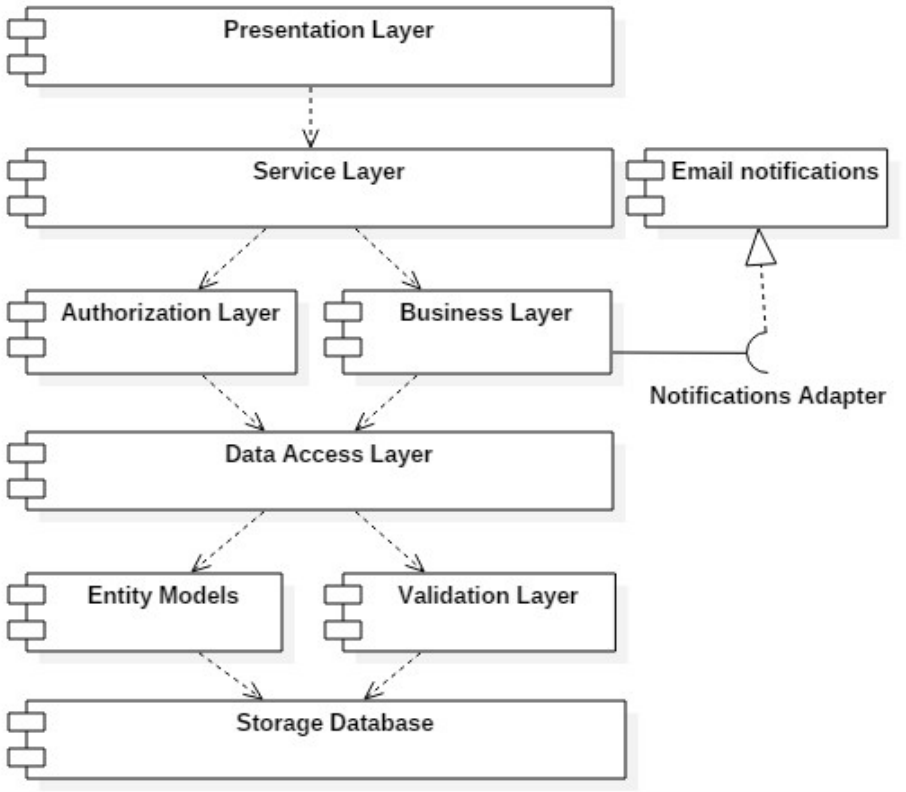
Components diagram: architettura

Nella terminologia del linguaggio di modellazione UML, un diagramma dei componenti, «Component Diagram», è un diagramma che ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini dei suoi componenti principali e delle relazioni fra di essi.

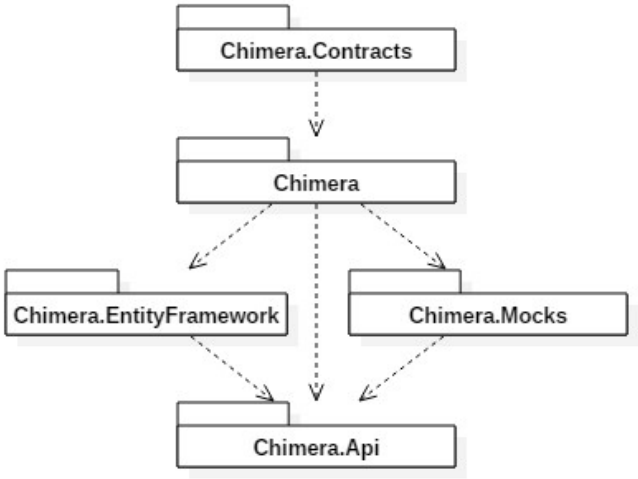


icubed

Package diagram: classificazione

Si tratta di un diagramma in grado di esprimere un aggregazione degli elementi di base del sistema (i modelli) usando una classificazione «logica» degli stessi, e non un approccio «fisico» come può essere nel caso di un «Component Diagram».

L'obiettivo principale di un «Package Diagram» è quello di raggruppare gli elementi secondo una specifica regola che permette a tutti gli elementi coinvolti di essere suddivisi in macro-categorie, che permettono di comprendere in maniera più immediata la strutturazione del sistema.



icubed

Deployment diagram: infrastruttura

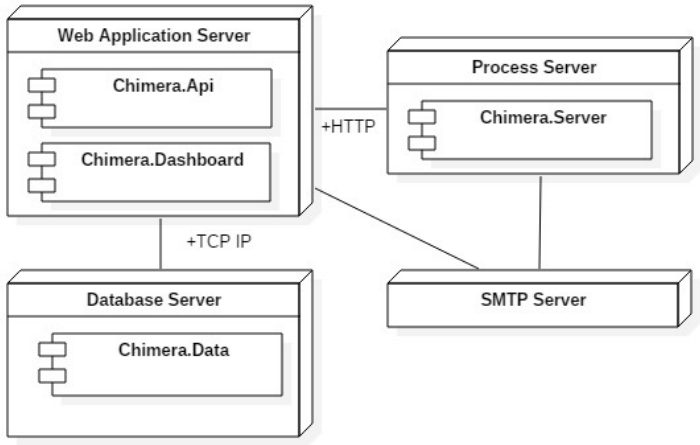
E’ un diagramma UML che permette di disegnare una overview dei componenti di infrastruttura fisica («hardware») su cui sarà eseguita l’applicazione finale.

