**
Actors	User
Entry Conditions	Un albero è presente nel database e lo si vuole esportare in un file.
Description	Un utente può decidere di esportare un albero presente nel database in un file.
Codice	ESF

Reference Scenario	L'utente sceglie un albero dal Database. L'utente specifica una directory dove verrà salvato il file. Viene generato un file albero che viene salvato nel file system. Viene generato un Feedback sull'avvenuta operazione.
Exit Conditions	Una copia dell'albero presente in un database si trova in un file.
Exceptions	-
Include	SelezioneAlbero, SelezioneCartellaENomeFile, GenerazioneFile, Feedback
Included in	ModificaValori
Extends	-
Extended By	-
Notes	-

Use Case Name	**SelezioneCartellaENomeFile**
Actors	User
Entry Conditions	L'utente sta creando un nuovo albero o esportando un albero già presente nel database in un file sul filesystem.
Description	Permette all'utente di salvare un file in un determinata posizione del file system.
Codice	SCF
Reference Scenario	Un utente specifica una directory ed un file quando si sta generando un nuovo file albero.
Exit Conditions	Si sono fornite alla procedura in corso le informazioni su dove e con che nome generare un file.
Exceptions	- Directory non esistente - File già esistente
Include	-
Included in	EsportaFile, CreazioneAlbero.
Extends	-
Extended By	-
Notes	Attenzione: sembra simile allo use case SelezioneFile, ma si usa in casi differenti.

Use Case Name	**Feedback**
Actors	User
Entry Conditions	Un'operazione richiesta dall'utente si è conclusa e l'utente deve essere notificato dell'esito.
Description	L'utente viene notificato dell'esito di un'operazione richiesta al sistema. Questo use case è generico e riutilizzato per tutte le operazioni che richiedono una comunicazione all'utente.
Codice	FDB
Reference Scenario	Viene mostrata una finestra che informa l'utente del successo o del fallimento dell'operazione richiesta.
Exit Conditions	L'utente è a conoscenza dell'esito dell'operazione richiesta al sistema.

Exceptions	-
Include	-
Included in	EsportaFile, CreazioneAlbero, ModificaValori, CalcoloSuAlbero, UploadFile
Extends	-
Extended By	-
Notes	-

Use Case Name	**SelezioneNodiInizialeFinale**
Actors	User
Entry Conditions	L'utente deve aver selezionato un albero sul quale eseguire il calcolo.
Description	Il sistema permette all'utente di effettuare calcoli su alberi facendo selezionare due nodi, iniziale e finale, che identificano il path sul quale eseguire tale calcolo.
Codice	SNF
Reference Scenario	L'utente sceglie due nodi, iniziale e finale.
Exit Conditions	Si sono fornite alla procedura in corso le informazioni sul path sul quale eseguire il calcolo.
Exceptions	L'operazione fallisce con avviso all'utente se i nodi selezionati non esistono.
Include	
Included in	CalcoloSuAlbero
Extends	SelezionaElementi
Extended By	-
Notes	-

Use Case Name	**ModificaValori**
Actors	User
Entry Conditions	Deve esistere già un albero sul quale effettuare tale modifica.
Description	Il sistema permette all'utente di modificare i valori degli attributi di un nodo o di un arco.
Codice	MDV
Reference Scenario	L'utente sceglie un albero dal Database. L'utente sceglie gli elementi dell'albero da modificare. Può sceglie un arco o un nodo oppure tutti e due. L'utente seleziona gli attributi, degli elementi selezionati, che vuole modificare. L'utente inserisce i nuovi valori degli attributi scelti. Viene esportato il nuovo file albero modificato. Viene mostrato una notifica sull'avvenuta operazione.
Exit Conditions	Valori degli attributi modificati.
Exceptions	Valori inseriti non appartenenti al dominio corretto.
Include	SelezioneAlbero, SelezioneAttributi, InserimentoNuovoValore, SelezionaElementi, EsportaFile,Feedback.
Included in	-
Extends	-

Extended By	-
Notes	EsportaFile è di base incluso in questo use case perché il file si considera il mezzo principale per la persistenza dei dati.

Use Case Name	**Calcolo**
Actors	User
Entry Conditions	Sono stati gia' selezionati attributi e elementi (siano essi archi o nodi) su cui eseguire il calcolo.
Description	Esegue il calcolo di somma sugli attributi degli elementi selezionati.
Codice	CAL
Reference Scenario	Viene effettuato il calcolo.
Exit Conditions	Calcolo eseguito con successo ritornando un valore numerico.
Exceptions	-
Include	-
Included in	CalcoloSuAlbero
Extends	-
Extended By	-
Notes	-

Use Case Name	**SelezioneAttributi**
Actors	User
Entry Conditions	Un path di archi e nodi su un albero è già stato selezionato.
Description	Consente di selezionare gli attributi (sia di nodi che di archi) su cui si vogliono eseguire calcoli.
Codice	SAT
Reference Scenario	L'utente seleziona gli attributi.
Exit Conditions	Si sono fornite alla procedura in corso le informazioni sugli attributi su cui effettuare il calcolo.
Exceptions	-
Include	-
Included in	CalcoloSuAlbero, ModificaValori
Extends	-
Extended By	-
Notes	-

Use Case Name	**Inserimento Parametri**
Actors	User
Entry Conditions	L'utente ha richiesto al sistema la creazione di un nuovo albero.
Description	L'utente ha la possibilita di inserire gli attributi e i valori di questi con il fine di creare un nuovo albero.

Codice	INP
Reference Scenario	L'utente specifica i valori per: • Split Size : Il numero di vertici che vengono generati da un dato nodo. • Depth: La profondità dell'albero. • Attribute List: lista degli attributi dei nodi e dei vertici L'utente, inoltre, deve scegliere come generare i valori degli attributi specificati in AttributeList tra due possibili modalità: • Random(K-N): l'engine assegna automaticamente i valori agli attributi generando in modo casuale un numero tra K-N. • String(Value): Il valore viene generato come una stringa di valore “Value”
Exit Conditions	Il sistema ha le informazioni sui parametri su cui è definito l'albero.
Exceptions	L'utente non ha inserito tutti i campi richiesti quindi l'operazione non può essere completata.
Include	
Included in	Creazione Albero
Extends	-
Extended By	-
Notes	-

Use Case Name	**Generazione File**
Actors	User
Entry Conditions	L'albero deve essere definito nella sua completezza o presente nel database.
Description	Il sistema genera il file in cui sono presenti i dati che definiscono l'albero. Questo file può essere importato nel database oppure esportato da quest'ultimo.
Codice	GEF
Reference Scenario	Il sistema genera il file albero.
Exit Conditions	Un nuovo file albero è stato generato.
Exceptions	Il sistema non può generare il file poiché è in corso un'operazione sull'albero.
Include	
Included in	CreazioneAlbero, EsportaFile
Extends	-
Extended By	-
Notes	

Use Case Name	**InserimentoNuovoValore**
Actors	User
Entry Conditions	L'utente sta lavorando su un albero ed ha già stabilito quali attributi modificare.
Description	Selezionato un attributo, permette di inserire un nuovo valore.
Codice	INV
Reference Scenario	L'utente seleziona un nuovo valore per un attributo presente nell'albero.
Exit Conditions	Si sono forniti alla procedura in corso nuovi valori per gli attributi

	precedentemente selezionati.
Exceptions	Il nuovo valore non appartiene al dominio corretto.
Include	-
Included in	ModificaValori
Extends	-
Extended By	-
Notes	-

Use Case Name	**SelezionaElementi**
Actors	User
Entry Conditions	L'utente sta lavorando su un albero.
Description	Il sistema offre la possibilità di selezionare determinati elementi di un albero su cui effettuare delle modifiche. È possibile selezionare un elemento singolo (arco o nodo), più elementi non collegati o più elementi appartenenti ad un path sull'albero (vedere extend).
Codice	SEL
Reference Scenario	L'utente seleziona gli elementi dell'albero su cui vuole lavorare.
Exit Conditions	Si sono forniti alla procedura in corso gli elementi che saranno oggetto di modifica.
Exceptions	-
Include	-
Included in	ModificaValori
Extends	-
Extended By	SelezionaNodiInizialeFinale
Notes	-

Use Case Name	**SelezioneFile**
Actors	User
Entry Conditions	Deve esistere il file dell'albero da selezionare.
Description	Il sistema fornisce la possibilità di caricare nel database un albero precedentemente definito in un file. Per farlo è necessario accedere alla funzione di upload, da cui selezionare il medesimo file.
Codice	SFI
Reference Scenario	L'utente seleziona il file su cui vuole lavorare.
Exit Conditions	Il file albero è stato selezionato ed è pronto all'upload.
Exceptions	-
Include	-
Included in	UploadFile
Extends	-
Extended By	-
Notes	-

Use Case Name	**GenerazioneAlberoInDB**
Actors	User
Entry Conditions	Deve essere stato selezionato un file albero.
Description	Il sistema permette di generare un nuovo albero nel database.
Codice	GDB
Reference Scenario	Un nuovo albero viene generato nel database.
Exit Conditions	Il database ha un nuovo albero.
Exceptions	L'albero è già stato generato nel database o è presente un albero differente con stesso nome o tipo.
Include	-
Included in	UploadFile
Extends	-
Extended By	-
Notes	-

A.1.5 Requirements Prioritization

Requirement	**Priorità**	**Motivazione**
CreazioneAlbero	Alta	Funzione indispensabile del sistema
InserimentoParametri	Alta	Senza questa funzionalità si creerebbero alberi con caratteristiche di default poco utili all'utente
UploadFile	Alta	Il file è il metodo principale di salvataggio degli alberi ma è necessario caricare i dati su database per eseguire alcune delle funzionalità ad alta priorità
SelezioneFile	Alta	Indispensabile per la funzionalità UploadFile indicata con priorità alta.
EsportaFile	Media	È indispensabile alla funzionalità di modifica dei valori di un albero che ha priorità media.
Feedback	Bassa	Operazione di supporto che avvisa le varie componenti,compreso l'utente, che un'operazione è stata eseguita correttamente oppure no.

SelezionaCartellaeNomeFile	Bassa	Non indispensabile per le funzionalità EsportaFile e Creazione File indicate con priorità Alta. Non è indispensabile poiché possiamo supporre di salvare i file albero in una porzione di memoria standard del nostro sistema con nomi generati automaticamente. Per esempio possiamo avere nel file system una cartella chiamata "path_calc_files" dove quando si esportano,i files, vengono salvati automaticamente qui con nomi del tipo: "file_01","file_02".
GenerazioneFile	Alta	Funzionalità indispensabile che assicura la persistenza dei dati . Senza la creazione del file l'albero non può essere importato nel Database e quindi non può essere utilizzato.
GenerazioneAlberoInDB	Alta	Funzionalità indispensabile per il sistema. Permette di avere una albero nel Database quindi permette all'utente di lavorare con quest'ultimo.
SelezioneAlbero	Media	Indispensabile per le funzionalità ModificaValori ed EsportaFile indicate con priorità media.
CalcoloSuAlbero	Alta	È una delle funzionalità principali del sistema esplicitamente richieste dal committente.
SelezioneAttributi	Bassa	La selezione di attributi specifici da sommare può essere implementata dopo la funzionalità di somma. Senza questa funzionalità è comunque possibile effettuare il calcolo su un albero per tutti gli attributi a disposizione.
Calcolo	Alta	È la vera è propria computazione necessaria per svolgere l'operazione di calcolo su un albero. Indispensabile alla funzionalità CalcoloSuAlbero ad alta priorità.

ModificaValori, SelezionaElementi e InserimentoNuovoValore	Media	La funzionalità ModificaValori e le altre due funzionalità annesse, sono funzionalità aggiuntive offerte da noi al cliente. Riteniamo che queste funzionalità non siano fondamentali per il corretto utilizzo del sistema ma possono portare, nel caso in cui l'utente si trovi a dover cambiare valori, a una diminuzione dei carichi di lavoro per il sistema. Per cambiare un valore non sarà necessario ricreare l'intero albero.
SelezionaNodoInizialeFinale	Alta	Data la necessità di avere alberi con un numero elevato di nodi, fare calcoli su tutti i nodi sarebbe troppo dispendioso in termini di prestazioni, è quindi molto utile permettere all'utente di indicare il sottoinsieme di nodi da considerare.

A.2 Non-Functional Requirements

I requisiti non funzionali di maggiore importanza che il sistema deve garantire sono:

- Performance
- Scalabilità
- Maintainability e Modularità

Perché Performance?

- Questo sistema sarà interrogato da un sistema di produzione, perciò deve essere rapido a ritornare il risultato dei calcoli richiesti, come specificato dal customer.

Perché Scalabilità?

- Questo sistema deve supportare un numero di accessi concorrenti (tra i 10 ed i 20 nei periodi di picco) che potrebbero aumentare con l'avanzare del tempo, ragion per cui deve essere scalabile come da specifica.

Perché Maintainability e Modularità?

- Poiché è noto già dal principio che il nostro DB non verrà utilizzato dal customer e che sarà sostituito, abbiamo capito che la maintainability e la modularità sono due requisiti non funzionali importanti atti a rendere la modifica e l'adattamento del nostro sistema più agevole.

A.3 Contents

I dati utilizzati dal sistema provengono direttamente dall'utente finale che ne farà uso e sono

conservati in file albero. Un database è utilizzato per effettuare varie operazioni sugli alberi.

