SWE-B: Leggere tra le righe

Eduard Bicego

23-05-2016

Contents

List of Figures

1 Diagrammi delle classi

1.1 Iterator Pattern

"sequenza di amminoacidi"

Attenzione, quando si descrive una collezione, sequenza e insieme è richiesto l'inserimento dell'**ITERATOR**. Evita la tentazione di usare Collection Java, l'implementazione del pattern è uguale e ti farà guadagnare più punti.

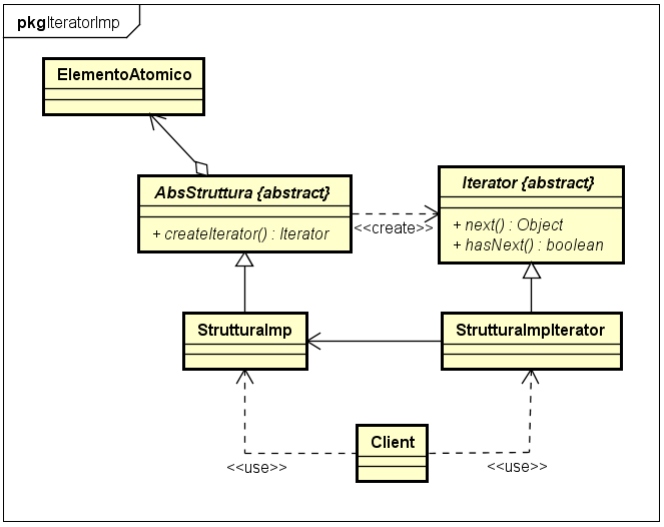

Figure 1: Iterator

1.2 Strategy e Template Pattern

> *"Il sistema, successivamente, utilizza un algoritmo di ricerca su database esterni per individuare la proteina codificata dagli amminoacidi."*

In generale quando c'è la parola **ALGORITMO** o c'è da utilizzare lo **STRATEGY** o il **TEMPLATE**.

> *"Nella versione attuale del software l'algoritmo è ancora poco efficiente ed utilizza una ricerca lineare sulla lista degli amminoacidi, iterando su di essa in modo sequenziale. Poiché gli sviluppatori hanno in progetto di rendere più efficiente tale processo, hanno modellato il sistema in modo tale da poterlo facilmente estendere con nuovi algoritmi."*

Quando si parla di *ALGORITMO* ed è specificato che tale *ALGORITMO* verrà esteso con *NUOVI ALGORITMI* e reso più efficiente si parla di **STRATEGY**. Anche il **TEMPLATE** avrebbe senso ma lo **STRATEGY** è più semplice ed accettato.

> *"È possibile scegliere fra tre tipologie di giocatori guidati dal computer, ad ognuno dei quali corrisponde un modello di intelligenza artificiale via via migliore."*

Intelligenza artificiale, algoritmi che cambiando sono via via migliori. Sempre usare lo **STRATEGY** in questi casi. "guidati dal computer" in questo caso esplicita chi ha la relazione con la gerarchia.

> *"Sulla base del formato del libro, che può essere AZW3, PDF, MOBI, il sistema operativo seleziona un algoritmo differente di resa a video (impaginazione) che determina la prossima pagina da leggere."*

Altro caso, quando si parla di 'formati' e si elenca una lista abbiamo ancora lo **STRATEGY**. In questo caso si specifica anche chi utilizza l'astrazione della gerarchia dello **STRATEGY**, ossia il sistema (il **Controller**).

> *"Le richieste HTTP possono essere codificate sia in XML sia in JSON"*

Necessario utilizzare un pattern **STRATEGY**, in questo caso per il Parser (codificazione) dei due formati.

Spesso la relazione con la superclasse della gerarchia dello **STRATEGY** e scritta nel testo.