Il linguaggio UML fornisce dei simboli che hanno il compito di favorire la creazione di un quadro chiaro di come il settaggio e la composizione finale dell’hardware dovrà essere. L’elemento principale dell’hardware è costituito da un nodo («node»), che costituisce un nome generico per ogni tipo di risorsa hardware.

Un cubo rappresenta un nodo.

Un nodo ha un suo nome ed è possibile anche usare uno stereotipo (icona) per indicare il tipo di risorsa che esso rappresenta.

Una linea che unisce due cubi rappresenta una connessione tra i due nodi: è possibile usare anche uno stereotipo anche per fornire informazioni sulla connessione, ma in generale si preferisce dare indicazioni riguardo la tipologia di connessione che esiste tra i due elementi.

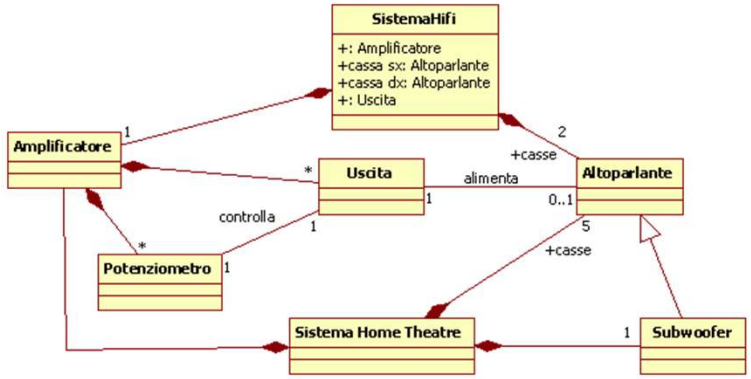


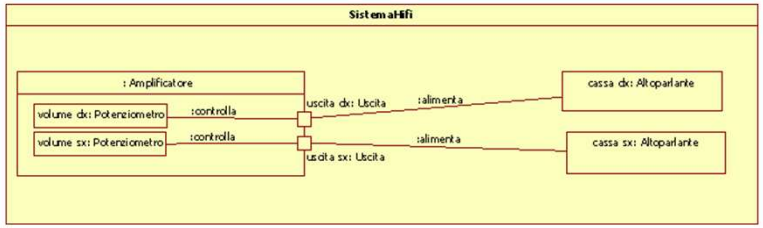
icubed

Composite Structure Diagram

I «diagrammi di struttura composta» sono un nuovo tipo di diagramma introdotto nella versione 2.0 del linguaggio UML. Consentono la rappresentazione della struttura interna di classi e altri componenti software.

Prima dell'introduzione di questo tipo di diagrammi era comunque possibile rappresentare strutture complesse usando le notazioni «parte-componente» e un diagramma di classi; ma il disegno risulta diventare molto complesso con il crescere della complessità dell'oggetto da rappresentare.





icubed

Use case diagram: cosa sono? (1)

Utilizzare uno «Use Case Diagram» permette di avere una rappresentazione grafica di come il sistema far interagire le classi del dominio, ed evidenziare come le stesse si evolvono nel corso del tempo.

Lo Use Case Diagram rappresenta un eccellente strumento per stimolare dei potenziali utenti a intervenire con le loro critiche ed opinioni sulla funzionalità che il sistema dovrà avere.

Non è sempre facile per gli utenti riuscire a spiegare tecnicamente come essi intendano costruire il sistema desiderato: spesso è più semplice per loro esprimere un concetto utilizzando esempi pratici che razionalizzare il comportamento in un modello teorico.

icubed

Use case diagram: cosa sono? (2)

La sequenza di eventi descritta da uno Use Case viene iniziata da una persona, o da un altro sistema o da un pezzo di hardware o ancora dal passare del tempo.

Le entità che iniziano la sequenza di eventi sono definiti Actors. Il risultato della sequenza deve portare a qualcosa di utile all'actor che ha iniziato la sequenza o, anche, ad un differente actor.

«Inclusion» - questo modalità permette di utilizzare i passi appartenenti ad una sequenza di un use case e inglobarli in un altro use case mettendo un riferimento al precedente

«Extension» - è possible creare un nuovo use-case semplicemente aggiungendo dei passi ad un use case esistente.

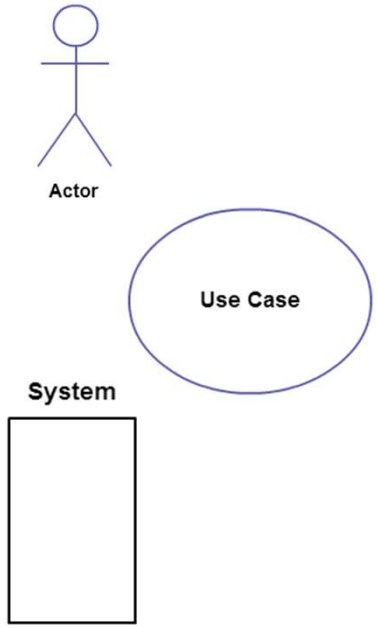
icubed

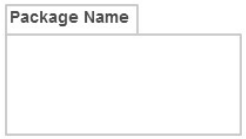
Use case diagram: i componenti

Un «actor» inizia la sequenza di un particolare use case, ed un «actor» - lo stesso che ha iniziato la sequenza, ma non deve essere necessariamente lo stesso - riceve un ritorno dallo use case.

