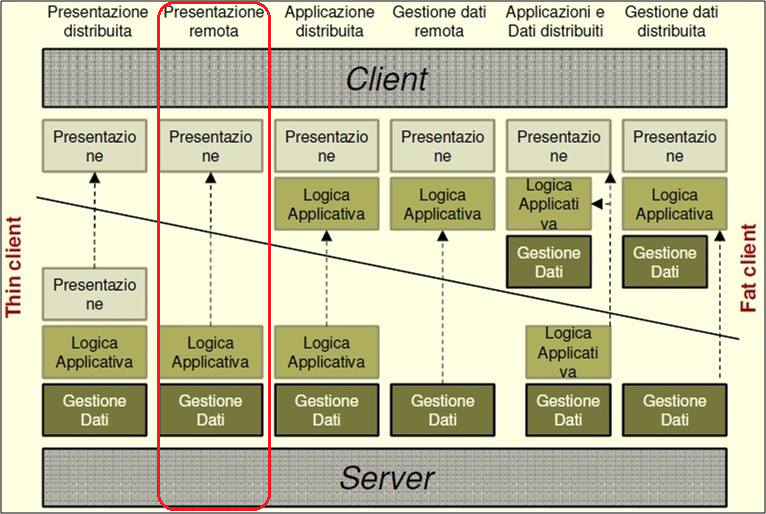
**Documento dell’Architettura software**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versione** | **Data** | **Descrizione** | **Autore** |
| Elaborazione 1 | 11/11/2013 | Revisione ed aggiunta di nuove regole | Luca finocchio  Vittoriano Muttillo  Daniele Leombruni  Stefano Dell’Osa |

L’architettura software dell’applicazione Time-Logging si basa su alcuni principi e pattern fondanti.

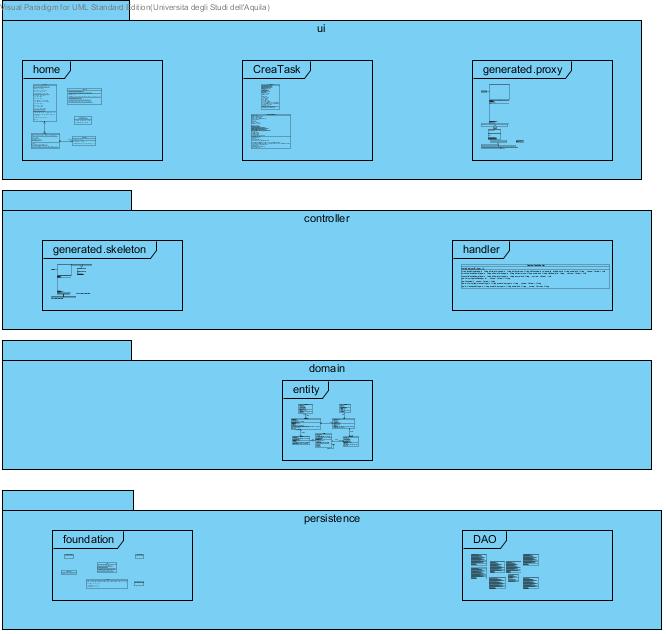
In particolare si è scelto di realizzare un’architettura distribuita e di strutturare il software implementando il pattern MVC.

Come illustrato nella Figura 1 , si è scelto di realizzare un client “leggero” ( solo interfaccia utente ) ed un server robusto.

Il dettaglio della struttura dell’architettura software si evince dall’immagine seguente, raffigurante il diagramma dei package:

Figura 1 - Architetture client server

**Client**: Le classi del subpackage **Swing**, demandate alla realizzazione dell’interfaccia, comunicano con il Server tramite le classi contenute nel subpackage **Proxy**



**Server**: Si compone di tre package, rispettivamente:

* **Controller** : riceve i comandi dall’utente attraverso la **view** ( UI sul Client) e li attua modificando lo stato del **Model** (Domain).
* **Domain** : fornisce gli strumenti per accedere ai dati utili all’applicazione.
* **Foundation :** fornisce strumenti per interfacciarsi con il DB tramite le classi del subpackage DAO.