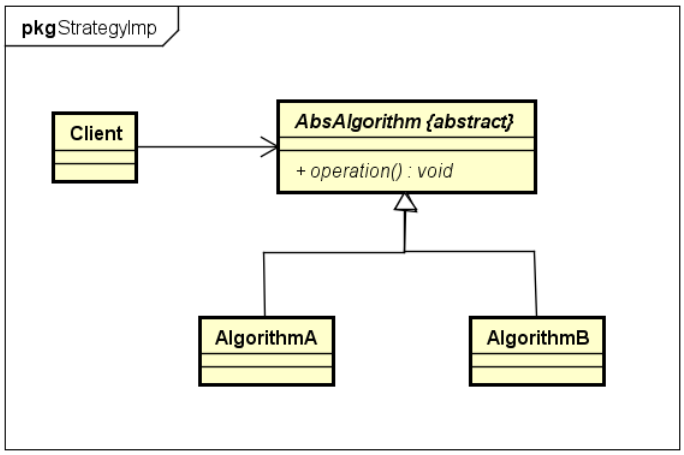

Figure 2: Strategy + client

> *"La componente che visualizza le immagini sullo schermo, applica una funzione di rendering che differisce solo in parte sulla base del formato, ma che possiede numerose parti in comune fra le immagini in MP4 e in MVK"*

Quando c'è la terminologia *'in parte'* o *'parti'* si riferisce chiaramente all'uso di un **TEMPLATE**.

> *"La trasformazione degli oggetti in istruzioni per le testine avviene utilizzando un apposito componente. Parte delle istruzioni sono comuni per entrambi i tipi di dato, parte è invece dipendente dal tipo di dato stesso."*

Ancora utilizzo di *'parte'* per suggerire l'utilizzo del pattern TEMPLATE.

1.3 Model View Controller e Observer Pattern

Ogni volta che si riferisce a SISTEMA o simili (SISTEMA OPERATIVO) quello è il CONTROLLER del pattern MVC.

> *"utilizzando un'architettura che aderisca al pattern MVC"*

Il pattern MVC è sempre richiesto espressamente come architettura. Nota che se non è richiesto significa che l'applicazione non dispone di interfaccia grafica, viceversa se è richiesta un'interfaccia grafica usare sempre l'MVC. Attenzione l'uso dell'MVC richiede sempre di utilizzare il pattern OBSERVER e la distinzione di tre entità:

- **View** (il subject)
- **Controller** o FrontController (l'observer, spesso identificato come il SISTEMA OPERATIVO o SISTEMA, spesso questa istanza può essere un SIGNLETON)
- **Model** (riferimento all'interno del Controller, spesso presenta un'interfaccia per disaccoppiarsi con il Controller)

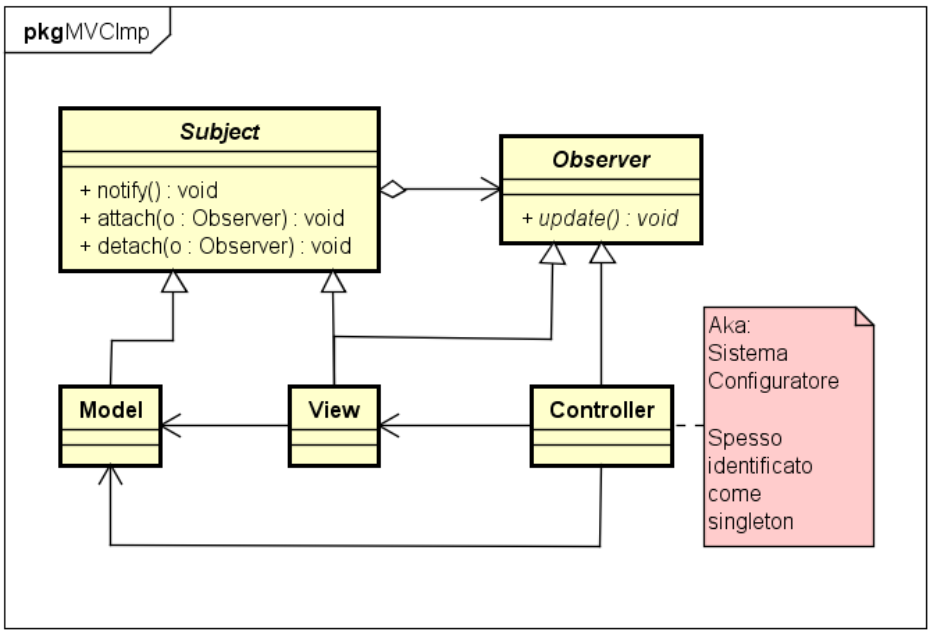

Figure 3: MVC - Observer - Singleton

> *"Il gioco deve disporre di un'interfaccia grafica basilare, che permetta all'utente di scegliere le proprie mosse."*