Graficamente, un' «ellisse» rappresenta un use case e la stilizzazione di una persona rappresenta un «actor».

L'actor che inizia la sequenza di eventi di un use-case viene sempre inserito alla sinistra dell'use case mentre, al contrario, l'actor che riceve gli effetti causati dalla sequenza scatenata dall'use case stesso, viene disegnato alla destra.





icubed

Use case diagram: i componenti

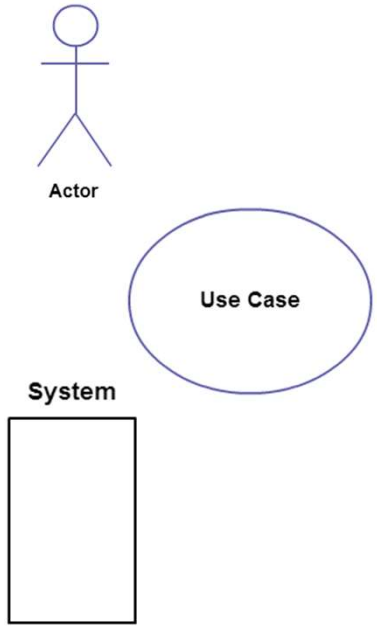
Il nome dell'actor appare appena sotto la raffigurazione dello stesso actor mentre il nome dell'Use Case appare o dentro l'ellisse o appena al di sotto di esso. Una linea di associazione, infine, connette un actor all'use case e rappresenta la comunicazione tra l'actor e lo use case.

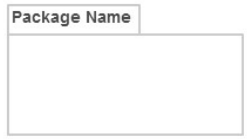
La linea di associazione è una linea continua, simile a quella che connette due classi in associazione tra loro.

Due componenti aggiuntivi utilizzati in questo contesto sono

«system» - serve a rappresentare il sistema di esecuzione dello use case

«package» - serve a raggruppare diversi «use case» per una migliore leggibilità





icubed

Use case diagram: la sequenza

Come detto, ogni use case è una lista di «scenari», ed ogni scenario è una sequenza di «passi». Per ciascun use case, ogni scenario avrà la sua propria pagina rappresentata nel seguente modo:

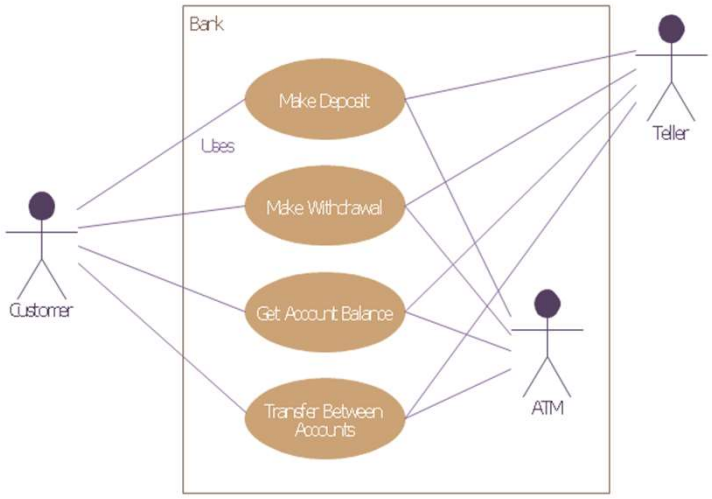
Un «actor» che dà inizio alla sequenza dell'use case

Traccia delle «pre-condizioni» per lo use case (i requisiti di esecuzione)

I passi dello scenario vero e proprio (il più possibile dettagliati)

Traccia delle «post-condizioni» quando lo scenario è completo

L' «actor» che beneficia dello use case e della sua computazione



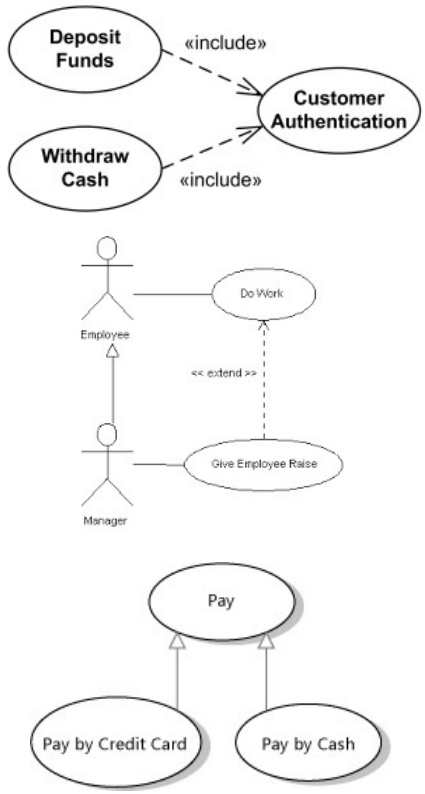
icubed

Use case diagram: relazioni tra elementi

Inclusion - Permette il riutilizzo di un use case all'interno di un altro use case. Per rappresentare graficamente una inclusion, si utilizza il simbolo utilizzato per la dipendenza tra classi – una linea tratteggiata che connette le classi con una freccia che punta sulla classe da cui dipende l'altra classe. Appena sotto questa linea, si aggiunge uno stereotipo – la parola <<include>> racchiusa tra doppie parentesi formate dai simboli "<<" e ">>"