A.4 Assumptions

[Codice ASS-01]: Abbiamo assunto (soddisfando il requisito non funzionale di scalabilita') che il sistema supportera' un massimo di 20 connessioni concorrenti.

[Codice ASS-02]: Abbiamo assunto che il sistema integrabile entro i confini di una infrastruttura di rete locale già predisposta verso vettori quali "responsiveness", performance e sicurezza. L'architettura del sistema provvedera' a fornire un modello di scalabilita' tale da mantenere le performance pressoche' costanti (entro i limiti di concorrenza assunti precedentemente) qualora piu' utenti richiederanno i servizi offerti dal sistema.

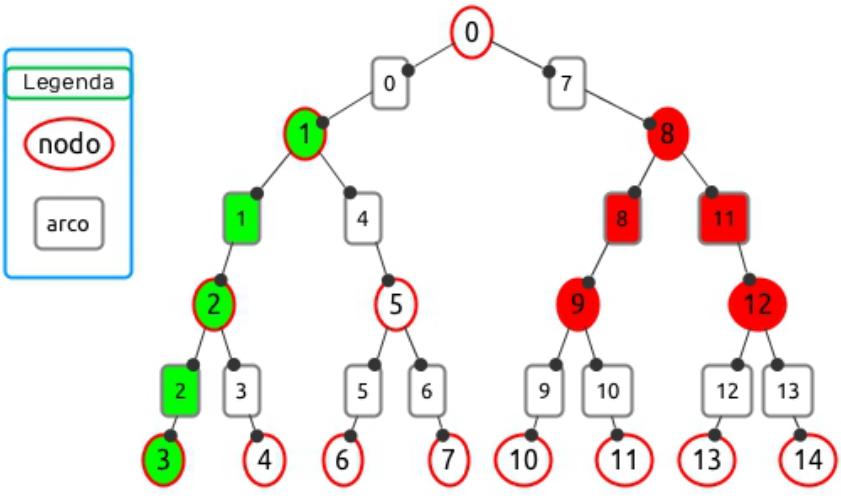
[Codice ASS-03]: Un utente può decidere di modificare un albero già caricato nel DB in qualsiasi momento; inoltre, come un file contente un albero può essere importato nel DB, un albero già caricato può essere esportato su un file prima e/o dopo una modifica. Queste due funzionalita', pur non essendo richieste esplicitamente dal committente, verranno fornite in quanto assicurano maggiore flessibilita' d'utilizzo.

[Codice ASS-04]: Stimando che un albero abbia in media un milione di vertici (con un massimo di due milioni), è chiaramente desiderabile (come da specifica d'altronde) che l'utente possa eseguire i calcoli su range di archi e nodi, senza doverli selezionare manualmente; tuttavia, rispetto alla specifica iniziale, il nostro team ha ritenuta utile aggiungere una funzionalità che sia in grado di accettare range di selezione multipli, anche disgiunti (i.e. senza elementi in comune), in modo da agevolare situazioni in cui occorra calcolare la somma di piu' percorsi diversi all'interno dello stesso albero.

[Codice ASS-05]: Si assume che il sistema disponga di risorse di calcolo adeguate alla mole di dati che si intende manipolare.

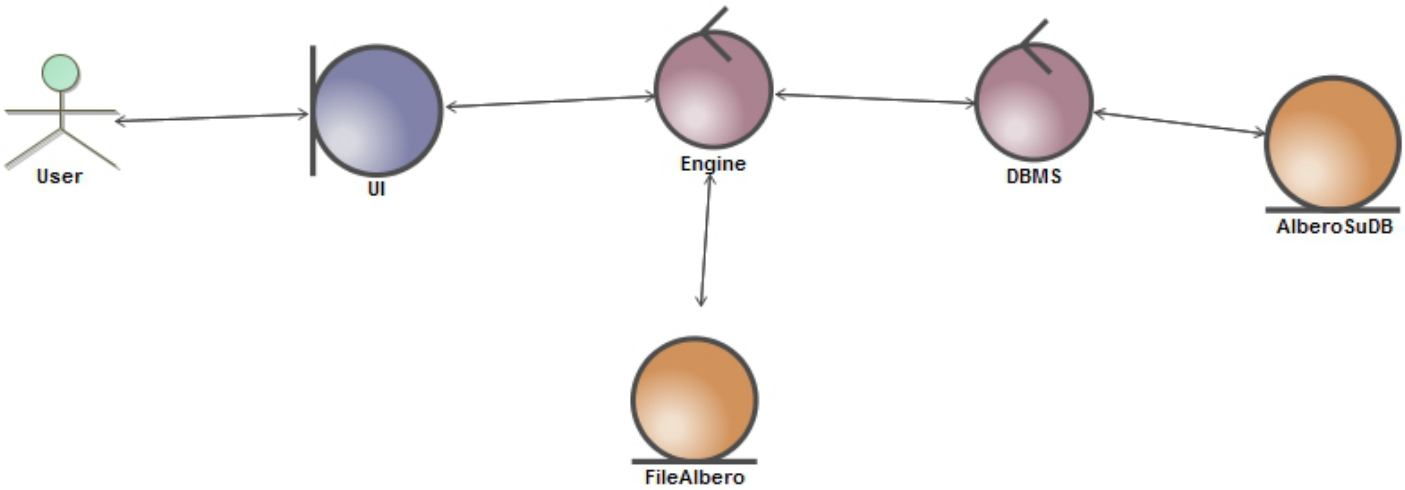
[Codice ASS-06]: Si assume che un'importazione di un file albero contenga effettivamente XML relativo ad un'oggetto albero, e non sia effettuata con file differenti. Un'eccezione verrà lanciata qualora si verificasse quest'ultima situazione.

[Codice ASS-08]: Si assume che il calcolo venga eseguito fornendo due nodi che si trovano nello stesso sottoalbero (dove per sottoalbero non si intende il caso matematico dell'albero intero); pertanto è possibile effettuare il calcolo soltanto su un percorso che sia "verticale". Per comprendere la logica (tramite costruzione preorder) con cui vengono forniti gli identificativi ecco una piccola infografica: il percorso verde è un percorso lecito, mentre quello colorato di rosso non è ammesso.



B. Analysis Model

Partendo dai requisiti e dagli Use Case Diagrams siamo riusciti ad identificare le componenti principali del sistema e suddividerle nei tre tipi di oggetti: Boundary, Control ed Entity.



Come si evince dal diagramma riportato in figura, l'utente può accedere al sistema tramite la UI. La UI, identificata come boundary-object, è responsabile della comunicazione da/verso l'Engine, in quanto permette all'utente di inserire dati, scegliere le operazioni da effettuare e ricevere feedback relativi alle operazioni eseguite. Esempi di comunicazione “da/verso l'Engine” sono le funzionalità come “CreazioneAlbero”
(User → UI → Engine → …) e Feedback (... → Engine → UI → User).

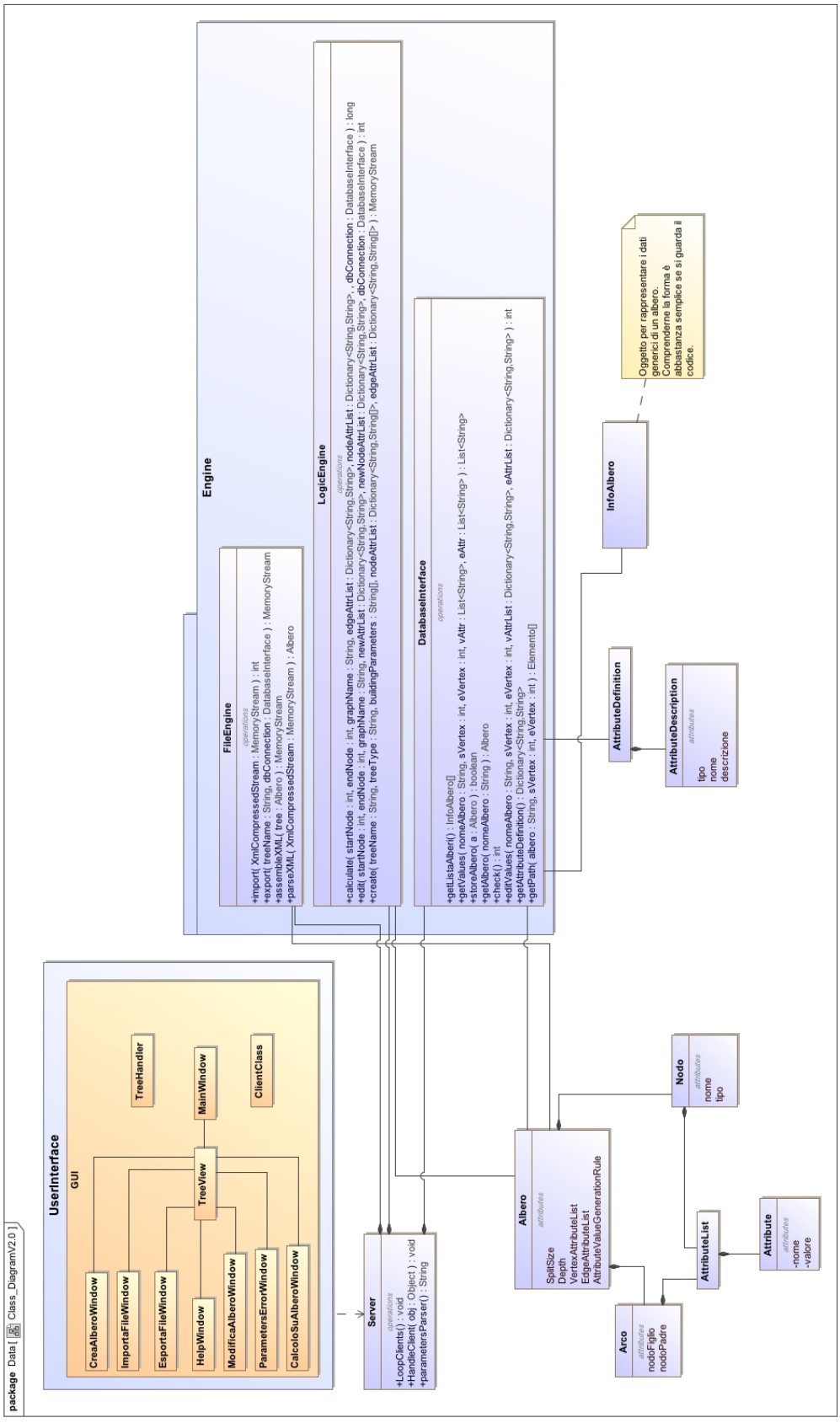
Il control-object Engine è l'elemento principale del sistema in quanto attraverso esso provvede a fornire tutte le funzionalità cardine (per maggiori informazioni, consultare la sezione C.2 Component Diagram).

Altro elemento importante è il control-object DBMS che gestisce tutte le queries al database.
Da notare come nel precedente deliverable questo fosse “collegato” al control-object Engine attraverso il boundary-object Dbinterface; tale scelta tuttavia risultava implicare il database come un elemento esterno al sistema, e di conseguenza e' stata rivalutata.

Nel nostro sistema abbiamo identificato due entity-object principali che sono FileAlbero e AlberoSuDB. L'entity-object FileAlbero e' rappresentativo del file (mantenuto nel filesystem) nel quale si trovano tutte le informazioni relative ad un albero. Con AlberoSuDB si indicano i dati presenti nel database costituenti un albero e la sua AttributeDefinition, ossia la struttura dati (richiesta dal committente secondo specifica, vedi REQ-13) che contiene gli attributi assegnati agli elementi dell'albero stesso. Abbiamo ritenuto importante evidenziare la distinzione tra FileAlbero e AlberoSuDB perché in seguito alle domande poste al cliente è stato esplicitamente chiarificato che il file deve essere il mezzo fondamentale per la persistenza dei dati.