Quando specificato che l'interfaccia è interattiva con l'utente usare l'**OBSERVER** con MVC, se non è solo l'utente a interagire, ma anche il sistema, utilizzare l'**OBSERVER** in entrambi i casi: **View-Controller** e **Model-View** (subject-observer)

"Il configuratore fornisce una UI interattiva, ma semplice all'utente."

Il **CONFIGURATORE** è un'altra figura che rappresenta il **Controller**, solitamente fornisce qualcosa.

1.4 Polimorfismo

"Ogni giocatore può essere guidato da un utente o dal computer."

Quando si specifica che una probabile tipo di oggetto può essere di diverse tipologie usare il POLIMORFISMO.

> *"Amazon ha prodotto molte versioni del sue reader, attualmente ne esistono due: kindle e kindle paperwhite. Entrambi usano il medesimo sistema operativo."*

Attenzione, mentre qui sembrerebbe necessario il POLIMORFISMO la seconda frase dice di evitarlo poiché il SISTEMA è il medesimo, ciò significa che la differenza non è a livello di Business logic per cui il SISTEMA (il Controller) sarà unico.

Nota: quando si hanno dubbi sull'uso di un'interfaccia e una classe astratta o classe, optare per la prima se si è sicuri che tale interfaccia non utilizzi altre classi, usare una classe concreta per tutti gli altri casi anche se fare una gerarchia con classi concrete è pura follia. Fare ciò per evitare la presenza di errori.

"Esistono due tipi di frame: quello di HD e quello in UHD"

In questo caso rispetto al **POLIMORFISMO** si preferisce utilizzare un **ENUM**.

1.5 Abstract Factory Pattern

> *"Amazon ha però voluto distinguere alcuni tratti delle UI di conseguenza pulsanti, slider e liste di articoli hanno un look-and-feel differente nei due e-reader."*

Questo sta a significare che si ha una famiglia di prodotti diversi con tipologie diverse, è il caso di utilizzare un **ABSTRACT FACTORY** per ogni prodotto così da disaccoppiare i tratti grafici (item grafici) implementati dalla **View**. Nel caso presentato le due tipologie di item sono quelle dei due tipi di Kindle.

> *"In particolare è richiesto loro di progettare una componente, ossia pulsanti. Questi possono essere di tre tipologie, ossi a pulsanti semplici, situati in una barra di navigazione o in una scheda (tab). L'aspetto di questi pulsanti deve essere omogeneo tra loro, ossia afferire alla stessa famiglia di oggetti"*

Altro esempio in cui si necessita l'utilizzo di un **ABSTRACT FACTORY**, si riconosce per la presenza della parola 'famiglia' e 'tipologie' e la necessità di utilizzare **POLIMORFISMO** e disaccoppiamento delle componenti. Attenzione che per utilizzare una **ABSTRACT FACTORY** c'è la necessità di definire una **FACTORY** concreta, il testo la, o le se sono più di 1, suggerisce sempre.

> *"I modelli per cui la casa fornisce la configurazione online sono i propri modelli di punta, ossia il modello A ed il modello B."*

Anche qui si ha una famiglia di prodotti: A e B, per ogni tipologia di accessorio si utilizza una gerarchia differenziata per il modello, in poche parole **ABSTRACT FACTORY**.