Scendendo nel dettaglio della struttura dei package osserviamo la presenza dei subpackage **Skeleton** e **Proxy**, entrambi dovuti all’utilizzo della tecnologia software ICE per la realizzazione dell’architettura distribuita. Essi sono generati in maniera automatica una volta definito uno *slice*.

Ogni *slice* viene mappato sul **client** come classi **Proxy** e sul **Server** come classi **Skeleton**.

Un ulteriore vincolo tecnologico è imposto dall’utilizzo di un ORM, nello specifico Hibernate. Questo è facilmente riscontrabile dalla presenza del subpackage DAO all’interno del Foundation. Anche queste classi sono state generate in maniera automatica.

Per favorire l’espandibilità e la duttilità del software si è scelto di implementare un nuovo strato, posizionato concettualmente al di sopra del DAO, il cui compito è quello di fornire un’ ”interfaccia” al model indipendente dalla tecnologia utilizzata per gestire la persistenza.

**Progettazione guidata dalle responsabilità (RDD)**

Dall’analisi del caso d’uso espanso nella fase di Ideazione si è giunti alla realizzazione di 3 diagrammi SSD che mostrano l’interazione tra gli utenti coinvolti ed il sistema.

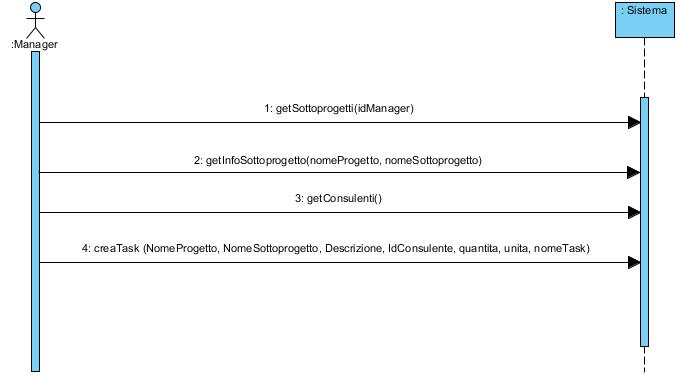


Figura 2 - SSD Gestisci Task -> Crea task



Figura 3 - SSD Gestisci Task -> modifica task

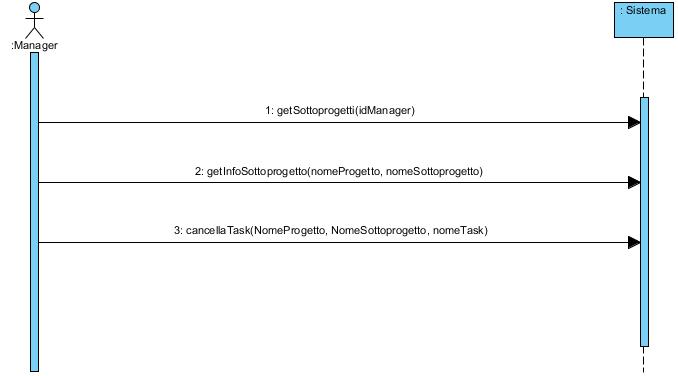


Figura 4 - SSD Gestisci Task -> elimina task

La prima criticità che ci siamo trovati ad affrontare riguarda l’individuazione dell’oggetto, oltre lo strato UI, il cui compito è quello di ricevere e coordinare le operazioni di sistema.

La soluzione è stata trovata applicando i principi del pattern GRASP Controller. Si è scelto quindi di creare una nuova classe, rappresentante il caso d’uso, chiamata gestisciTaskHandler (come suggerito dal pattern).

Successivamente si è passati alla creazione del Sequence Diagram contenente questo ed altri principi GRASP utili al contesto.