C. Software Architecture

C.1 Class Diagram



C.1.2 Tracciabilità

LogicEngine

Operazione	Codici Requirements
buildTree()	CRA / INP
SumValues()	CSA / CAL / INP / FDB
editValues()	MDV / INP / FDB
editValues()	INP / FDB

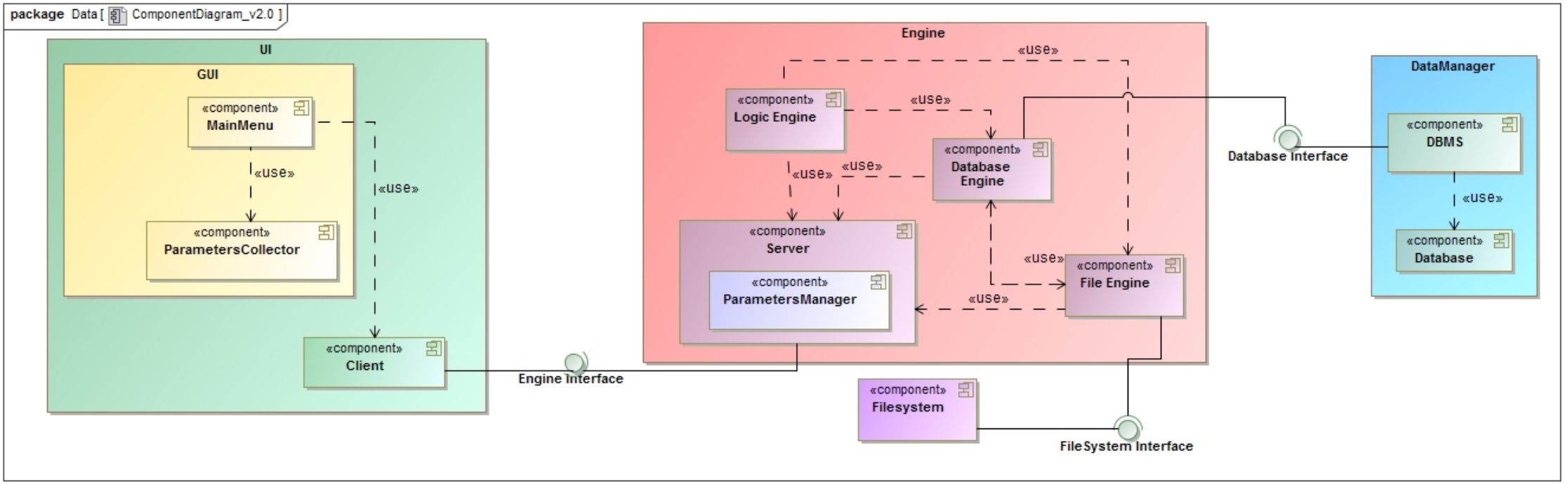
FileEngine

Operazione	Codici Requirements
generaFileAlbero()	ESF / CRA / FDB
generaAlberoInDB()	UPF / GDB

C.2 Component Diagram

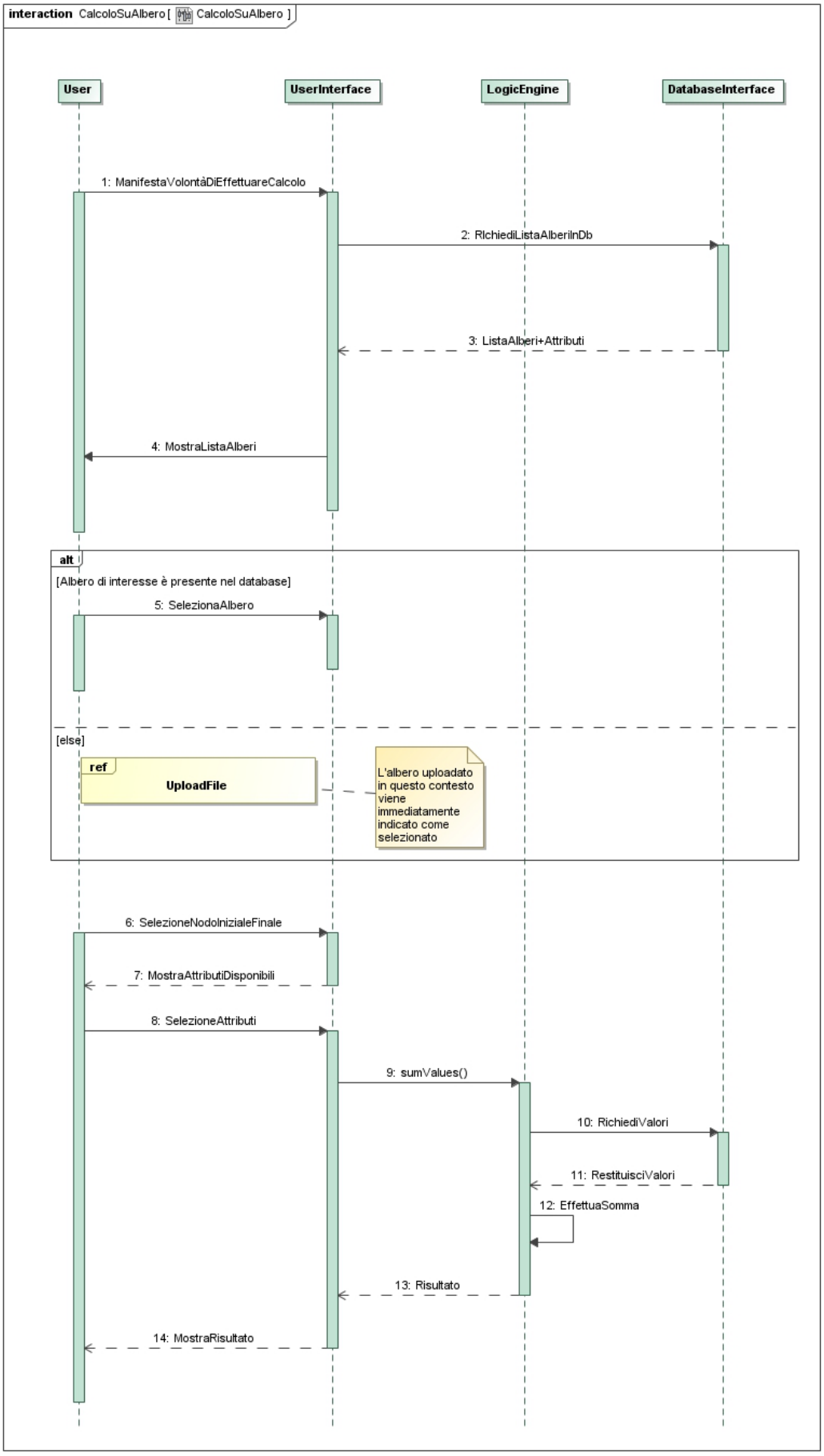
C.2.1 Documentazione Component Diagram

Il design del sistema prevede la separazione delle logiche operative (Engine), di interfaccia (UI) e di gestione dati (DataManager).
Il sottosistema UI permette di accedere ai servizi che il sistema come da requisito deve offrire. Nella GUI il menu' di selezione principale (MainMenu) si occuperà di far partire la comunicazione con il sistema, offrendo una raccolta di punti di accesso alle funzionalità messe a disposizione dall'Engine.
A seconda dell'operazione che si desidera eseguire, verrà chiesto all'utente di inserire dei parametri specifici; nella GUI questa operazione e' svolta per mezzo di ParametersCollector, che avrà la funzione di gestire i parametri da inviare all'Engine successivamente alla scelta dell'utente.
La comunicazione tra UI e il resto del sistema avviene grazie all'implementazione di un'architettura Client-Server che consente al sistema stesso di essere allocato in una posizione diversa dall'utente che lo utilizza via rete. In tale contesto, il componente Client, fulcro del sottosistema in questione, gestirà le scelte dell'utente, relative all'operazione da svolgere e ai parametri inseriti, e le indirizzerà al processo Server in ascolto eseguito dall'Engine.
Il sottosistema Engine e' il gestore della computazione; le operazioni, avviate dal componente Server in base alle informazioni ricevute, vengono suddivise tra LogicEngine, FileEngine DataBaseEngine. Il primo si occuperà delle procedure di creazione, modifica e calcolo su albero; il secondo gestirà le importazioni e le esportazioni tra database e filesystem; il terzo, di contro, costituirà il canale per effettuare le le query sul database, permettendo così un buon grado di portabilità.


package Data [ComponentDiagram_v2.0]
UI
GUI
«component»
MainMenu
«use»
«component»
ParametersCollector
«use»
«component»
Client
Engine Interface
Engine
«use»
«component»
Logic Engine
«use»
«component»
Database
Engine
«use»
«use»
«component»
Server
«component»
ParametersManager
«use»
«component»
File Engine
«use»
«component»
Filesystem
FileSystem Interface
Database Interface
DataManager
«component»
DBMS
«use»
«component»
Database

C.3 Dynamic Model: Sequence Diagram

C.3.1 Calcolo Su Albero



Riferito a use case [CRA].

1. Utente inserisce i parametri per il nuovo albero nella User Interface.

2. Utente conferma le proprie scelte una volta inseriti tutti i parametri.

3. User Interface inoltra la richiesta di creazione nuovo albero con i parametri raccolti al Logic Engine.

4. Il Logic Engine inivia un feedback alla User Interface per comunicare se i parametri forniti sono corretti.

5. Se ci sono problemi con i parametri la User Interface lo comunica all'utente e richiede un nuovo inserimento di parametri (che avviene al punto 6).

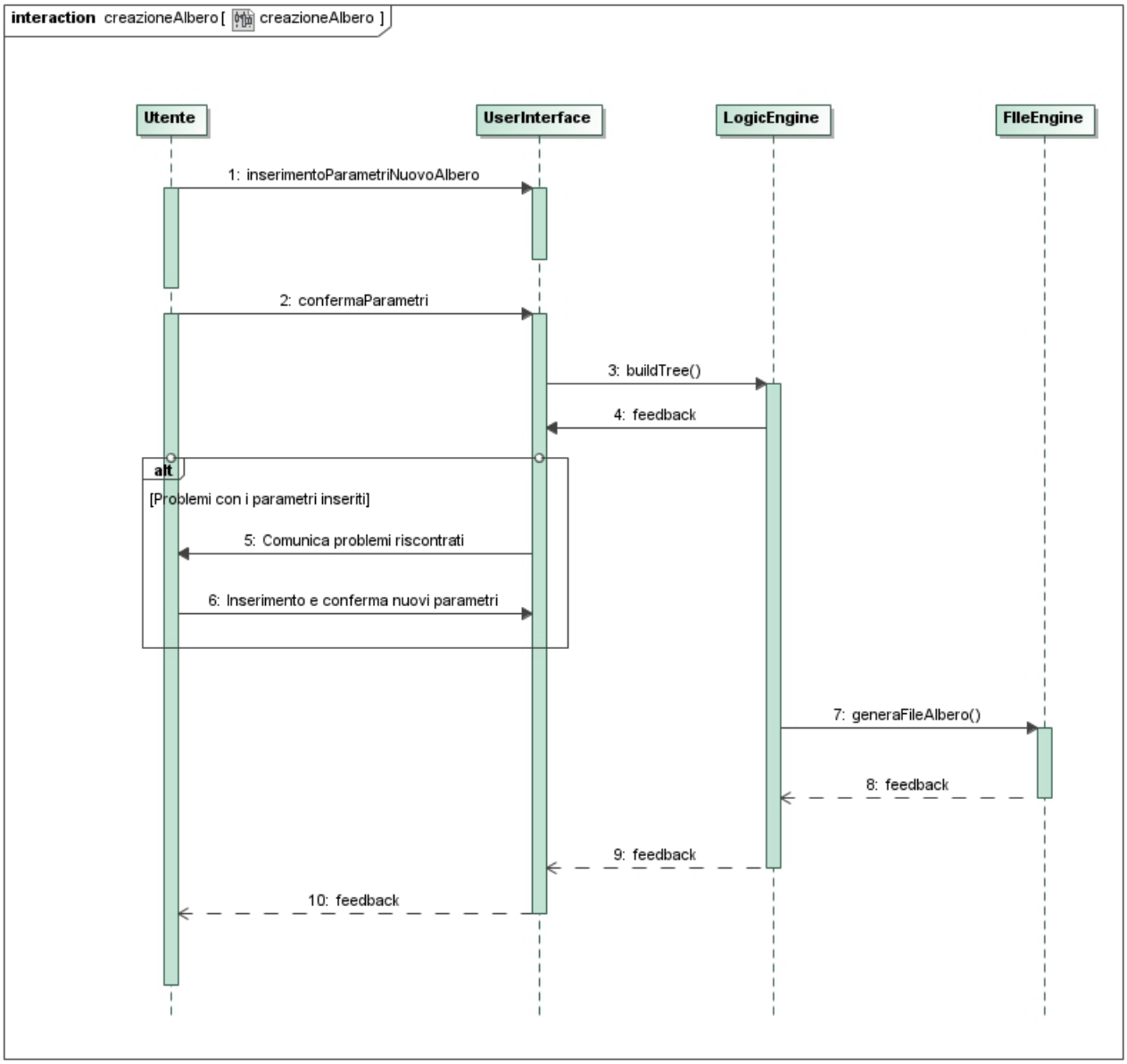
7. Il Logic Engine crea la struttura dati temporanea per il nuovo albero e la comunica al File Engine perchè si occupi di codificarla nel formato adatto ad essere scritto su file e di creare effettivamente il file.

8. Il File Engine comunica l'esito dell'operazione al Logic Engine.

9. Il Logic Engine comunica l'esito dell'operazione alla User Interface.

10. La User Interface mostra l'esito all'utente.

C.3.2 Calcolo su Albero



Riferito a use case [CSA]

1. Attraverso la User Interface l'utente seleziona la funzionalità che gli permette di effettuare calcoli su un albero.

2. La User Interface deve fornire all'utente la lista degli alberi disponibili. Per questo motivo invia una richiesta alla Database Interface.

3. La Database Interface esegue la query presso il dbms e restituisce alla User Interface la lista degli alberi disponibili corredata da strutture dati per rappresentare direttamente gli attributi di archi e nodi di ogni albero.

4. User Interface mostra la lista degli alberi disponibili all'utente.

5. L'utente sceglie l'albero su cui effettuare il calcolo. Se questo non è presente nella lista sceglie l'opzione di caricarne uno da file richiamando la procedura descritta per il caricamento di un albero

da file in db. In questo contesto il caricamento da file si conclude con la selezione automatica dell'albero appena caricato.

6. L'utente seleziona un nodo iniziale e finale dell'albero (indicandoli nell'interfaccia e non scegliendoli da una lista).

7. User Interface presenta all'utente la lista degli attributi disponibili per nodi e archi dell'albero selezionato.

8. L'utente seleziona gli attributi sui quali effettuare il calcolo.

9. La User Interface invia la richiesta di calcolo al Logic Engine con i parametri raccolti dall'utente.

10. Logic Engine richiede i valori per gli attributi specificati, per gli elementi specificati dell'albero specificato alla Database Interface.