1.6 Decorator Pattern

> *"La configurazione riguarda gli aspetti più comuni dell'auto, ossia i cerchi (in lega e non) . . . "*

Ogni qual volta si ha un particolare oggetto con una estensione bisogna pensare al **DECORATOR**. Dalle soluzioni però non sempre viene utilizzato e talvolta viene utilizzato erroneamente. Il consiglio quindi resta di inserirlo solo se si ha certezza quasi assoluta che ci stia. Attenzione a non confonderlo con l'**ABSTRACT FACTORY** nel caso in cui ci siano famiglie di elementi e ci sia la necessità di rappresentarle per tipologie differenti, la loro costruzione avviene sempre tramite **ABSTRACT FACTORY** e mai si utilizza un **DECORATOR**.

Si fa uso del **DECORATOR** soltanto quando nel testo si richiede di aggiungere funzionalità ad un oggetto.

1.7 Proxy Pattern

> *"La sorgente del programma e la componente osservatore sono dislocate su macchine differenti, ma interagiscono fra loro come se condividessero lo stesso contesto di esecuzione"*

Quando si parla di componenti su macchine differenti è il classico problema che risolve un **PROXY**.

2 Diagrammi dei casi d'uso

2.1 Identificare un caso d'uso

Molto spesso ogni caso d'uso proviene da una frase:

> oggetto + predicato + complemento oggetto -> (esempio: "La registrazione avviene fornendo username e password")

Ogni frase elementare (costituita dalla terzina) è un use case al 100%. Nel caso dell'esempio abbiamo due casi d'uso che mappano in uno più generale: 'La registrazione' è il caso d'uso generale (oggetto), le frasi semplici, ossia i casi d'uso più atomici, sono:

- *'La registrazione avvine fornendo username'*;
- *'La registrazione avviene fornendo password'*.

L'oggetto è in comune per cui è il caso d'uso generale delle due frasi semplici estrapolate dall'esempio. Il predicato si trasforma nell'azione di un ipotetico utente. Le due frasi quindi specializzeranno il caso d'uso generale in un nuovo diagramma.

2.1.1 Generalizzazione o specializzazione?

Si utilizza la **generalizzazione** nei casi in cui il caso d'uso principale è esteso e il caso d'uso eredita il suo comportamento specializzandolo. Notare che la **generalizzazione** tra casi d'uso è sempre la soluzione corretta nei casi in cui il testo descriva due comportamenti differenti per un unico caso d'uso utilizzando logicamente un OR (ad esempio quando si condivide qualcosa attraverso i vari social network o come nel caso della Ricerca spiegata in seguito).

2.2 Probabili casi d'uso

2.2.1 Registrazione

Sempre un use case a parte specializzato con l'inserimento di USER o MAIL e PASSWORD con qualche «extend» di errore;

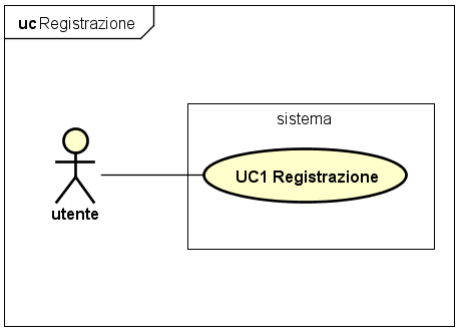

Figure 4: Caso d'uso generale Registrazione

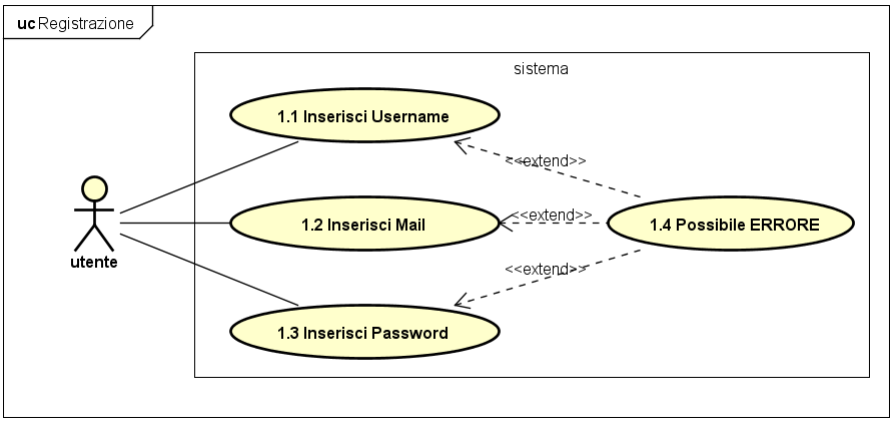

Figure 5: Caso d'uso in dettaglio Registrazione

2.2.2 Utenti

Spesso il caso è questo:

