Design model

Introduzione da pagina 20, 222, 604

Obiettivi:

- Definire object-oriented analysis e design
- Illustrare un esempio relativo

Conoscere un linguaggio di programmazione ad oggetti, come Java, è uno step necessario ma non sufficiente per la creazione di un sistema. Apprendere come "*pensare ad oggetti*" sicuramente non rappresenta una fase di totale semplicità, ma favorisce alla creazione di software robusto, ben progettato e manuntenibile, caratteristiche esterne ed interne che valorizzano al massimo lo sviluppo e l'implementazione di un progetto.

Tutti i passi che richiedono la metamorfosi di un *domain model* ad un *object-oriented model* non sono mai componenti di un procedimento meccanico, poichè, giunti a tale punto, è stata già effettuata un'analisi del problema e dei requisiti essenziali che dovranno essere poi riproposti all'interno della soluzione. Per cui si riconosce ciò che il sistema dovrebbe fare, ora occorre progettare un procedimento automatico che compia ogni attività voluta.

Il termine *design* sottolinea una *soluzione concettuale* che rientri appieno nei requisiti, ancora prima di una vera e propria implementazione; per cui conclusi diagrammi comportamentali, come *use case* oppure *activity diagram*, e diagrammi strutturali, come *class diagram*, la fase successiva consiste nella definizione degli oggetti software e al modo in cui essi interagiscono per poter soddisfare tutte le richieste.

Modello decisionale

Spesso è adottato un modello decisionale ispirato alle specifiche del framework *Unified Process*. L'elenco funzionale seguente rappresenta un esempio a cui ispirarsi, una semplificazione, dato che per progetti sempre più complessi e dettagliati il modello potrebbe racchiudere aspetti sempre più specifici e sottili.

Un tipico approccio risulta:

- Definizione dei *casi d'uso*. La progettazione di sistemi software nasce dall'esigenza di certi requisiti funzionali, i quali possono essere descritti sintatticamente mediante il *diagramma dei casi d'uso*, rispettando tutte le condizioni del contesto adottato.
- Definizione dei *diagrammi di interazione*. Il passaggio da una qualsiasi modellazione al *Object-Oriented design*, avviene per formalizzare oggetti software e le loro interazioni. Una soluzione spesso adottata consiste nell'uso del *sequence diagram*, capace di illustrare come gli oggetti interagiscono per mezzo di segnali, articolati in molteplici *lifelines*. Come da regola del *framework*, un diagramma di interazioni deve essere adeguato per ogni *use case* del punto precedente.
- Definizione del *diagramma delle classi*. Stabilito il modello che rappresenti una visualizzazione dinamica delle differenti interazioni, potrebbe essere utile ai fini implementativi

analizzare un approccio statico mediante singole *classi* che caratterizzino gli oggetti. Perciò è adoperato il *diagramma delle classi*, capace di descrivere azioni e attributi inerenti al sistema.

- Dividere il sistema in *interface layer* e in *domain layer*. Questo step rescinde chiaramente il lato amministratore dal lato client, attribuendo differenti finalità rispetto all'utente che si interfacci con il sistema. Le ragioni che spingono tale suddivisione sono relative ad azioni di manutenzione e di modifica al sistema software, in cui lo sviluppatore abbia piena libertà di poter apportare cambiamenti, senza che incidi sull'usufruibilità dell'utente finale.

Esempio