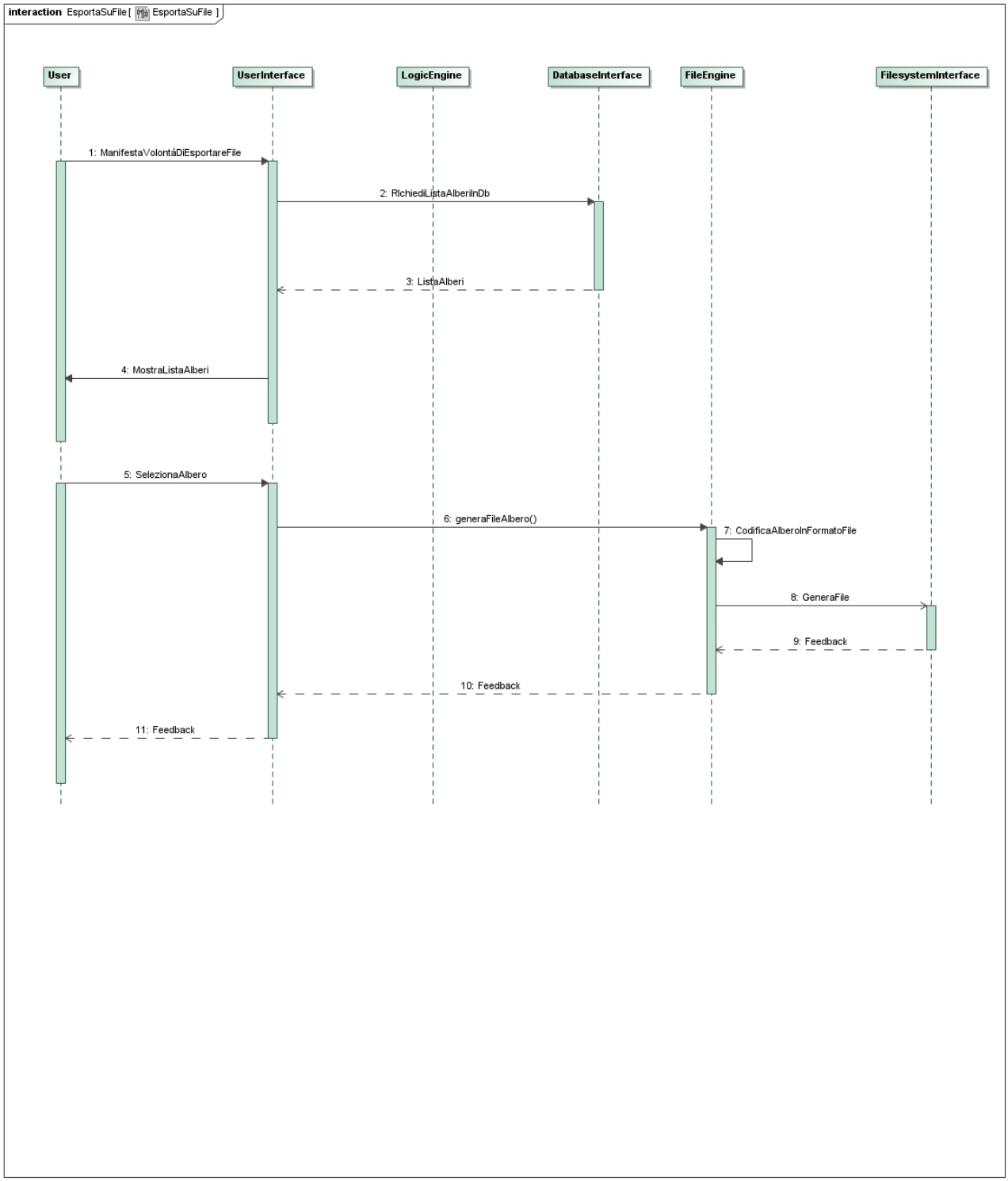
11. Logic Engine riceve le informazioni richieste.

12. Logic Engine effettua la somma.

13. Logic Engine comunica il risultato alla User Interface.

14. User Interface rende disponibile il risultato all'utente.

C.3.3 Esporta su File



Riferito a use case [ESF]

1. L'utente seleziona la funzionalità per esportare un albero dal database ad un file.

2. La User Interface richiede all'interfaccia con il database la lista degli alberi.

3. La User Interface riceve le informazioni richieste.

4. La User Interface mostra la lista degli alberi.

5. L'utente seleziona l'albero ed un nome file e path di destinazione del nuovo file.

6. La User Interface invia la richiesta di creazione del nuovo file al File Engine.

7. Il File Engine codifica l'albero selezionato nel formato da scrivere su file.

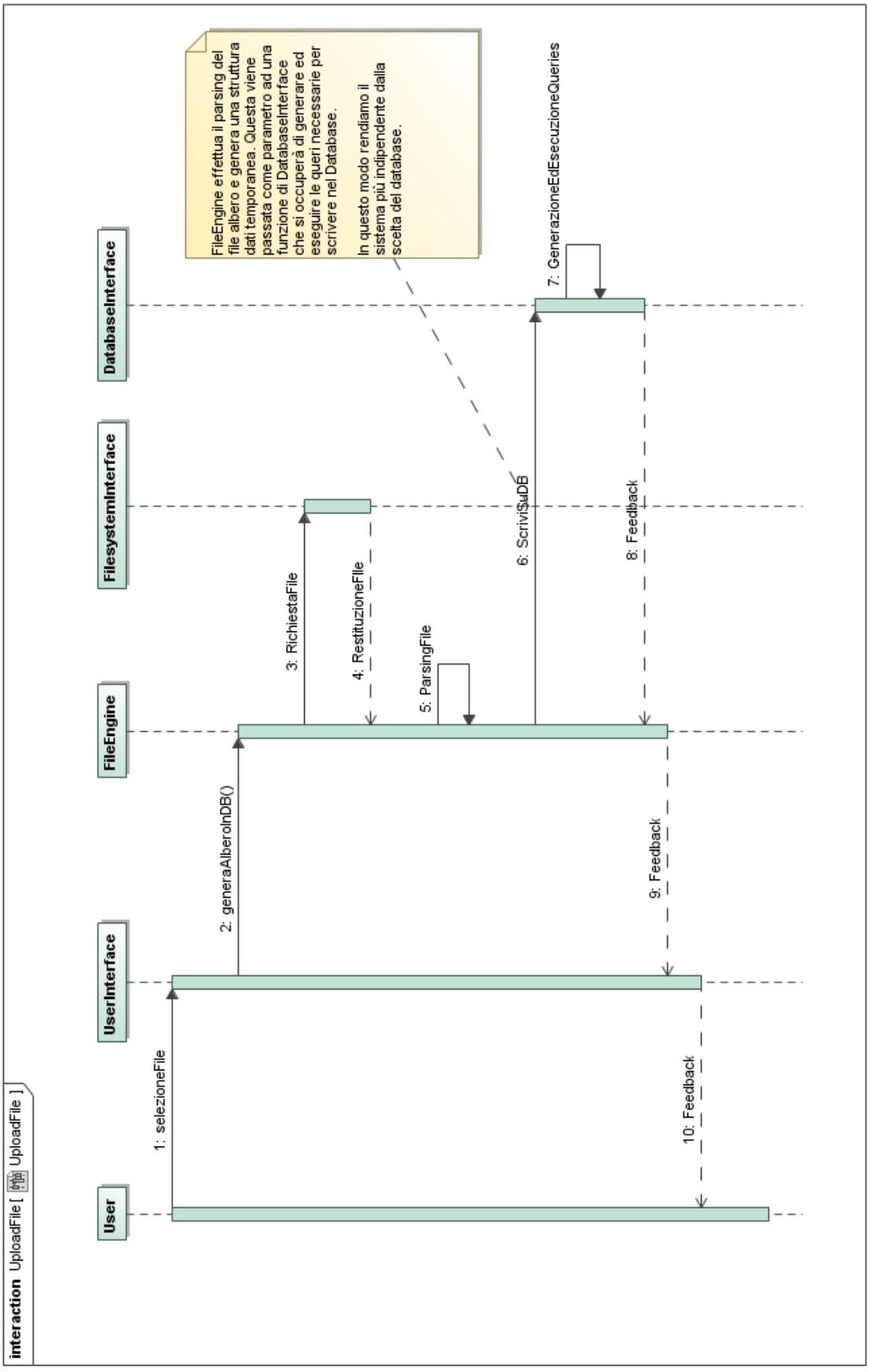
8. Il File Engine invia i dati all'interfaccia con il filesystem con l'istruzione di creare il nuovo file albero.

9. L'interfaccia con il filesystem invia un feedback al File Engine.

10. File Engine inoltra il feedback alla User Interface.

11. User Interface mostra l'esito all'utente.

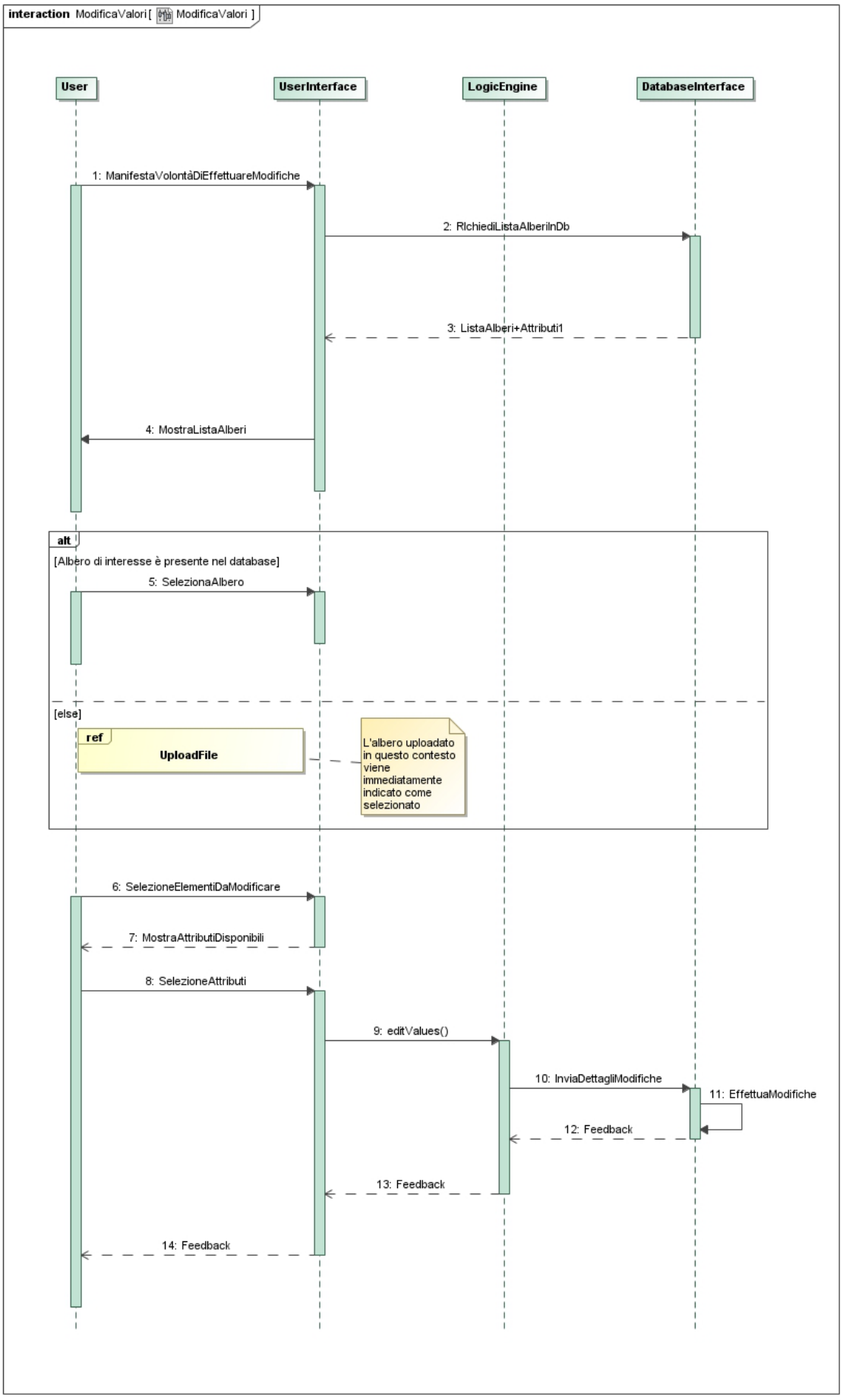
C.3.4 Upload File



Riferito a use case [UPF]

1. L'utente seleziona un file albero attraverso la User Interface.
2. La User Interface avvia la procedura del File Engine per il caricamento di un albero nel database.
3. Il File Engine richiede all'interfaccia con il filesystem di aprire il file specificato dall'utente.
4. Il File Engine riceve il riferimento al file aperto dall'interfaccia con il filesystem.
5. Il File Engine effettua il parsing del file e genera la struttura dati albero temporanea.
6. Il File Engine invia la struttura dati dell'albero all'interfaccia con il database che ...
7. ... si occupa a partire dalla struttura dati di generare ed eseguire le query sul database.
8. L'interfaccia con il database notifica il File Engine dell'esito dell'operazione.
9. File Engine comunica l'esito alla User Interface.
10. User Interface mostra l'esito dell'operazione all'utente.