1. Utente non autenticato: ossia colui che può registrarsi, fare il Login e effettuare quelle operazioni definite spesso pubbliche.

2. Utente autenticato: ossia colui che si è loggato e può effettuare più operazioni oltre a quelle pubbliche. Solitamente non è indicato quest'ultimo punto per cui non serve una gerarchia di utenti. Nel caso servisse vedere il diagramma dei casi d'uso:

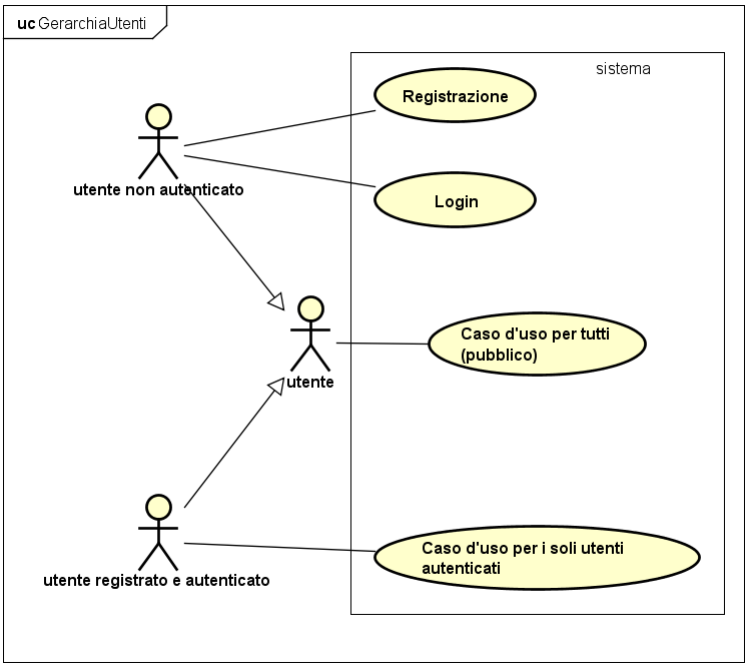

Figure 6: Gerarchia utenti generica

2.2.3 Altri utenti

L'utilizzo di altri utenti avviene nel caso si definisca il chi manipola dei dati esternamente dal nostro sistema (solidamente altri sistemi come Google, Twitter, Banca, Facebook etc). In particolare nelle condivisioni spesso si ha il caso di utilizzare un ulteriore attore esterno al quale accade qualcosa (condividi su facebook -> uso di Server Facebook come attore esterno), talvolta questo caso è insieme ad una **generalizzazione**. Gli utenti esterni non devono essere mai utenti della nostra applicazione. Nel caso si riferisca ad altri utenti nel testo della consegna si consideri tali utenti gli stessi con cui si sta