Extension - Permette di creare un nuovo use case aggiungendo dei passi in più ad un use case già esistente. Anche qui una linea tratteggiata con una freccia finale viene utilizzata per rappresentare l'extension, insieme con uno stereotipo che mostra la parola <<extends>> tra parentesi. All'interno dell'use case base, il punto di extension appare sotto il nome dell'use case stesso.

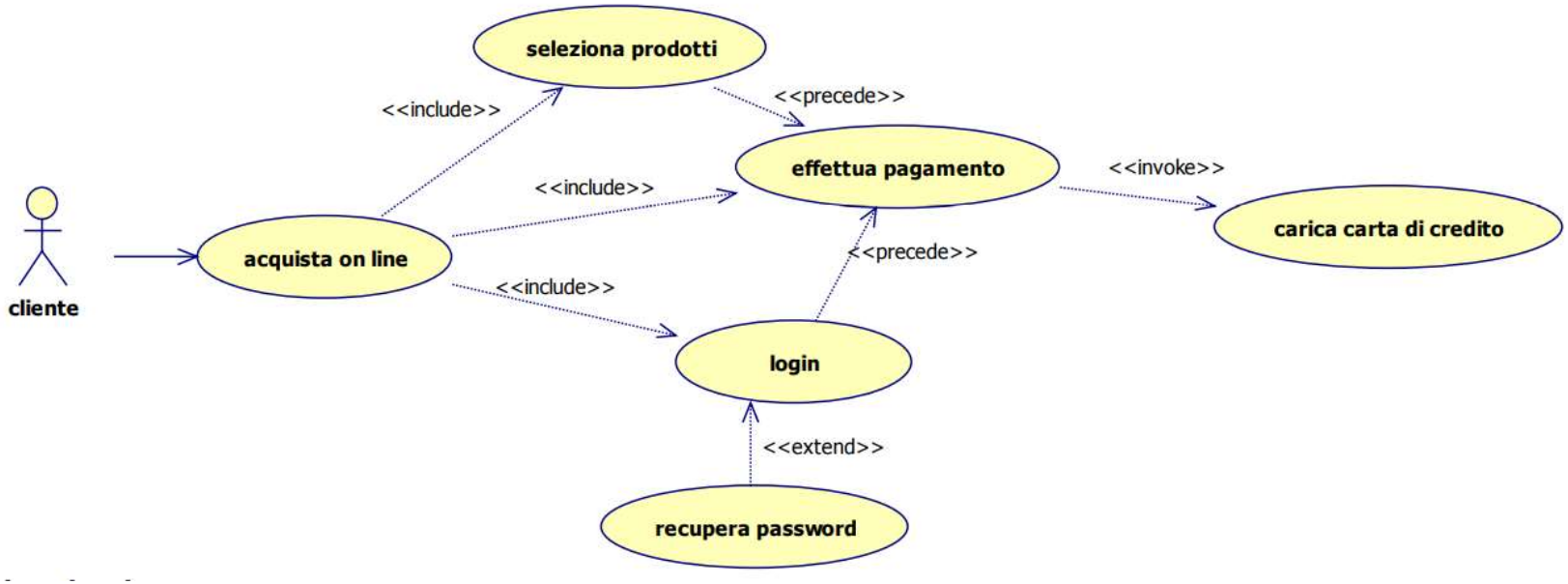
Generalization - Le classi possono ereditare le caratteristiche di un'altra classe (classe padre). Allo stesso modo, tale ragionamento è applicabile agli use case. Nella eredità tra use case, lo use case figlio eredita il comportamento ed il significato dal padre ed in più aggiunge le sue caratteristiche specifiche.



icubed

Use case diagram: un esempio pratico

Sono dati due sistemi informativi: un software per la gestione degli ordini di un ristorante e un sistema di e-commerce. Gli use case descritti negli schemi rappresentano una semplificazione del processo che – tuttavia – è in grado di dare immediatamente una panoramica di quello che si vuole realizzare:



icubed

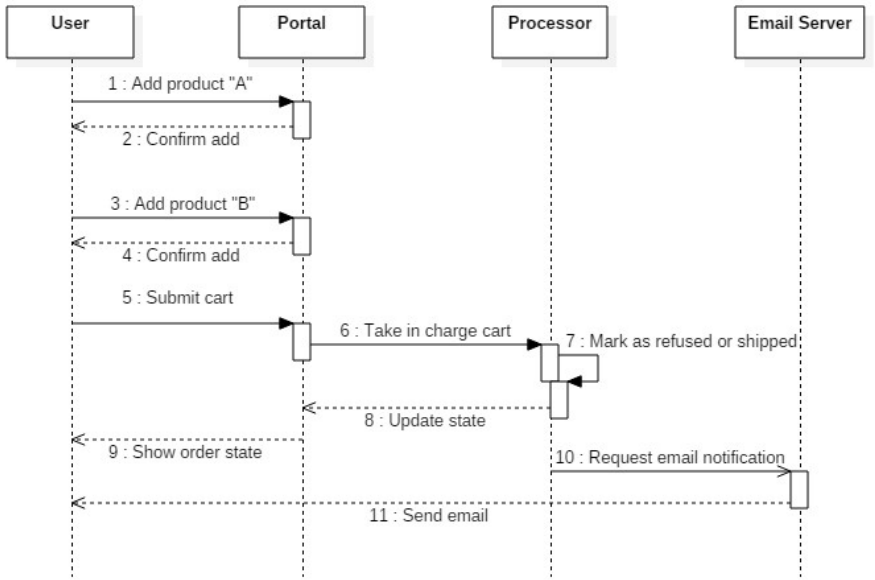
Sequence Diagrams

I «diagrammi di sequenza» coprono l'aspetto comportamentale del sistema che definisce «come» gli elementi interagiscono tra di loro e come si svolge il processo logico.

Il focus principale è sulla «sequenza temporale»

Il Sequence diagram è costituito da oggetti rappresentati nel modo ormai usuale: rettangoli recanti un nome, messaggi rappresentati da linee continue recanti una freccia alla loro fine e il tempo rappresentato come progressione verticale.

Ogni singola azione che si svolge nel sistema è marcata con un numero di sequenza, che rende maggiormente leggibile il processo di accadimento delle azioni e quali entità sono convolte in essa come «emettitori» dell'azione e quali invece solo predisposti come «ricettori» della risposta relativa all'azione (linee tratteggiate).



icubed

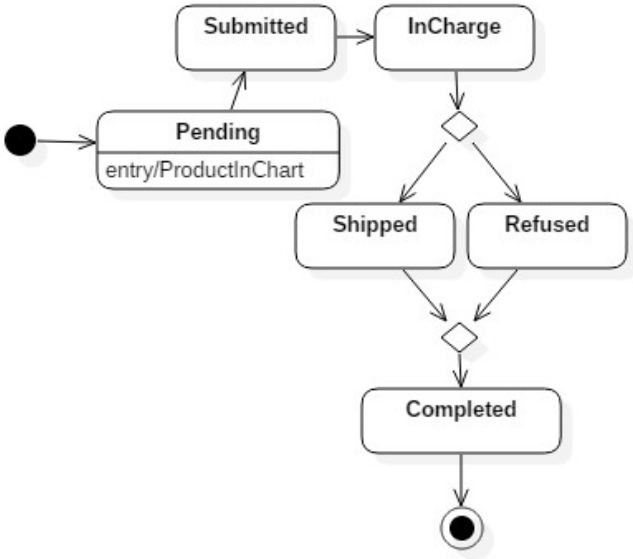
State Diagrams: macchina a stati

Lo scopo principale dei «diagrammi di stato» è tracciare il mutamento degli «stati» di un particolare oggetto; come questo varia il suo comportamento in funzione alla situazione/configurazione («stato») in cui lo stesso oggetto si trova.

Definisce la «transizione» tra uno stato ed un altro, come uno stato viene attivato in funzione di un evento esterno all'oggetto oppure proveniente da una particolare azione che l'elemento spontaneamente avvia.

Lo «State Diagram» mostra l'inizio della sequenza di stati («initial state») e la sua conclusione («final state») in modo esplicito, è composto da elementi decisionali («choice»), di ricongiunzione («join») e di separazione/ricomposizione («fork» e «merge») che servono per dare una raffigurazione completa delle possibile casistiche che si possono incontrare in un processo di cambiamento di stato di un oggetto.

E' possibile indicare sugli State Diagram un evento che causa il verificarsi di una transizione (un evento trigger) e l'azione che viene eseguita che fa sì che il cambio di stato possa avvenire.



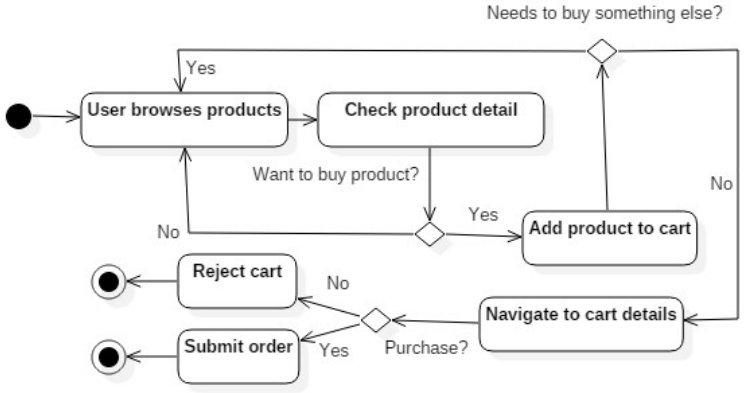
icubed

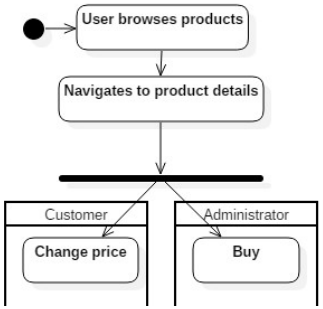
Activity Diagrams: i diagrammi di flusso