C.3.5 Modifica Valori



Riferito a use case [MDV]

1. L'utente seleziona la funzionalità per modificare i valori di un albero.

2, 3, 4, 5, 6, 7. Il sistema consente all'utente di scegliere un albero con procedura analoga a quella adottata per l'operazione di Calcolo su Albero.

8. L'utente indica elementi da modificare.

9. La User Interface mostra all'utente la lista di attributi disponibili per la modifica nello specifico albero per i tipi di elementi specificati.

10. L'utente seleziona gli attributi inserendo al contempo i nuovi valori.

11. La User Interface richiama la procedura per la modifica dei valori dal Logic Engine con i parametri forniti dall'utente.

12. Il Logic Engine invia il comando di modifica all'interfaccia con il database.

13. L'interfaccia con il database effettua le modifiche.

14. Il Logic Engine riceve informazioni sull'esito dell'operazione e ...

15. ... le comunica alla User Interface ...

16. ... che a sua volta le renderà disponibili all'utente.

F. Design Decisions

F.1 Operazioni su Filesystem e Database

Codice [DDE-01]

Decisione adottata: È stato scelto di delegare tutte le operazioni inerenti a comunicazione con filesystem e database a due componenti apposite (indicate come nel Component Diagram come DatabaseEngine e File Engine).

Alternative scartate: l'alternativa alla decisione presa sarebbe stata far comunicare ogni componente che ne avesse bisogno direttamente con il database o il filesystem.

Motivazioni a sostegno: mantenere queste funzionalità in componenti appositi è un approccio ispirato al principio di "separation of concerns". In questo modo il sistema può essere modificato più rapidamente in futuro per operare con database diversi o set diversi di chiamate di sistema per l'interazione con il filesystem. Inoltre da un punto di vista progettuale le due componenti fungono da interfaccia tra sottosistemi differenti e ci permettono di ridurre il coupling tra di essi.

F.2 Risposte alle richieste di lista alberi

Codice [DDE-02]

Decisione adottata: quando viene richiesta al database la lista degli alberi disponibili, nella risposta viene inclusa anche la lista degli attributi disponibili per archi e nodi di ogni albero.

Alternative scartate: inizialmente la risposta comprendeva unicamente la lista degli alberi.

Motivazioni a sostegno: Questa risposta arricchita e' motivata dal fatto che l'alternativa sarebbe stata una ulteriore richiesta da User Interface a Database con il solo scopo di controllare gli attributi disponibili dopo la selezione dell'albero. Valutiamo che essendo gli attributi di archi e nodi di un albero in numero contenuto (infatti non si parla di valori su ogni nodo ed arco, ma soltanto degli attributi comuni che coinvolgono nodi e archi, stimati in un numero medio totale di circa 10 per albero) il sistema possa operare in maniera più efficiente grazie a questa scelta.

F.3 Individuazione di nodi ed archi

Codice [DDE-03]

Decisione adottata: è compito dell'utente inserire l'identificativo dei nodi per le operazioni in cui è necessario individuarne (e.g. per la somma di attributi su un determinato cammino è necessario scegliere un nodo di partenza ed uno di arrivo). Il sistema si occupa in seguito di controllare l'effettiva presenza nell'albero degli elementi indicati.

Alternative scartate: è stata scartata l'alternativa di mostrare all'utente una lista di tutti i nodi presenti in un determinato albero.

Motivazioni a sostegno: nelle discussioni con il committente è stato chiarito che il sistema che

stiamo progettando non mostra mai rappresentazioni complete dell'albero all'utente ed è stato lasciato intendere che saranno a disposizione dell'utente altri mezzi per consultare l'albero.

F.4 Esecuzione di somme

Codice [DDE-04]

Decisione adottata: la somma di valori di attributi di nodi/archi selezionati è un'operazione delegata all'Engine.

Alternative scartate: in una fase iniziale di progettazione era stata considerata la soluzione di far eseguire i calcoli al dbms.

Motivazioni a sostegno: seguendo l'alternativa scartata, il dbms si sarebbe dovuto occupare di restituire i risultati delle query ricevute e, nel caso di una richiesta di calcolo, si sarebbe dovuto occupare di selezionare i valori degli elementi coinvolti e successivamente sommarli, restituendo unicamente il risultato complessivo dell'operazione svolta. Questo approccio, pur apparendo semplice ed intuitivo, in nostra opinione non soddisfa appieno i requisiti non funzionali di manutensibilità e scalabilità; gli stessi che, al contrario, appaiono essere allineati con la scelta da noi infine adottata. Confrontando le due possibilità con degli scenari d'esempio, motiveremo il cambio di scelta.

Scenario 1. Il database è in grado di soddisfare con performance costanti un certo numero di richieste. Qualora fosse necessario servirne una quantità maggiore, essendo il database delegato al calcolo, l'unico modo per soddisfare questa impellenza implicherebbe un potenziamento dell'hardware sottostante il database stesso, o comunque un aumento delle risorse a disposizione di quest'ultimo. Questo tipo di scalabilità e' intrensicamente “verticale”, e possiede una flessibilità alquanto ridotta; l'impiego di soluzioni più sofisticate come l'uso di database distribuiti o ambienti paravirtualizzati e' in ogni caso piu' dispendioso, e al contempo rende notevolmente complessa l'infrastruttura di supporto al sistema. Vedendo il medesimo scenario con l'Engine arbitro delle procedure di calcolo, il database si occuperebbe delle sole funzioni relative allo storage (mantenendo dunque una separazione concettuale delle logiche), allegerendo notevolmente il carico di lavoro generato. Saranno, infatti, le macchine eseguenti l'Engine ad occuparsi delle somme, in modo locale. Quest'ultimo modello delinea una scalabilita' “orizzontale” ove, ad un numero di istanze concorrenti crescente parallelarmenet viene associato un quantitativo di risorse maggiore.

Scenario 2. L'azienda ha il bisogno di sostituire il database del sistema con un altro completamente o parzialmente incompatibile. E' necessario dunque riadeguare il codice del sistema che gestisce il database. Se il database fosse delegato al calcolo, oltre alla riscrittura delle singole query, sarebbe necessario riadattare anche le sotto-operazioni richieste per eseguire la somma. Questo incremento di lavoro tecnico potrebbe al contrario essere evitato se il calcolo fosse adibito all'Engine, limitando le modifiche alle sole componenti coinvolte nelle operazioni di interfaccia con il database. Questo approccio e' ulteriormente conveniente se si pensa che le parti sottoposte a revisione sono in numero minore, pertanto minore e' anche il rischio di incorrere in nuovi bug o in cattiva riprogettazione.

Quanto alle performance, abbiamo motivo di stimare paritarie le due modalità di esecuzione del

calcolo.

F.5 Formato dei file albero

Codice [DDE-05]

Decisione adottata: è stato deciso di impiegare come contenitore dei dati relativi agli alberi il formato EXI (Efficient XML Interchange [Standard]).

Alternative scartate: le altre alternative considerate sono state JSON e XML.

Motivazioni a sostegno: nelle pagine successive è fornita un'estesa documentazione relativa alle motivazioni. Segue un semplice riassunto di ciò che il nostro team ha evinto.

XML è un formato di scambio dati generico, adattabile a qualsiasi contenuto si voglia condividere. Per quanto sia appetibile la scelta di JSON per via dell'agevole formato, della miglior (e più efficiente) mappatura degli oggetti in linguaggi object-oriented, e delle performance di encoding e deconding in generale superiori ad XML per via della sua semplice modellazione dei dati, il nostro team ha attualmente deprecato la sua scelta per via di tre cause fondamentali:

1. **adattabilità**: XML è un formato di scambio dati generico, adattabile a qualsiasi sia il contenuto che si voglia condividere; in futuro potrebbe poter essere richiesto di ospitare negli alberi dati differenti da semplice testo o numeri, e in tal caso utilizzare XML fornirebbe un sostanziale vantaggio, in quanto non sarebbe necessario cambiare il formato del file (in accordo al principio della manutensibilità). Al contrario questo non potrà essere reso possibile tramite JSON, se non attraverso una fase di troubleshooting significativa.
2. **sicurezza**: non essendo il nostro team in grado di prevedere come i file albero vengano spostati al di fuori dei confini aziendali, e non potendo fare assunzioni limitino gli spostamenti degli stessi, usare XML costituisce una precauzione in più: è rinomatamente conosciuto che la decodifica di formati JSON contenenti programmi eseguibili può favorire la proliferazione di malware introdotti dall'esterno[4].
3. **estendibilità**: per quanto in molti casi XML e JSON siano al pari in quanto a questa tematica, in tale situazione XML è ancora lo strumento più adatto alle esigenze del committente in quanto non viene fornito uno strumento di visualizzazione dei dati interno al sistema. Essendo XML un linguaggio di markup, è possibile tramite un'estensione al sistema o per mezzo di applicativi di terze parti fornire una visualizzazione del contenuto degli alberi, qualora fosse necessaria. Il formato JSON non offre la medesima funzione con la stessa semplicità ed immediatezza, poichè per costruzione (non essendo un linguaggio di markup) non è in grado di identificare una descrizione presentativa dei contenuti.

Alla scelta di XML si sono successivamente accostate fasi di ricerca che ci hanno condotto a valutare EXI. Quest'ultimo è compatibile con XML nativo, è standardizzato e supportato dallo stesso consorzio di XML, e sta velocemente guadagnando terreno in specifici domini dove la flessibilità di XML è necessaria ma occorrono prestazioni superiori in quanto a encoding, decoding e compressione dei dati. EXI adempie tutti questi obiettivi, essendo estremamente performante (si

veda la documentazione alle pagine successive per ulteriori dettagli) e porgendo attenzione a limitare l'uso delle risorse quali cpu-overhead e memoria. Un file in formato EXI è facilmente convertibile in XML nativo.

Punti a favore di XML rispetto a JSON:

- JSON non provvede nessuna capacità di visualizzazione dal momento che non e' un linguaggio di markup.
- I documenti XML possono contenere ogni tipo di dato immaginabile – dai classici tipi come testo o numeri, passando per oggetti multimediali come file audio, fino a formati dinamici come applet Java o componenti ActiveX. Questo implica che XML è più estensibile rispetto a JSON, considerato il fatto che quest'ultimo è in grado di immagazzinare solo tipi standard come testo o numeri.
- La possibilità di estendere gli attributi dei dati memorizzati in un file XML consente una maggiore flessibilità rispetto a JSON.
- JSON non e' adatto per agire come un veicolatore di file audio, immagini, o altre tipologie di file binari. JSON è ottimizzato per i dati; a ragion di ciò, convogliare eseguibili in un sistema di scambio data-based può introdurre severi problemi di sicurezza.
- Il modello di struttura dati sottostante XML è perfettamente mappabile sulla tipologia di formato necessaria al sistema (XML utilizza uno schema ad albero, JSON impiega mappe o liste).
- XML ha uno standard molto maturo, è in uso da quasi il doppio del tempo rispetto a JSON; quest'ultimo è definito secondo un RFC proposto molto recente[1], sebbene sia già largamente diffuso (non ai livelli di XML, tuttavia).

Punti a favore di JSON rispetto ad XML:

- XML è document-oriented, JSON è data-oriented. JSON può essere mappato più facilmente con gli oggetti di molti linguaggi di programmazione object-oriented.
- JSON è generalmente ritenuto più semplice da leggere per gli umani rispetto ad XML, cosi come è anche considerato più semplice da scrivere per via della sintassi più accessibile.
- JSON è processato con maggiore facilità rispetto ad XML per via della sua struttura semplice ed omogenea.

Punti a favore di entrambi:

- JSON è un buon formato di scambio dati. XML ha uno scopo più generico, ed è un buon formato per condividere documenti.
- JSON e XML hanno lo stesso potenziale di interoperabilità.