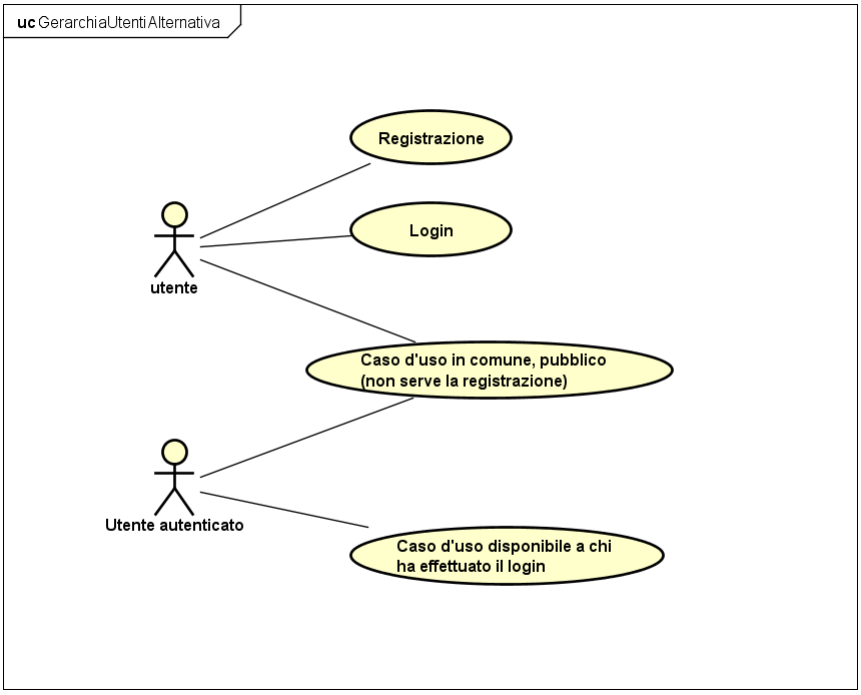

Figure 7: Gerarchia utenti generica alternativa

già lavorando. Ad esempio *"viene inviata una notifica ad un destinatario il quale può visualizzarla"* il destinatario non è altro che l'utente che stiamo manipolando ossia anche colui che causa l'invio di una notifica. In questo caso infatti basterà aggiungere il caso d'uso *Visualizza notifica* nei casi d'uso generali dell'utente della nostra applicazione, nient'altro.

2.2.4 Ricerca

Spesso il caso si divide in una generalizzazione dell'UC generico 'Ricerca...' nelle tipologie di 'Ricerca' descritte. Seguire il diagramma:

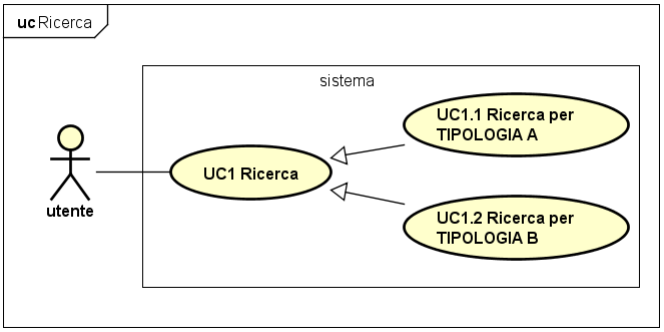

Figure 8: Ricerca generica

2.2.5 Lista di item

Spesso si trova che l'utente può visualizzare una lista, poco dopo sarà descritto cosa potrà visualizzare in dettaglio. In questi casi sempre:

1. Specificare il caso d'uso 'Visione lista item' con 'Visualizza desc items' o 'Visualizza titoli items' etc. come descritti. Ricordarsi di specificare le cose sempre al plurale in questo caso poiché si sta trattando un insieme di elementi.

2. Inserire il caso d'uso generale: 'Visualizza dettagli item' e poi specificarlo con 'Visualizza dettaglio A', 'Visualizza dettaglio B' etc.

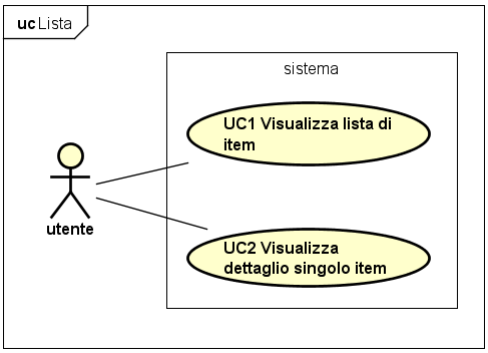

Figure 9: Lista generica

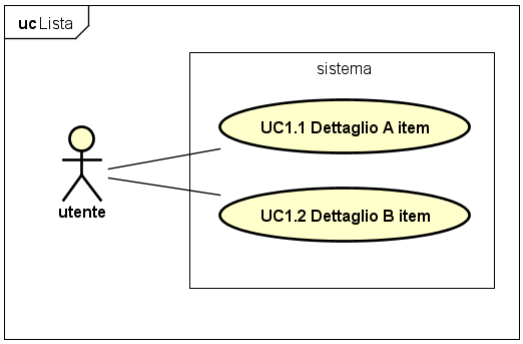

Figure 10: Dettagli item visualizzabili nella visione d'insieme della lista generica