Si tratta di diagrammi che posseggono alcune funzionalità «avanzate» rispetto ai classici diagrammi di flusso

Permettono di esprimere in maniera ancora più precisa il funzionamento del sistema in termine di «singole attività» che il software deve svolgere
per portare a termine lo scopo di business per il quale è stato concepito.

Rispetto ad un normale «flowchart» un activity diagram è in grado di le «swimlanes». Si tratta di aree dove il processo di attività si differenzia a seconda del ruolo dell'utente (riportato in alto alla «corsia di nuoto») che ha la funzione principale di racchiudere le activity specifiche del ruolo stesso.





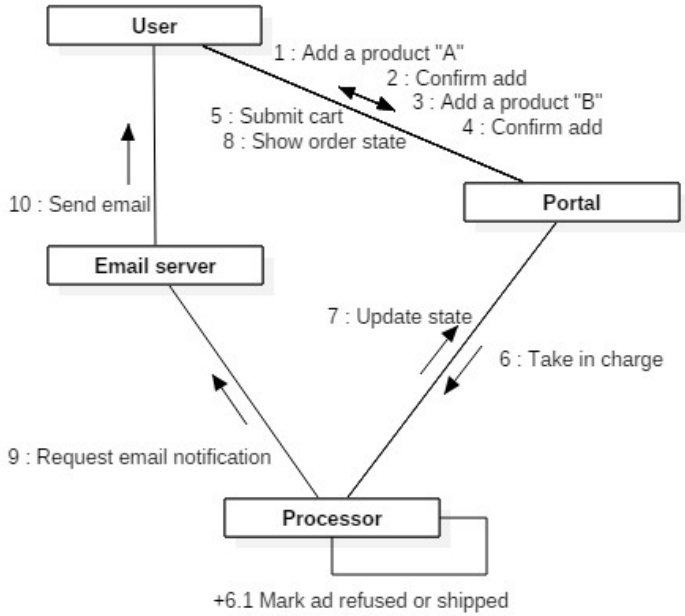
icubed

Communication Diagrams

Un «Communication Diagram» (o «Collaboration Diagram» come era chiamato nelle specifiche di UML 1.x) è un diagramma che descrive l'interazione fra più partecipanti alla realizzazione di una certa funzionalità.

Insieme al Sequence Diagram fa parte dei cosiddetti diagrammi di interazione («interaction diagrams»), che sono spesso utilizzati per specificare come collaborano i vari componenti del software.

Il suo contenuto è molto simile a quello di un diagramma di sequenza, con la differenza che gli elementi principali non sono distribuiti nella parte alta dei diagramma ma secondo una logica «collaborattiva» (più vicini quelli con maggiore interazione).