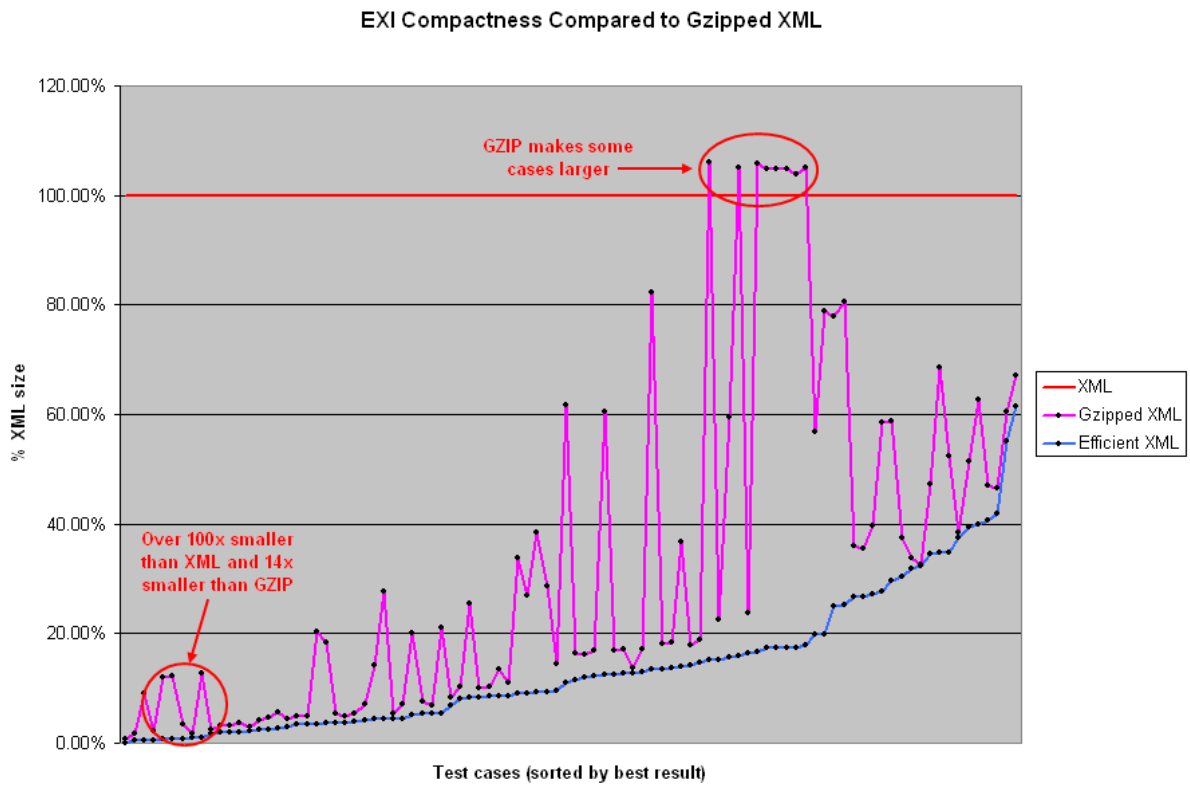
- E' disponibile un ampio range di software riutilizzabile per gestire sia XML che JSON, sia in versioni commerciali che gratuite ed open-source.
- Sia JSON che XML supportano l'internazionalizzazione tramite la codifica Unicode.
- Entrambi sono ben documentati ed hanno una vasta comunità di utenti e sviluppatori.
- I formati JSON e XML sono facilmente convertibili tra di loro

Il Formato EXI: la soluzione

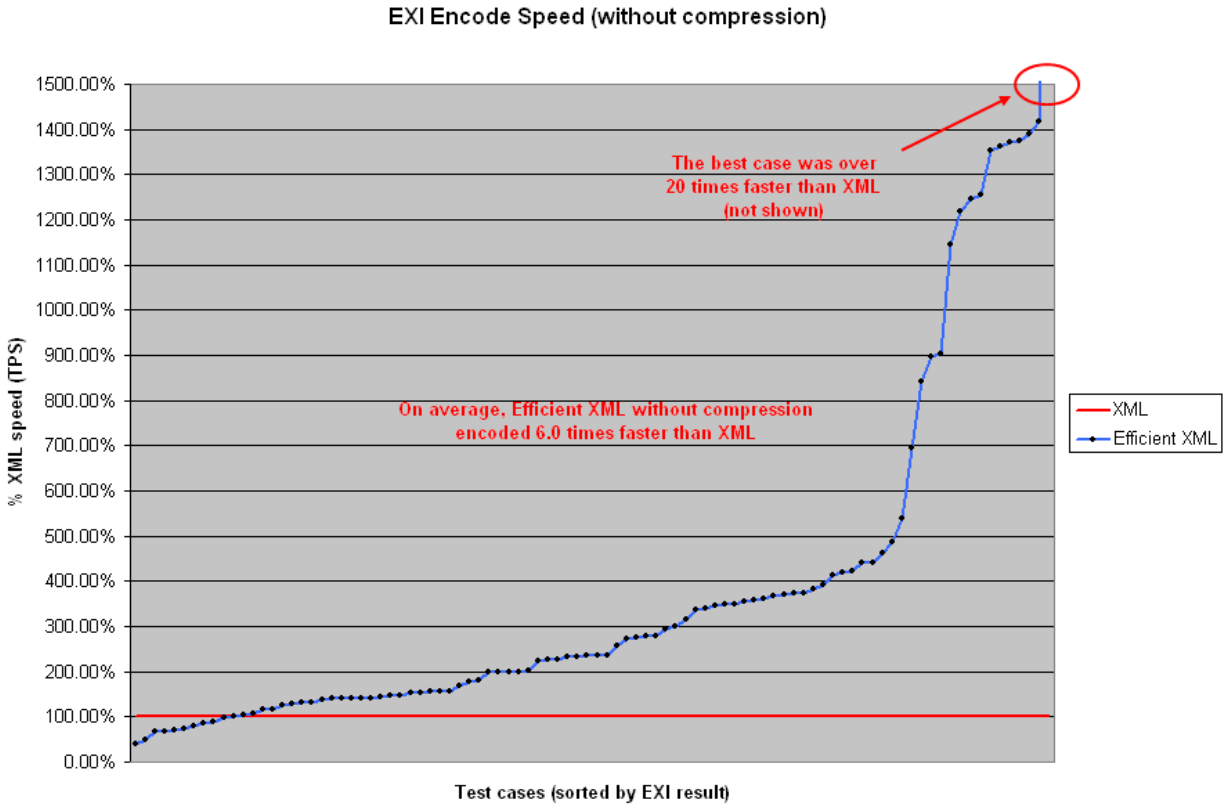
EXI è un formato molto compatto per la rappresentazione dell'information set di XML costruito per ottimizzare drasticamente le performance e ridurre il consumo di risorse simultaneamente. I principi di sviluppo di questo formato sono dettati[2] secondo le seguenti linee guida:

- generalità: massimizzare il numero di sistemi che possano comunicare tramite questo standard;
- minimalità: per raggiungere il più ampio set di sistemi, approcci semplici ed eleganti sono preferiti a quelli complessi e analitici.
- efficienza: EXI deve poter competer con formati binari ottimizzati manualmente in modo che possa essere utilizzato da applicazione che richiedono questo livello di efficienza.
- flessibilità: EXI deve poter gestire flessibilmente ed efficientemente documenti che contengono estensioni di schema arbitrari o che deviano dal loro stesso schema. I documenti che contengono deviazioni non devono far fallire la fase di encoding.
- interoperabilità: EXI si deve integrare al meglio con le esistenti tecnologie di XML, minimizzando (se non annullando) I cambiamenti richiesti da tali tecnologie. Deve essere compatibile con l'information set di XML, senza significativi sovrainsiemi o sottoinsiemi, in modo da mantenere interoperabilità con le specifiche XML attuali e future.

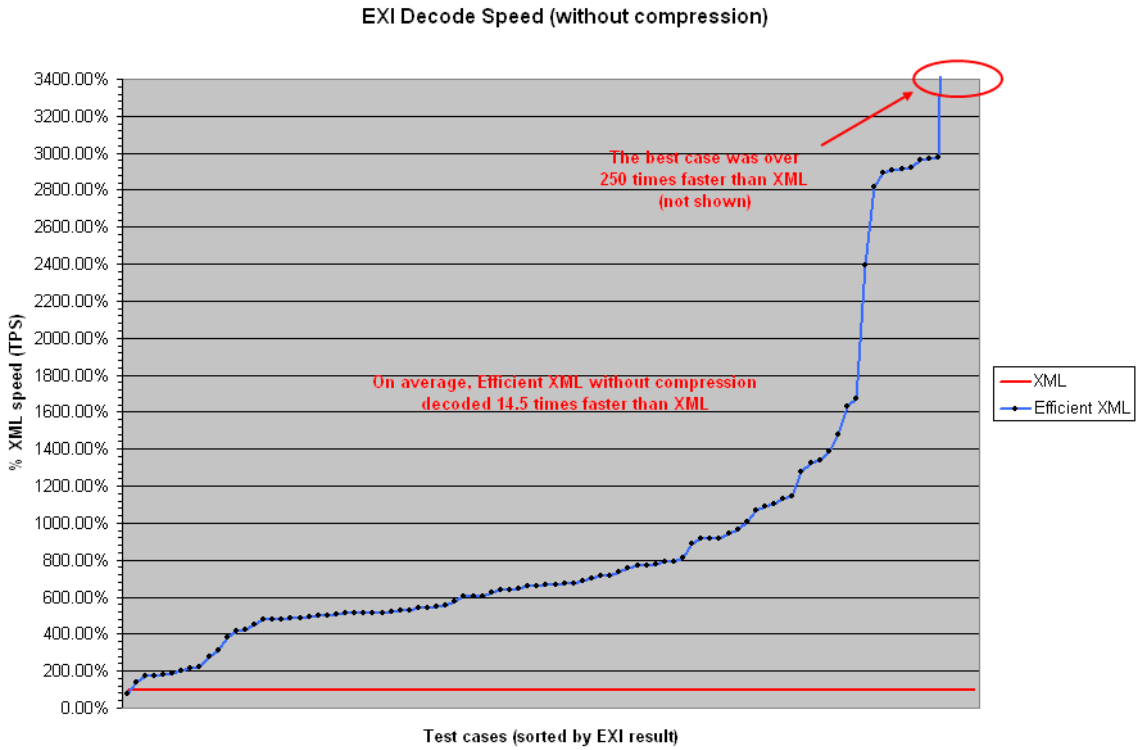
EXI eredita tutti i punti a favore di XML rispetto a JSON, inoltre aggiunge un focus molto incentrato sulle performance. Alcuni test ufficiali sono stati eseguiti, eccone qui riportate le infografiche[3]:



Descrizione [Figura 1]: La misurazione della compattezza rappresenta l'ammontare di compressione che un particolare formato raggiunge nella codifica di modelli di dati. In media EXI fornisce binari dalle 60 alle 80 volte più ridotti in dimensioni rispetto ad XML, e persino utilizzando un algoritmo di compressione (Deflate tramite Gzip è uno dei più usati), EXI riesce a mantenersi impareggiato (e di un margine piuttosto consistente).



Descrizione [Figura 2]: il grafico sopra esposto mostra la velocità di encoding di EXI in confronto ad un XML non compresso. In media, EXI e' circa 6 volte piu' rapido di una codifica in XML; nel caso migliore del test eseguito, esso si dimostra circa 21 volte più veloce.



Descrizione [Figura 3]: il grafico sopra riportato mostra come EXI sia superiore anche in fase di decoding rispetto ad XML. Nel caso medio, infatti, esso è circa 14.5 volte più veloce del concorrente, arrivando a toccare un picco del 257'000% (ossia circa 257 volte più performante).

Note

1. Lo standard è di Marzo 2014. Riferimento: https://tools.ietf.org/html/rfc7159.

2. Tali principi sono esplicitamente elencati entro lo stesso standard di EXI. Riferimento: http://www.w3.org/TR/exi/#principles.

3. I test di comparazione tra EXI ed XML sono stati eseguiti su una macchina con processore Intel Pentium4 3.0Ghz avente 1,5Gb di RAM. Il sistema operativo è Windows XP. L'implementazione di EXI utilizzata è EfficientXML, nella versione 4.0, che aderisce allo standard EXI 1.0. Ogni test è effettuato su 94 campioni differenti, in figura rappresentati da singolarmente da punti neri. Riferimento: http://www.w3.org/TR/exi-evaluation/#results.

4. Un esempio recente è il malware diffuso attraverso il file Manifest.json. Riferimento: http://blog.mitechmate.com/Manifest-json-Removal/.

Riferimenti

1. Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, Clemente Izurieta,“Comparison of JSON and XML Data Interchange Formats: A Case Study”, 2009.

http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf

2. JSON Official Site.

http://www.json.org/

3. Andy Newton, “Using JSON in IETF Protocols”, 2012.

http://www.internetsociety.org/articles/using-json-ietf-protocols

4. James Clark, “XML vs. the web”, 2010.

http://blog.jclark.com/2010/11/xml-vs-web_24.html

5. Tom Myer, “A really, really, really good introduction to XML”, 2005.

http://www.sitepoint.com/really-good-introduction-xml/

6. Rob Zazueta, “API Data Exchange: XML vs. JSON”, 2014.

https://www.mashery.com/blog/api-data-exchange-xml-vs-json

7. John Schneider, Takuki Kamiya, Daniel Peintner, Rumen Kyusakov, “Efficient XML Interchange (EXI) Format 1.0, 2nd Edition”, 2014.

www.w3.org/TR/exi/

F.6 Organizzazione dei dati da diversi alberi nel database

Codice[DDE-06]

Problematica: L'inserimento di diversi alberi su un database server ha presentato alcuni aspetti problematici che hanno richiesto un'analisi attenta. I più rilevanti sono i seguenti:

Motivazioni a sostegno: Inserire tutti gli elementi dei diversi alberi in tabelle comuni ci avrebbe portato a lavorare con tabelle di dimensione enorme con un costo eccessivo in termini di prestazioni. Gli elementi di alberi diversi (nodi ed archi) vengono generati con ID sequenziali a partire da 0 e tenerli in una tabella comune avrebbe creato ambiguità.

Decisione adottata: Creazione di un nuovo database per ogni albero salvato sul database server. In questo modo si ottengono tabelle di dimensioni minori e si tengono ben separate entità logicamente non connesse (i diversi alberi). (Vedere dettagli in Design Decisions).

F.7 Recuperare lista alberi con i dati di ognuno

Codice[DDE-07]

Decisione adottata: Mantenimento di un database per i metadata relativi agli alberi. È possibile con una sola query ottenere tutte le informazioni su tutti gli alberi.

Il costo di tale soluzione è esclusivamente una query di scrittura in più per ogni albero.

Problematica: Per il funzionamento del nostro sistema è necessario fornire all'utente in tempi rapidi una lista degli alberi disponibili sul database server completa delle informazioni ad essi legate (nome, tipo, lista attributi dei nodi, lista attributi degli archi).