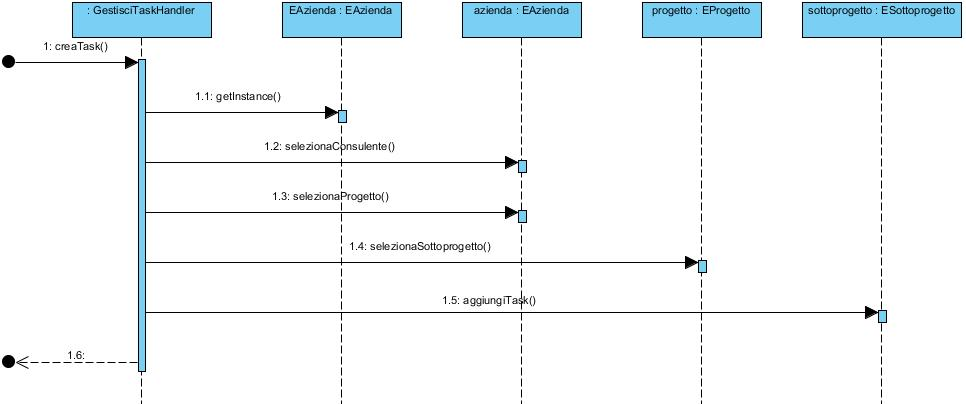


Figura 5 - SD creaTask()

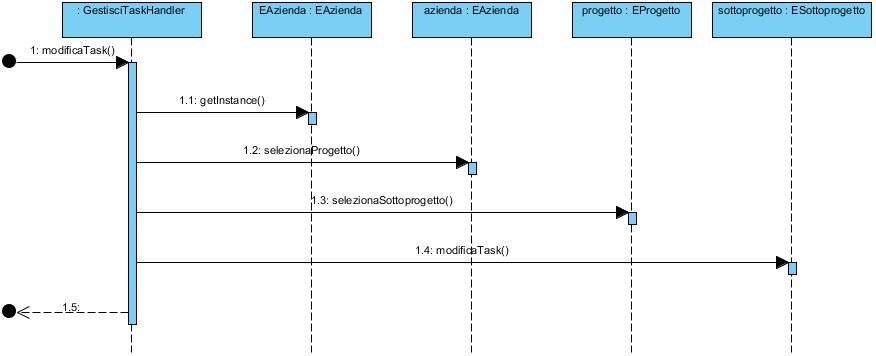


Figura 6 - SD modificaTask()

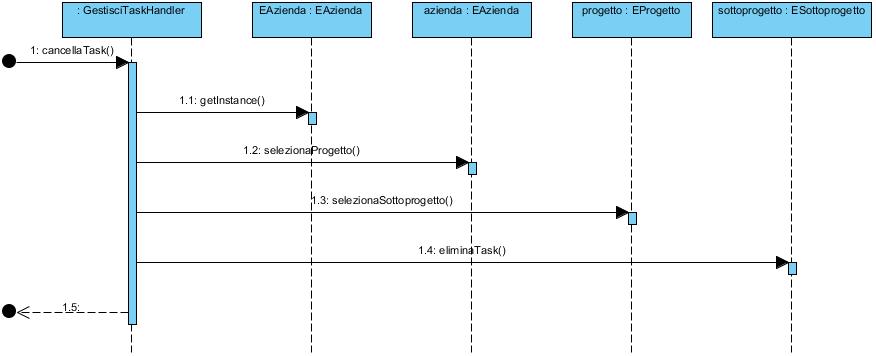


Figura 7 - SD cancellaTask()

Dalle immagini di cui sopra si evince come siano implementate le politiche del pattern GRASP Controller, in particolare come GestisciTaskHandler si fa carico di ricevere la richiesta da parte dell’utente e di gestirla.

Un’altra criticità è rappresentata dalla necessità di individuare l’oggetto in grado di recuperare il sotto-progetto al quale verrà aggiunto/modificato/eliminato il task. Pertanto, come suggerito dal pattern Information Expert, si è scelto di assegnare tale responsabilità alla classe che possiede tutte le informazioni necessarie, nel nostro caso la classe *progetto*. Tale classe, infatti, “contiene” al suo interno i sotto-progetti di interesse. Per lo stesso motivo si è assegnata alla classe Azienda la responsabilità di recuperare i progetti.

La responsabilità di creare/modificare/eliminare un nuovo oggetto Task è stata assegnata alla classe sotto-progetto in quanto, come specificato anche dal pattern GRASP *Creator*, essa “contiene” oggetti di tipo Task, li utilizza strettamente e si occupa della loro memorizzazione. Inizialmente si era pensato di assegnare questa responsabilità direttamente a GestisciTaskHandler che possiede i dati per l’inizializzazione del Task. La prima strada è stata preferita a quest’ultima per minimizzare il livello di accoppiamento (in questo modo GestisciTaskHandler non è accoppiato con Task).