Elementi di UML (1)

Università degli Studi di Bologna
Facoltà di Scienze MM. FF. NN.
Corso di Laurea in Scienze di Internet
Anno Accademico 2004-2005

Laboratorio di Sistemi e Processi Organizzativi

Metodo di Sviluppo

- Metodo = Linguaggio + Processo
- Il *Linguaggio di Modellazione* è la notazione per esprimere le caratteristiche del progetto
- Il *Processo* consiste in una serie di passi da intraprendere per produrre il progetto

Unified Modeling Language (UML)

UML è un linguaggio semiformale e grafico per:

- specificare
- visualizzare
- realizzare
- modificare
- documentare

gli *artefatti* di un sistema software.

Artefatti: sorgenti, eseguibili, documentazione, risultati di test, ecc.

Indipendente dai metodi

UML è *indipendente dai metodi*:

- non è legato ad uno specifico *processo*
- non fornisce indicazioni sul proprio utilizzo
- può essere usato da persone che seguono *Metodi di Sviluppo* diversi

A cosa serve?

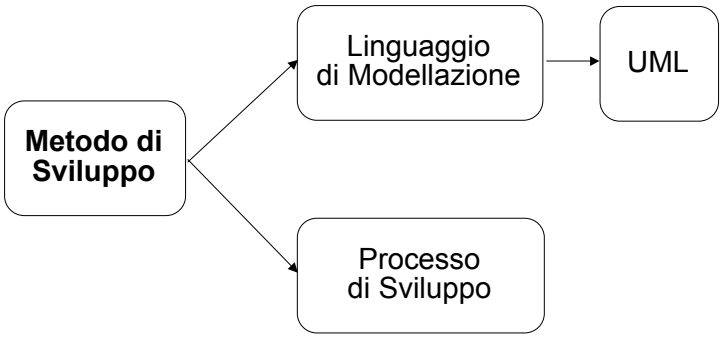
- Quindi UML serve a
  - specificare le caratteristiche di un nuovo sistema
  - a documentarne o modificarne uno esistente (senza perdersi nei dettagli dei linguaggi di programmazione durante la fase progettuale)

Cos'è e cosa non è

Linguaggio Unificato per la *Modellazione* di sistemi (software, hardware, organizzativi, ecc.)

- È uno strumento di comunicazione tra i diversi ruoli coinvolti nello sviluppo e nell'evoluzione dei sistemi
- È universale, ossia può rappresentare sistemi molto diversi.
- Non è un *Metodo di Sviluppo*
- Non è un *linguaggio di programmazione*
- È un ***linguaggio di modellazione*** **(o *progettazione*)**

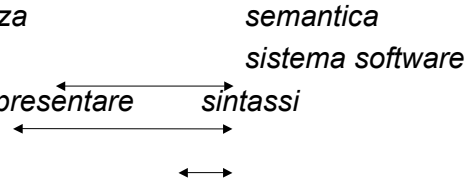
Metodologia e UML



Unified Modeling Language (UML)

Linguaggio:

- non una semplice *notazione* per disegnare diagrammi
- ma un *linguaggio* completo per catturare la *conoscenza* di un *soggetto* e *rappresentare* tale conoscenza.



Unified Modeling Language (UML)

Unificato:

– risultato dalla unificazione di metodologie e tecniche per la realizzazione di sistemi informatici

– definito con il contributo di molti metodologi e delle più importanti società di software mondiali

– gli originatori: *Grady Booch*, *Ivar Jacobson*, *Jim Rumbaugh*, tre esperti dell'approccio *Object Oriented*

Unified Modeling Language (UML)

Unificato:

– *la sua evoluzione è a carico dell'OMG (Object Management Group) e soggetta a procedure ben definite per ogni cambiamento*

– standard OMG dal novembre 1997 (UML 1.1)

– versione attuale: 1.5

– sito ufficiale di riferimento *http://www.omg.org*

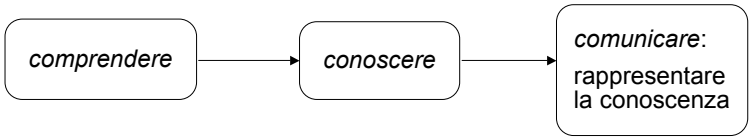
Unified Modeling Language (UML)

Modellazione:

– comprensione del soggetto in analisi

– conoscenza del soggetto in analisi

– essere in grado di comunicare la propria conoscenza



Perché modellare?