Motivazioni a sostegno: Avendo optato per salvare ogni albero in un database separato il recupero di queste informazioni diventa difficoltoso.

F.8 Manutenibilità del database

Codice [DDE - 08]

Problematica: Un database per ogni albero comporta una riduzione della manutenibilità .

Decisione adottata: La soluzione immaginata è la creazione di un metodo apposito per modificare i database degli alberi in serie.

Se ci fosse un unico database basterebbe effettuare la modifica una volta, ma essendo la modifica della struttura un'operazione relativamente rara, il costo di effettuare la modifica in maniera automatica per ogni database albero già creato non avrebbe un peso eccessivo.

Considerata la presenza del database metadata è inoltre piuttosto semplice scrivere un metodo che apporti queste modifiche iterativamente su ogni albero.

Il metodo opera alla maniera seguente:

```
editAlberiDb(listaModifiche) {
    listaAlberi = Query su database metadata;
    per ogni albero in listaAlberi {
        per ogni modifica in listaModifiche {
            ChangeDatabase(albero);
            Query(“ALTER TABLE ...”);
        }
    }
}
```

F.9 Recuperare lista alberi con i dati di ognuno

Codice [DDE - 09]

Problematica: Avevamo inizialmente trascurato un aspetto importante relativo alle operazioni sugli alberi come selezione di un percorso sull'albero, recupero e modifica dei valori su di essi: queste operazioni prevedevano di lavorare caricando ogni volta l'albero dal database e non contemplavano la possibilità di lavorare con alberi già tenuti in memoria dal componente di business logic.

Decisione adottata: I metodi incaricati delle operazioni su indicate (editValues, getValues, getPath) sono stati resi polimorfi: possono ricevere alternativamente il nome di un albero (in questo caso l'albero viene caricato dal database appositemente per svolgere le operazioni) o una struttura dati albero (in questo caso le modifiche avvengono direttamente sulla struttura dati fornita e vengono

immediatamente salvati sul database relativo a quell'albero).

F.10 Serializzazine binaria di oggetti albero in XML scartata a favore di utilizzo di sistemi custom (assemblaggio e parsing di codice XML creato su misura per PPC).

Codice [DDE - 10]

Decisione adottata: costruzione di un assembler XML in sostituzione ad un serializzatore binario.
Alternative scartate: Altri serializzatori XML binari (e.g. XmlSerializer).
Motivazioni a sostegno: La scelta si e' dimostrata impegnativa; il problema e' emerso quando, dopo aver implementato parte delle strutture dati fondamentali di nostro impiego (eg. Albero, nodo, arco) con tipi Dictionary forniti dal framework .NET, siamo venuti a conoscenza della seguente problematica:

“The XmlSerializer cannot process classes implementing the IDictionary interface. This was partly due to schedule constraints and partly due to the fact that a hashtable does not have a counterpart in the XSD type system. The only solution is to implement a custom hashtable that does not implement the IDictionary interface.”

A questo punto, si e' ritenuto importante decidere se continuare ad impiegare le strutture dati precedentemente create ed utilizzare un serializzatore XML ad hoc, oppure utilizzarne di altre ed approfittare dei serializzatori XML disponibili all'interno del framework di Microsoft. A seguito di copiose ricerche, abbiamo preferito implementare il nostro serializzatore (i.e. il metodo assembleXML( )), soprattutto per guadagnare in termini di prestazioni e consumo di memoria. Questo, infatti, impiega classi dello stesso .NET come XmlWriter e XmlReader maggiormente indicate laddove le parsimonia di risorse è richiesta e si vogliono massimizzare le prestazioni in scrittura e lettura, soprattutto se le operazioni avvengono in memoria centrale, senza il bisogno di effettuare I/O.
Costruzione dell'assembler e del parser XML: vista la necessità di dover manipolare quantità ingenti di dati, è stata nostra attenzione rendere il più efficienti possibili (secondo le nostre conoscenze tecniche) le operazioni di costruzione del file XML e del suo parsing; la prima è affidata ad un'assembler che lavora one-pass: questo scansiona un oggetto albero e scrive in uno stream di memoria (i.e. MemoryStream) -sotto forma di XML- tutti i contenuti rilevanti ad una futura ricostruzione dell' albero stesso. In particolare, il file XML generato (compresso tramite EXI) conterrà due macro sezioni: un header, limitato ad una regione proporzionalmente piccola del file totale, la cui analisi sara' sufficiente a ricostruire un albero del tutto uguale all'originale a meno dei valori sugli attributi; una elementTable, contenente questi ultimi, che consentirà di ripristinare l'oggetto albero nella sua interezza. La struttura del file riflette esattamente la sequenza di azioni che il parser da noi creato deve eseguire (decompressione del file tramite EXI → lettura header → costruzione albero a meno dei valori su attributi → ripristino valori).

F.11 Architettura generale

Codice [DDE-11]

Decisione adottata: è stato deciso di impiegare un'architettura MVC attraverso un'implementazione client-server che fornisca una separazione concettuale tra logica di calcolo, persistenza dei dati e presentazione degli stessi.

Alternative scartate: è stata scartata un'architettura monolitica o varianti con componenti fortemente integrate.

Motivazioni a sostegno: Si possono riassumere le motivazioni a sostegno di questa scelta con una singola parola: flessibilità. Abbiamo ritenuto vantaggiosa la possibilità di accedere ad una User Interface che pemetta di specificare a quale Engine e quale DB collegarsi[1]; in questo modo è possibile, dalla stessa postazione, connettersi ad un DataBase A, prelevare un file albero trasferendolo sulla propria macchina, scollegarsi e riconnettersi ad un DataBase B ed ivi importarlo con semplicità. Inoltre l'Engine è in grado di supportare multiple connessioni sfruttando le capacità multithread forniti dall' OS tramitre i moderni processori multicore.

Nota: [1] Per accedere al DB si devono fornire le credenziali di accesso.

G. Soddisfacimento dei requisiti funzionali e non funzionali attraverso le decisioni di design

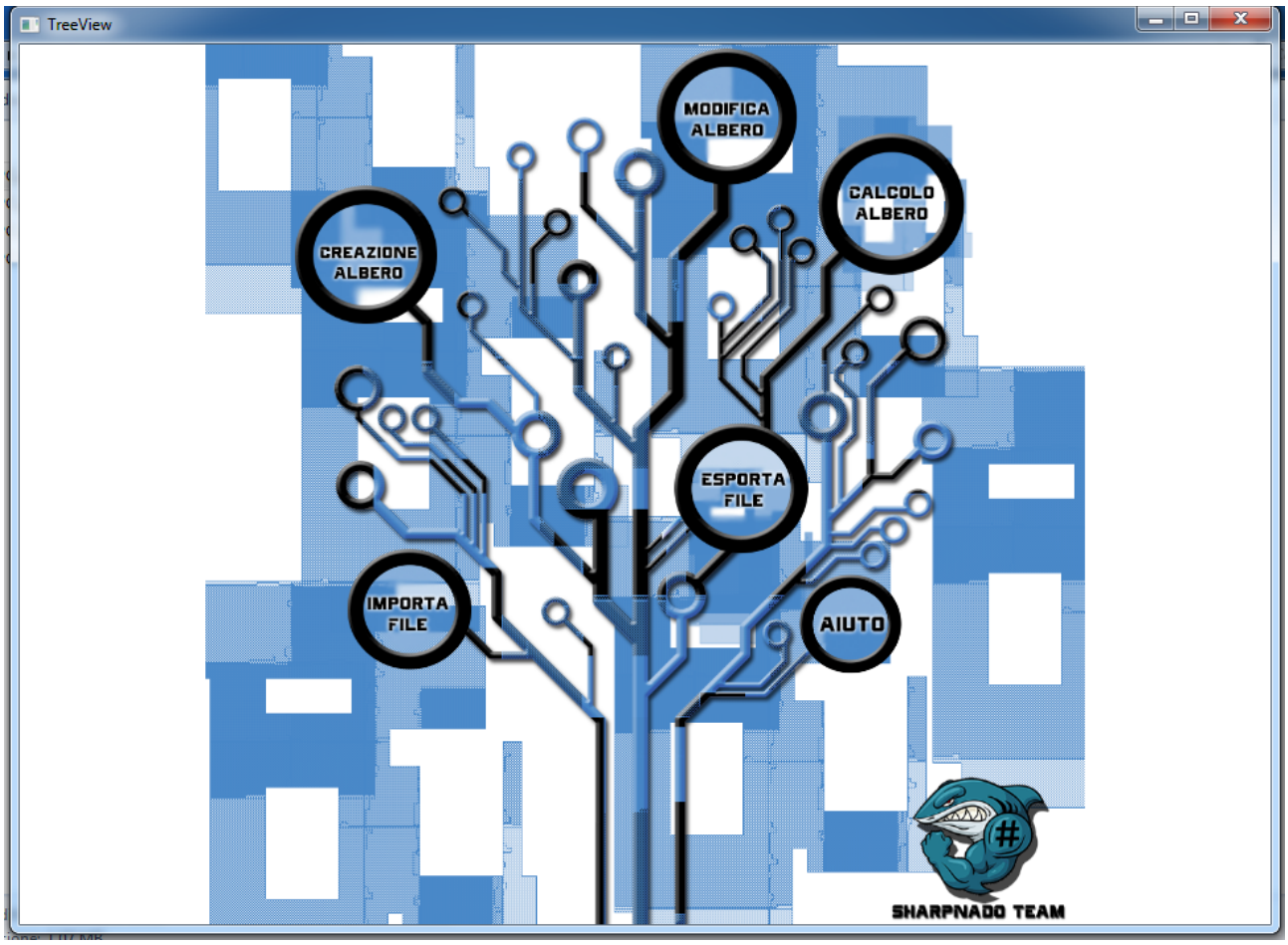
Nella documentazione delle design decisions (punto F di questo documento) sono presenti spiegazioni esaustive di come le scelte di design sono state ispirate dai requisiti non funzionali.

I requisiti funzionali sono stati tradotti direttamente con le funzionalità del sistema consultabili e ben documentate nella sezione C di questo documento.

Sistema Finale

User Interface

L'interfaccia utente e la logica di calcolo sono state separate per rispettare un'architettura Model View Control. Questo obiettivo è stato raggiunto implementando un server TCP all'interno dell' engine e un client TCP all'interno della GUI. La parte grafica è stata curata in modo da avere un look accattivante, mantenendo la desiderata user friendliness. Gli elementi visuali sono stati disegnati completamente a mano con l'ausilio di strumenti di grafica digitale. La User Interface (quindi, di conseguenza, anche la GUI) consentono di specificare engine e database ai quali collegarsi per fruire delle funzionalità del sistema, garantendo in tal modo la massima flessibilità.



Documentazione delle Finestre

Finestra “Start”:

I campi IP e Porta servono per specificare a quale Engine remoto connettersi. Inserire un IP e Porta validi. Le form DB IP ,Username e Password servono per l'autenticazione nel database.

Il bottone “Clear” serve per cancellare tutti i valori inseriti nelle form. Una volta inseriti tutti i campi in modo corretto,potrete cliccare sul bottone “Connect” per collegarvi all'engine.

Finestra “TreeView”

In questa finestra avete la possibilità di scegliere quale operazione eseguire cliccando sui bottoni posti sui rami dell'albero.

Finestra “Creazione Albero”

Questa è la finestra serve per la creazione dell'albero. In questa finestra possiamo definire tutte le caratteristiche che deve avere l'albero che vogliamo creare. La form “Nome” serve a specificare il nome dell'albero. Nella form SplitSize l'utente deve inserire un intero che definisce il numero di figli che ogni nodo deve avere. La form Depth serve a specificare quanti livelli dovrà avere l'albero. La parte sinistra della finestra,dopo le form, sarà riempita a run-time con tutti gli attributi già definiti nel database(Attribute Definition).Al click su un singolo attributo verrà aggiornata la

corrispondente parte destra della finestra. L'utente,attraverso le checkbox, può specificare se assegnare l'attributo ad un arco oppure ad un nodo .In base al dominio dell'attributo ,specificato in alto, l'utente può inserire una stringa o un range numerico.

Finestra “Modifica Albero”

Questa finestra serve a modificare i valori degli attributi per i nodi e gli archi compresi in un determinato path (Nodo Iniziale – Nodo Finale) che specifichiamo nella finestra. Dal menù a tendina “Seleziona Albero” possiamo selezionare uno tra tutti gli attributi presenti nel Database. Nelle form “Nodo Iniziale” - “Nodo Finale” l'utente dovrà inserire degli interi che corrispondono agli ID dei nodi. Se l'utente inserisce il numero 1 nella form “Nodo Iniziale” e 3 nella form “Nodo Finale” verranno modificati gli attributi dal nodo 1 al nodo 3.

Gli id dei nodi sono assegnati con una ricerca in profondità (depth-first search).

Appena selezionato l'albero vengono popolate a run-time le sezione Archi e Nodi con gli attributi corrispondenti agli archi e a i nodi. Da queste sezioni possiamo selezionare un attributo per assegnargli un nuovo valore in base al dominio specificato dal testo sovrastante.