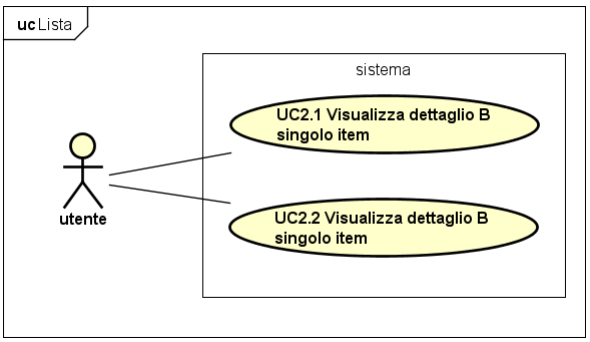

Figure 11: Dettagli singolo item

2.2.6 Condizioni

Quando nel testo si descrivono condizioni (ad es.: maggiore di N cose) si è in presenza di un «extends» obbligato. Nel caso d'uso che può generare in base al soddisfacimento della condizione dovrà essere appesa l'estensione del possibile avvenimento come mostrato nel diagramma:

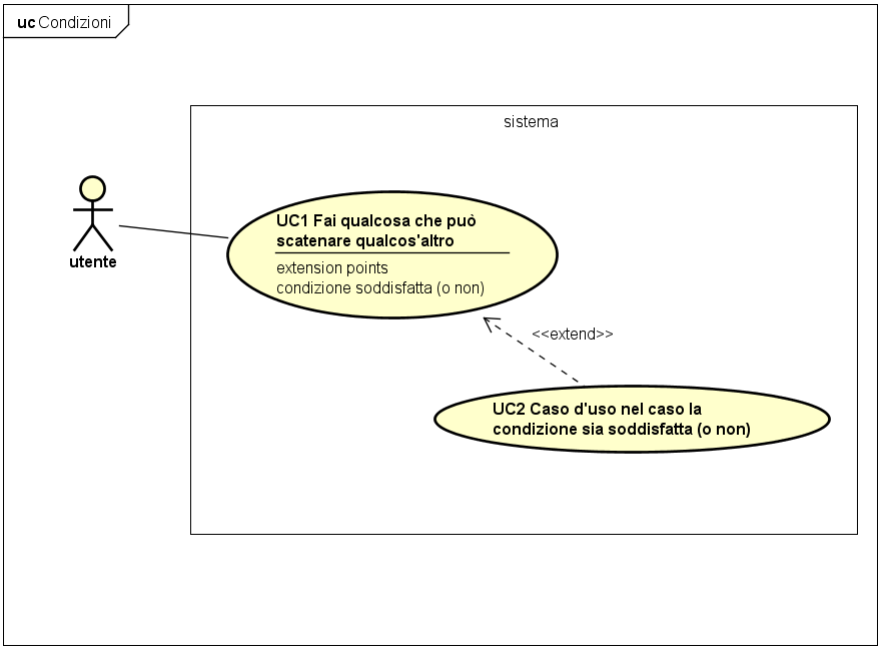

Figure 12: Uso di notazione *extends* e di una condizione

2.2.7 Inclusioni

L'uso di «include» si ha nei casi in cui il sistema a seguito di un caso d'uso fa qualcos'altro. Ad esempio un controllo sui dati inseriti. In generale l'inclusione si utilizza quando un caso d'uso dipende da un altro (relazione di inclusione). Dalle soluzioni riportate non viene mai usato, per cui nel minimo dubbio si **sconsiglia** sempre l'utilizzo di essa.