Si può scrivere codice

– disinteressatamente
  - l'importante è che funzioni
  - ci si preccupa di come apportare modifiche quando si ha la necessità di apportarne

– dopo aver formulato un modello adeguato
  - deve essere chiaro cosa si vuole
  - quindi come raggiungerlo

Perché modellare?

Si può scrivere codice

- disinteressatamente
  - l'importante è che funzioni
  - ci si preoccupa di come apportare modifiche quando si ha la necessità di apportarne
- dopo aver formulato un modello adeguato
  - deve essere chiaro cosa si vuole
  - quindi come raggiungerlo

Unified Modeling Language (UML)

Modellare:

- chiarifica bene le idee, non solo quelle del *progettista*, ma anche quelle del *cliente*
- permette di definire la struttura di un sistema
- permette di suddividere i compiti di sviluppo tra più persone
- lascia una traccia documentata delle attività e delle decisioni prese

Breve storia di UML

Agli inizi degli anni ’90 si sente la necessità di un linguaggio e di una notazione universale per la creazione di modelli software.

Si suggeriscono numerosi approcci e metodi di sviluppo, intorno ai quali si costituiscono gruppi di ricerca in competizione fra loro.

Nel 1994 ***Grady Booch*** e ***James Rumbaugh*** della *Rational Software Corporation* decidono di unificare i propri metodi, rispettivamente ***Booch*** e ***OMT*** (*Object Modeling Technique*)

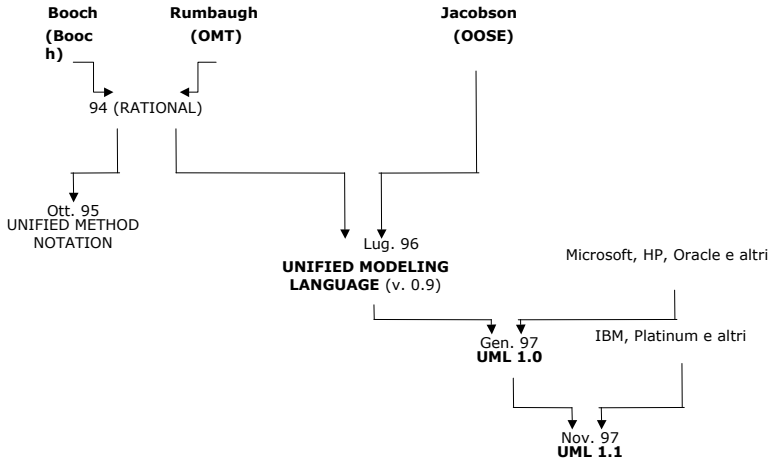
Breve storia di UML

Nel 1995 al processo di unificazione collabora ***Ivar Jacobson*** integrando il suo ***OOSE*** (*Object Oriented Software Engineering*).

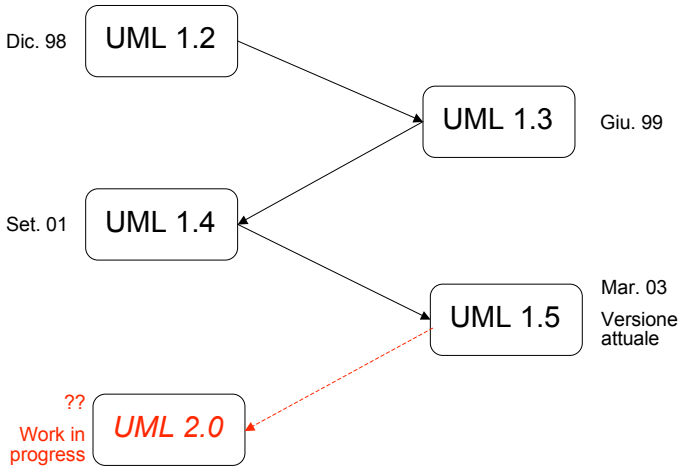
Nel 1996 nasce *Unified Modeling Language (UML)*.