Finestra “Calcolo Albero”

Questa finestra serve ad effettuare il calcolo sugli attributi per i nodi e gli archi compresi in un determinato path. Il calcolo che viene effettuato è una somma degli attributi selezionati.Dal menù a tendina “Seleziona Albero” specifichiamo l'albero su cui vogliamo effettuare il calcolo. Nelle form “Nodo Iniziale” e “Nodo Finale” l'utente deve specificare un intero che che corrisponde agli ID dei nodi.Se l'utente inserisce il numero 1 nella form “Nodo Iniziale” e 3 nella form “Nodo Finale” verrà effettuata la somma gli attributi dal nodo 1 al nodo 3.

Gli id dei nodi sono assegnati con una ricerca in profondità (depth-first search). Vengono popolate a run-time le sezioni Archi e Nodi. In queste sezioni sono contenuti gli attributi, l'utente può selezionarli e aggiungerli al calcolo attraverso il bottone “Aggiungi al calcolo”.

Finestra “Importa File”

Questa finestra serve per importare il file nel database. L'utente può scegliere il file nel file-system attraverso la finestra che si apre al click sul bottone scegli.

Finestra “Esporta File”

Questa finestra serve per esportare un albero sul file-system. L'utente può scegliere un albero nel menù a tendina “Seleziona Albero” e al click su esporta verrà creato un file “.exi”.

Business Logic

È stata dedicata una notevole quantità di tempo per valutare un'architettura software che rispondesse contemporaneamente sia ai requisiti di efficienza che di flessibilità. Per il componente LogicEngine si è adoperata un'interfaccia in modo tale che, qualora venisse richiesto di apportare delle modifiche tali da consentire operazioni su grafo, queste possano essere accolte con semplicità mantenendo le stesse signature attualmente impiegate e realizzando una classe grafo che implementi la suddetta interfaccia. L'altra componente, FileEngine, è ancora in fase di pieno sviluppo, in quanto

la sua implementazione ha presentato delle difficoltà non previste; la sua architettura, tuttavia, è stata modellata per gestire le stesse necessità di modifica di LogicEngine, secondo quanto detto sopra.

Database

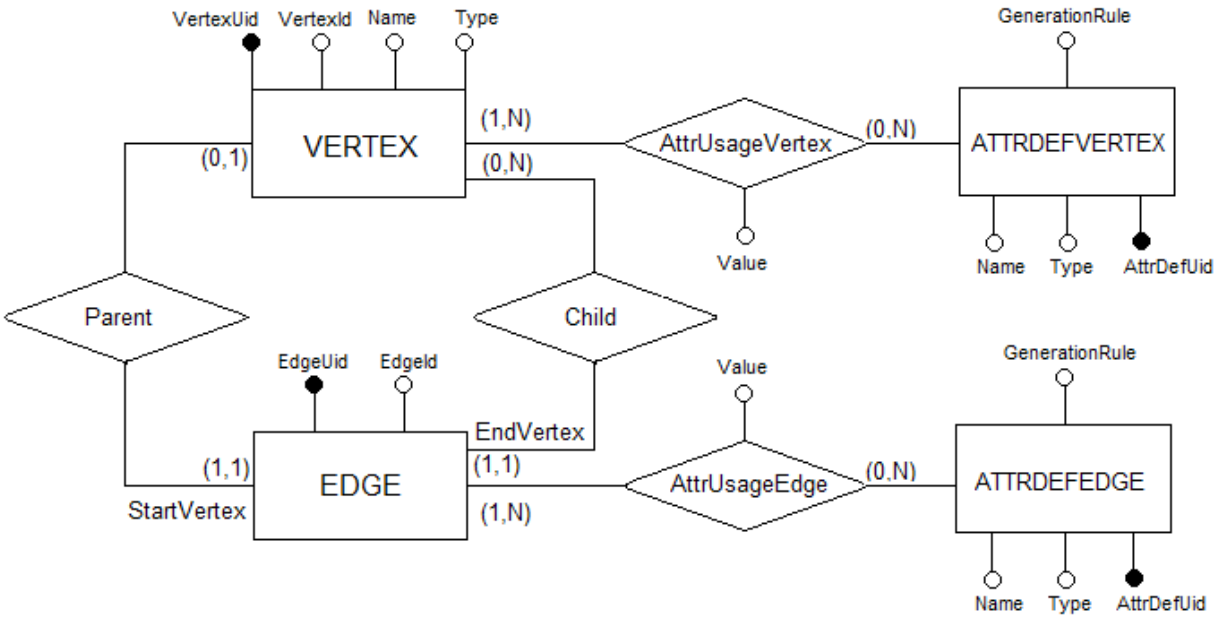
Decisions

Per il nostro sistema si è deciso di dedicare un database per ogni singolo albero. Avremo quindi all'interno del nostro server MS SQL più database i cui nomi corrispondono esattamente ai nomi degli alberi. Ai fini di tenere traccia di questi alberi si è inoltre deciso di creare un altro database di appoggio (InfoDB) per riportare le informazioni riguardanti gli alberi presenti nel server.

Naturalemente le info riguardanti un albero saranno presenti anche all'interno del singolo database relativo all'albero. Avremo quindi che i dati contenuti in questo database aggiuntivo saranno ridondanti ma la sua presenza semplifica notevolmente il lavoro durante le operazioni di recupero dei dati inerenti agli alberi. Per quanto riguarda il rischio di inconsistenza dei dati naturalmente derivante dalla ridondanza è stato considerato che non ne sarà problematica la gestione perchè la struttura di ogni albero viene toccata solo durante la creazione e mai modificata.

Sono riportati di seguito gli schemi ER, le descrizioni, i modelli relazionli e i codici relativi alle due tipologie di database.

Database



Analizzando la specifica e i requisiti funzionali sono state individuate cinque entità: **Vertex** che possiede gli attributi VertexUid (chiave primaria), VertexId, Name, Type.

Edge che possiede gli attributi EdgeUid (chiave primaria) e EdgeId.

AttrDefVertex e **AttrDefEdge** che possiedono gli attributi AttrDefUid (chiave primaria), Name, Type e GenerationRule (riporta se i valori dell'attributo al momento della creazione sono stati generati con una certa logica oppure random).

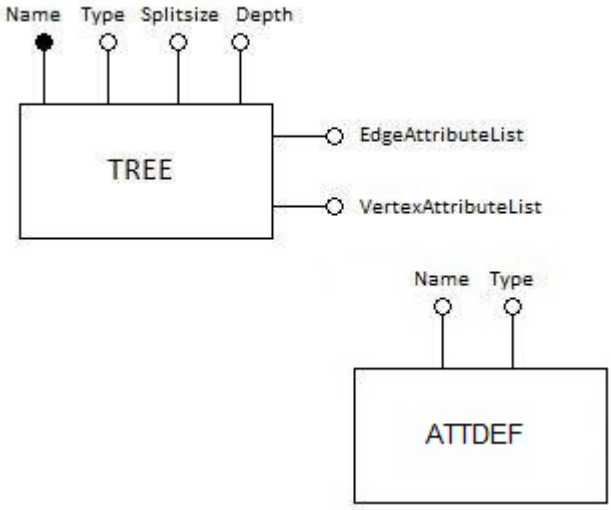
Queste entità sono messe in relazione tramite associazioni che sono:

Parent che mette in relazione Edge e Vertex con cardinalità (1,1) e (1,0) in quanto un Edge è connesso esattamente a un Vertex padre (StartVertex) mentre un Vertex può avere un Edge che lo connette a un Vertex padre (assente se nodo radice).

Child che mette ancora in relazione Edge e Vertex con cardinalità diverse di (1,1) e (0,N) in quanto un Edge è connesso esattamente a un Vertex figlio (EndVertex) mentre un Vertex può avere, a seoconda se si parli di un Vertice interno all'albero o di una foglia, da 0 a N archi con N che è pari alla SplitSize.

AttrUsageVertex e **AttrUsageEdge** che mettono in relazione rispettivamente Vertex-AttrDefVertex e Edge-AttrDefEdge con cardinalità di (1,N) e (0,N) per entrambe. Questo sta ad indicare che a un Vertex/Edge possono essere associati da 1 a N attributi, appartenenti a AttrDefVertex/AttrDefEdge, con valore= Value mentre un attributo di AttrDefVertex/AttrDefEdge può essere associato con valore= Value a 0 o a più Vertex/Edge.

Metadata: metadati sugli alberi presenti sul server



Descrizione

In seguito alla scelta fatta si è definita un'altra entità Tree interna a un DB separato rispetto a quelli degli alberi che ha il ruolo di descrivere gli alberi presenti all'interno del server (MS SQL Server).

Questa entità Tree ha come attributi Name (chiave primaria che identifica il nome del database relativo a un albero che è presente all'interno del server MS SQL), Type, SplitSize, Depth, EdgeAttributeList e VertexAttributeList.

Le due AttributeList avranno un formato particolare all'interno del database in quanto con una sola stringa descrivono l'AttrDefVertex o l'AttrDefEdge relative a un albero. Il formato utilizzato è di

questo tipo EdgeAttributeList = 'Attributo1.Tipo1.Rule1, Attributo2.Tipo2.Rule2,etcetc...'

in cui le virgole separano gli attributi e i punti separano le caratteristiche di questi attributi.

Il database metadata contiene anche la tabella AttDef che ospita la lista degli attributi che vengono resi disponibili per la scelta all'utente al momento della creazione dell'albero.

Modello Relazionale

TreeDB

Vertex(VertexUid, VertexId, Name, Type)

Edge(EdgeUid, EdgeId, StartVertexUid, EndVertexUid)

AttrUsageVertex(VertexUid, AttrDefUid, Value)

AttrUsageEdge(EdgeUid, AttrDefUid, Value)

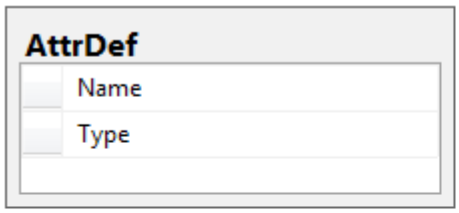
AttrDefVertex(AttrDefUid, Name, Type, GenerationRule)

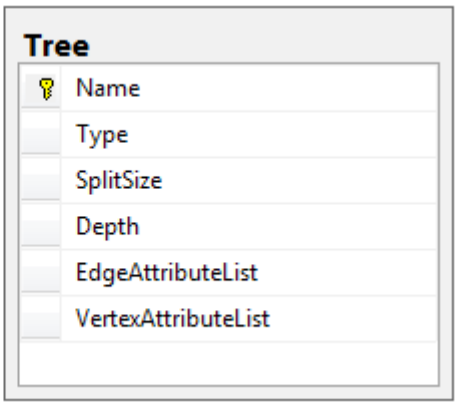
AttrDefEdge(AttrDefUid, Name Type, GenerationRule)

Metadata

Tree(Name, Type, SplitSize, Depth, EdgeAttributeList, VertexAttributeList)

AttDef(Name, Type)





Vincoli

Dal momento che con lo schema ER alcune codizioni non sono esprimibili ecco una serie di vincoli che devono essere rispettati.

- Solo il nodo root non ha archi entranti
- Solo i nodi foglia non hanno archi uscenti
- Da un nodo non foglia escono esattamente N archi (il nostro albero è completo)
- In un nodo non root entra esattamente un arco
- Un vertice non può avere due o più attributi uguali
- Un arco non può avere due o più attributi uguali
- Un attributo può essere presente solo una volta nella tabella AttrDef

DatabaseInterface

La classe DatabaseInterface rappresenta l'interfaccia al database a disposizione degli altri moduli. Oggetti di questa classe sono istanziati con i dati del database server al quale connettersi (scelti dall'utente e comunicati al sistema tramite la GUI).

Public DatabaseInterface (costruttore)

Il metodo costruttore accetta come parametri tre stringhe: indirizzo server, username, password. Se come indirizzo server si passa “local” i parametri username e password saranno ignorati e verrà instaurata una connessione al database server sulla macchina locale.

Int check

Restituisce 0 se le credenziali fornite durante l'istanziazione consentono di connettersi al database server.

Int editValues

Modifica i valori di attributi di nodi ed archi su un path tra due nodi indicati dall'utente su un albero indicato dall'utente. L'albero può essere indicato come stringa (nome dell'albero che verrà cercato nel database) o come struttura dati Albero (in questo caso le modifiche saranno effettuate sia sulla struttura dati che sul database).

Dictionary<string,string> getAttributeDefinition

Legge dal database metadata e restituisce la lista degli attributi disponibili in fase di creazione di un nuovo albero.

InfoAlbero[] getListaAlberi

Legge dal database metadata e restituisce la lista degli alberi presenti ognuno corredato dei suoi dati specifici.

List<string> getValues

Raccoglie i valori degli attributi indicati dall'utente per gli elementi su un path tra due nodi indicati dall'utente su un albero indicato dall'utente come stringa (nome albero) o come struttura dati Albero.

Elemento[] getPath

Cerca un percorso su un albero specificato dall'utente tra due nodi specificati dall'utente. Restituisce gli elementi che lo compongono (sia archi che nodi).

Int storeAlbero

Salva un albero sul database. Restituisce 0 in caso di successo.

Void addNodoRecur

Funzione ricorsiva ad uso interno per effettuare le query per l'inserimento di nodi ed archi nel database.

Albero getAlbero

Recupera un albero dal database.

Void setValoriCorretti

Metodo ad uso interno di supporto a getAlbero.

String dbCreationQuery

Metodo ad uso interno, restituisce la query per la creazione del database per salvare un albero settando il nome corretto per il nuovo database.

Codice

Tutto il codice del prototipo è disponibile nel repository all'indirizzo:
https://github.com/sharpnado/PPC4