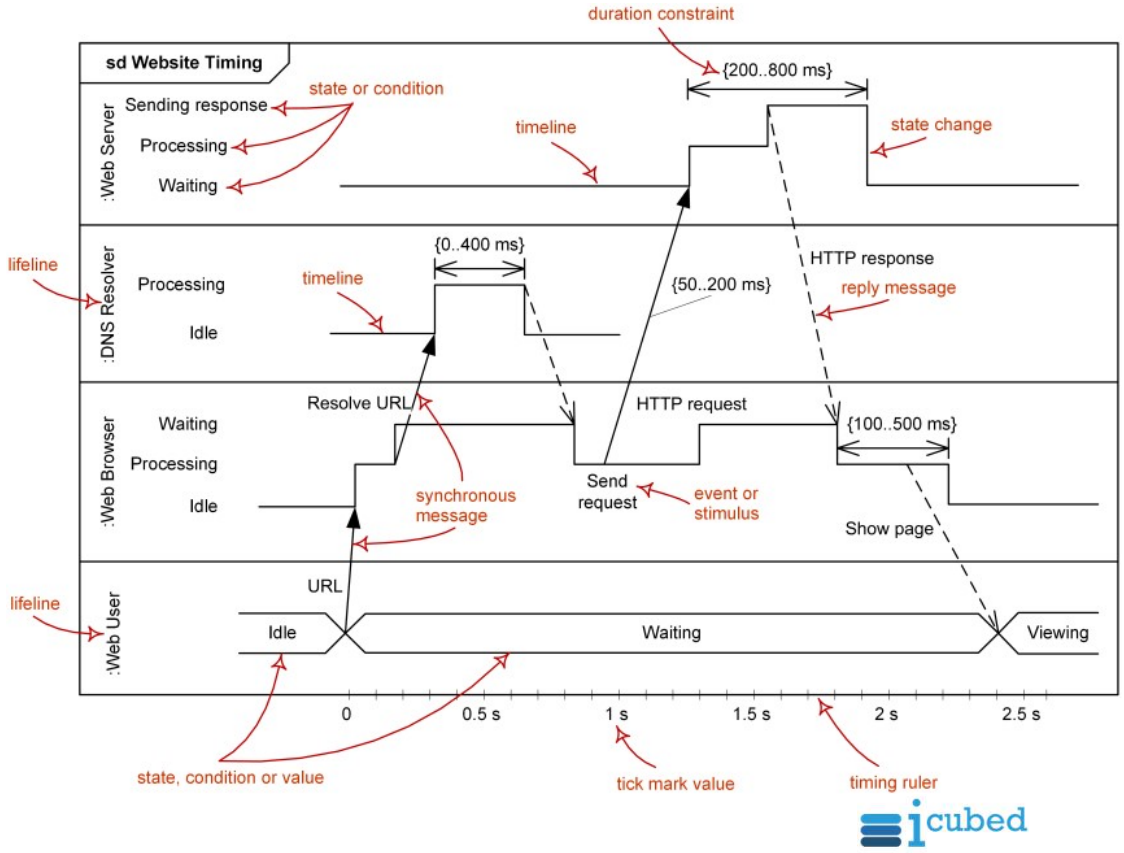
icubed

Timing Diagram

Un «timing diagram» o diagramma temporale, nella versione 2.0 del linguaggio UML, è uno specifico tipo di diagramma di interazione, dove l'attenzione è focalizzata sui vincoli di tempo.

Questo tipo di diagrammi sono usati per esplorare il comportamento di un oggetto in un dato periodo di tempo.

Rappresentano una forma speciale di «sequence diagram» che sposta sull'asse delle ascisse la variabile temporale e su quello delle ordinate l'oggetto di interazione (detti «lifeline») ed eventualmente i suoi stati («state o condition»).

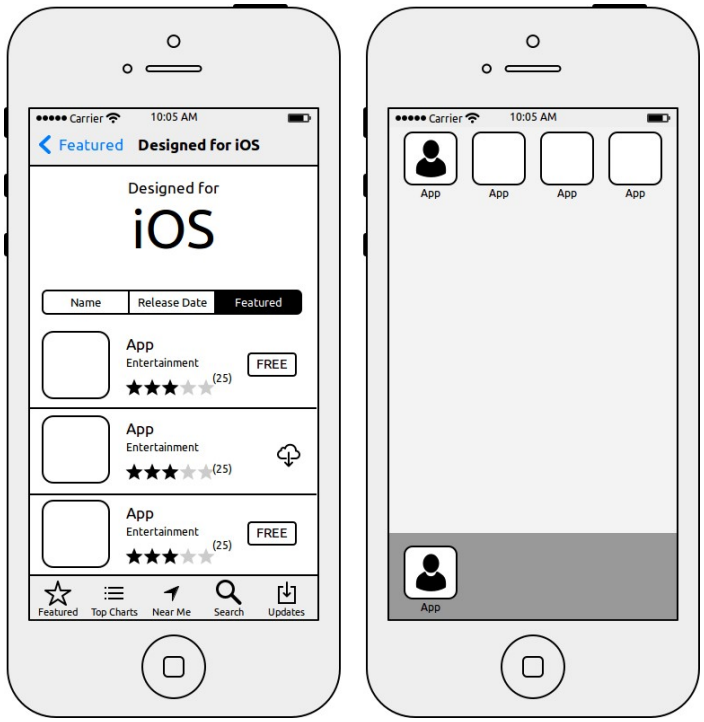


L’importanza del «wireframing»

Il wireframing rappresenta un modo per realizzare delle interfacce grafiche «fittizie» in grado di dare un’idea del prodotto che si vuole realizzare.

Il wireframing è una parte molto importante nel processo di sviluppo di un sistema informatico. Un wireframe rappresenta la struttura base della user interface di un software; gli elementi che costituiscono il design sono rappresentati da box in scala di grigio che - grazie alla semplicità di forme e colori - permettono la rapida identificazione di problemi legati all’interazione uomo-macchina e all’usabilità.

Il cliente finale è in grado di dare un feedback sul sistema ancor prima che venga realizzato e il team di sviluppo non dovrà perdere tempo a cambiare continuamente la «presentation» del software.