Nel 1997 l’*OMG* emette una versione standard di *UML* che si basa sulla versione di *Booch*, *Rumbaugh* e *Jacobson* e sui contributi di alcune delle più importanti software house mondiali

Breve storia di UML



Evoluzione di UML



Affinità e somiglianze

- UML è un linguaggio non proprietario, standard
- UML è un'evoluzione di proposte preesistenti
- UML ha forti affinità e somiglianze con:
  - Diagrammi Entity-Relationship (ER)
  - Flow Chart
  - Notazioni Object Oriented (OO)

Motivazioni di UML

- Notazione di modellazione standardizzata e semanticamente ricca
- Unificazione di presistenti metodi di modellazione:
  Booch, OMT, OOSE
- Offrire flessibilità nell'applicazione di diversi processi di sviluppo

Definire un linguaggio formale

Linguaggio formale:

Sintassi + Semantica

Sintassi = Le regole attraverso cui gli elementi del linguaggio *(ad es. le classi)* sono raggruppati in espressioni *(ad es. i diagrammi)*

Semantica = Le regole che assegnano un significato alle espressioni sintattiche

Regole: esempi

- ***Regola sintattica*:**
  "Una classe deve essere rappresentata da un rettangolo suddiviso in tre parti da due linee orizzontali"

- ***Regola semantica*:**
  "Se una classe non è astratta, tutte le sue operazioni devono essere associate ad un corrispondente metodo presente nella classe"

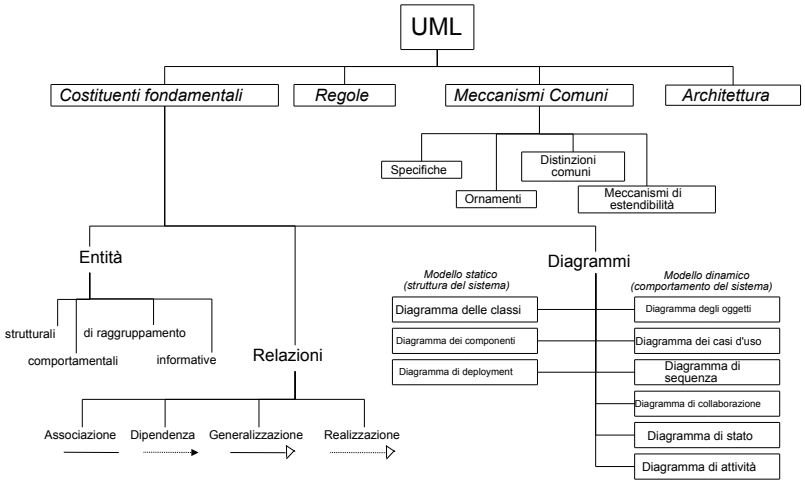
Il metamodello di UML

- UML è descritto da un modello composto da diversi elementi collegati tra loro da regole precise
- La sintassi e la semantica di UML sono descritte tramite UML stesso
- Questa metadescrizione si chiama ***metamodello***
- Esiste una rappresentazione XML del metamodello che si chiama XMI

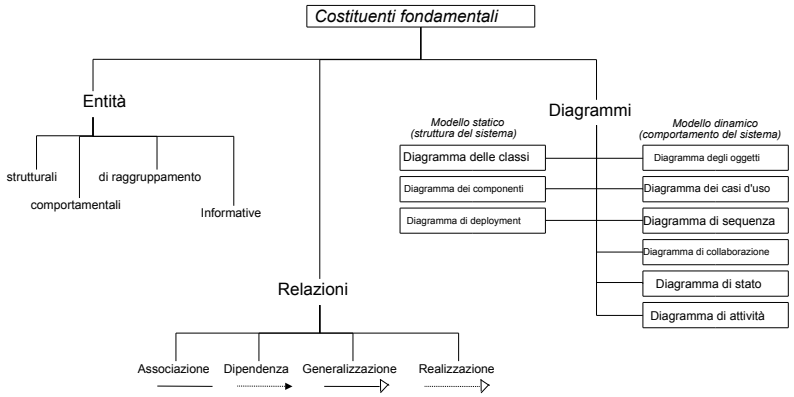
Il metamodello di UML

- Il metamodello concettuale di UML è costituito da quattro elementi principali:
  - ***Costituenti fondamentali*** (*building blocks*): entità, relazioni e diagrammi di base
  - ***Regole*** (*rules*) con cui comporre insieme i costituenti fondamentali
  - ***Meccanismi comuni*** (*common mechanisms*) tecniche comuni per raggiungere specifici obiettivi con UML
  - ***Architettura*, modo in cui UML esprime l'architettura di un sistema**