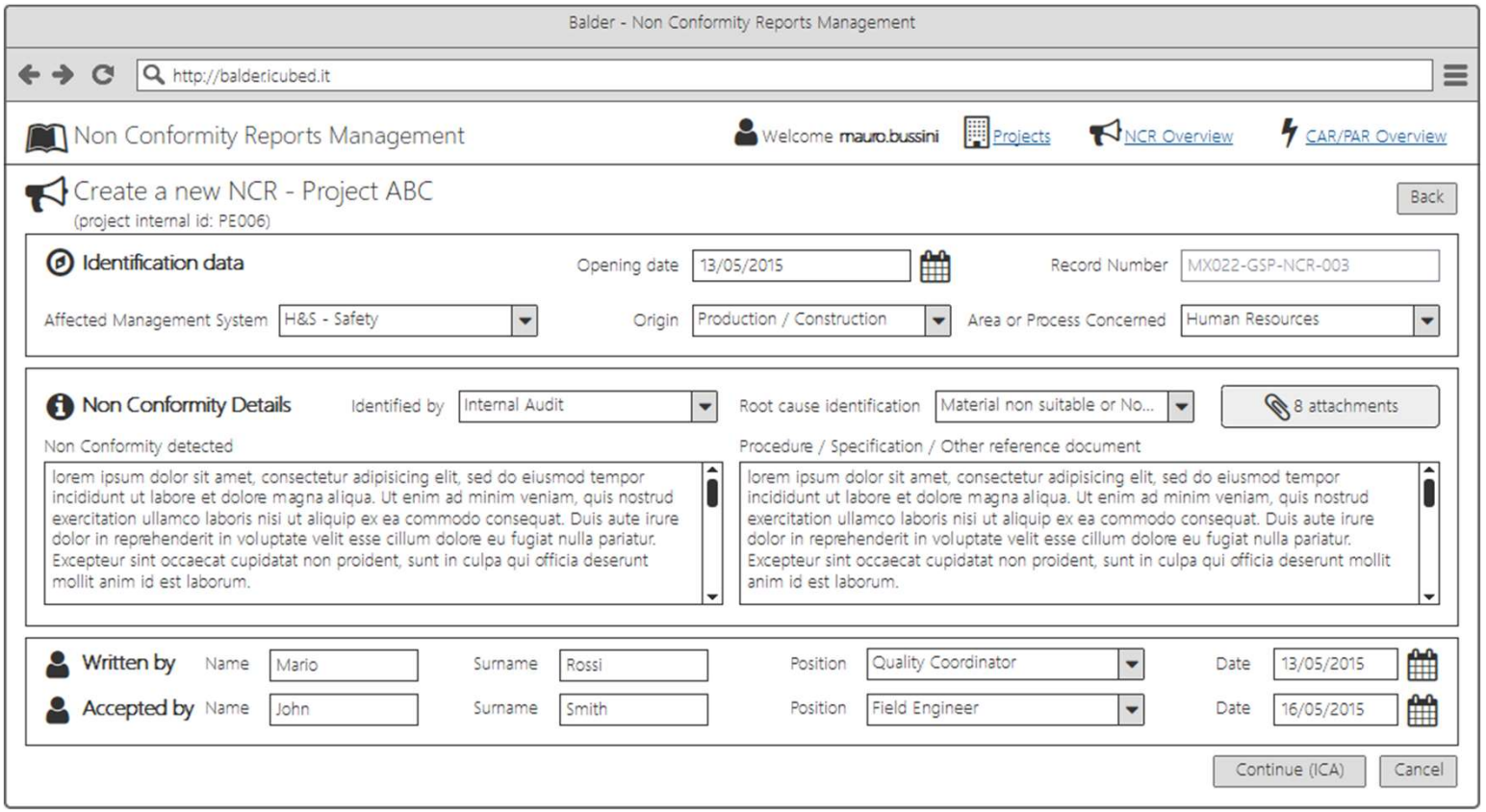
icubed

Wireframing: i vantaggi

L'adozione di strumenti di questo tipo permettono anche una migliore comunicazione tra il team di analisi e quello operativo di sviluppo.

Si riducendo drasticamente le situazioni di incomprensione e si condivide l'obiettivo finale senza lasciare spazio ad intepretazioni personali.

Wireframing tool consigliato: **Balsamiq** (https://balsamiq.com)



icubed

Demo

I **dipendenti** delle officine hanno il compito di **registrare i dati dei veicoli** in ingresso, di interrogare l'archivio delle riparazioni da effettuare, di effettuarle e **registrarne la terminazione**, consegnando i veicoli riparati ai clienti. Tuttavia, per riparazioni particolarmente complesse, quest'ultimo compito viene lasciato ai **direttori** che provvedono al **rilascio di una particolare garanzia ai clienti**.

Infine i direttori devono avere la possibilità di interrogare ed eventualmente modificare i dati personali dei propri dipendenti.

Esercitazione n.5

Si vuole realizzare un'applicazione per la gestione di biglietti di visita al museo.

Il sistema vuole tenere traccia sia dei biglietti venduti che degli utenti che hanno fatto richiesta di partecipazione.

In particolare per ciascun biglietto si vuole tenere traccia di:

- Codice identificativo tipo «AB123», data della visita, prezzo del biglietto, del museo per il quale si fa richiesta (nome) ed eventualmente della guida di riferimento.

Di ogni visitatore si vuole tenere traccia del nome, cognome e dell'età.

Delle guide si vuole tenere traccia del nome, cognome, numero di telefono.

Gli utenti possono visitare il museo, comprando un biglietto alla biglietteria o usando biglietti acquistati precedentemente.

La visite avvengono da soli oppure con una guida.

Alcune categorie di visitatori hanno diritto ad un biglietto ridotto, previa dimostrazione dell'applicabilità della riduzione.

Si rappresenti il sistema con un diagramma dei casi d'uso.

Data la sequenza di requisiti riportati nella slide precedente realizzare il diagramma delle classi, l'object diagram e lo use case diagram.

icubed