Metamodello



Costituenti fondamentali



Costituenti fondamentali

- ***Entità (things)***: elementi principali del modello *(classi, interfacce, componenti, casi d'uso, ecc..)*
- ***Relazioni (relationships)***: legano tra loro le entità, definendone le correlazioni semantiche *(associazioni, generalizzazioni, dipendenze, ecc..)*
- ***Diagrammi (diagrams)***: consentono di vedere il contenuto del modello da "punti di vista" diversi *(diagrammi delle classi, dei casi d'uso, d'interazione, ecc..)*

Entità

Quattro categorie:

- ***Entità strutturali (structural things)***: costituiscono la parte statica di un modello, rappresentano elementi sia concettuali che fisici
- ***Entità comportamentali (behavioral things)***: costituiscono la parte dinamica del modello, rappresentano il comportamento nel tempo e nello spazio
- ***Entità di raggruppamento (grouping things)***: costitiscono la parte organizzativa del modello, consentono la decomposizione del modello
- ***Entità informative (annotational things)***: *costituiscono la parte esplicativa del modello, sono commenti per descrivere, evidenziare, rimarcare qualsiasi elemento del modello*

Entità strutturali

Sette tipi:

- Class: descrive un insieme di oggetti condividenti gli stessi attributi, operazioni, relazioni e semantica
- Active class: classe i cui oggetti rappresentano elementi il cui comportamento è concorrente con altri elementi
- Use case: descrizione di un insieme di azioni che un sistema esegue fornendo un risultato osservabile ad un particolare attore

Entità strutturali

- Collaboration: definisce una interazione ed è una collezione di ruoli che insieme forniscono un comportamento cooperativo
- Interface: collezione di operazioni che specificano un servizio di una classe o componente; descrive il comportamento esterno visibile di quell'elemento
- Component: una parte fisica del sistema che fornisce la realizzazione di un insieme di interfacce
- Node: un elemento fisico che esiste a run time e rappresenta una risorsa computazionale

Entità comportamentali

Due tipi principali:

- Interaction: un comportamento comprendente un insieme di messaggi scambiato tra un insieme di oggetti in un particolare contesto per compiere uno specifico scopo
- State machine: un comportamento specificante la sequenza degli stati che un oggetto assume durante la sua vita in risposta ad eventi

Entità di raggruppamento

Un solo tipo principale:

Package:

- meccanismo generale per organizzare gli elementi in gruppi
- *structural thing*, *behavioral thing* ed altro possono essere raggruppati in un *package*

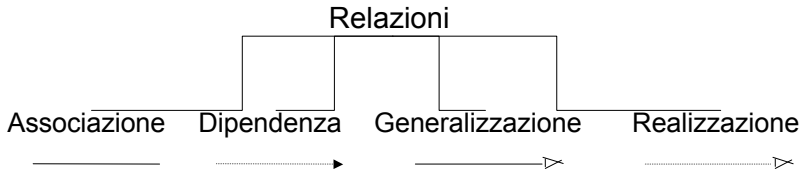
Entità informative

Un solo tipo principale:

Note: commenti per descrivere ed evidenziare qualsiasi elemento in un modello

Relazioni

- ***In un modello, le relazioni mostrano come due o più entità siano tra loro correlate.***
- ***Quattro tipi:***
  - ***Associazione (Association)***
  - ***Dipendenza (Dependency)***
  - ***Generalizzazione (Generalization)***
  - ***Realizzazione (Realization)***



Relazioni

- ***Associazione (Association)***

  relazione strutturale che descrive l'insieme di collegamenti tra entità

  Ad esempio:

- ***Dipendenza (Dependency)***

  relazione semantica tra due entità in cui il cambiamento di una (quella indipendente) può influenzare la semantica dell'altra (quella dipendente)

Relazioni

- ***Generalizzazione (Generalization)***

  relazione di generalizzazione/specializzazione, in cui oggetti dell'elemento specializzato (figlio) sono sostituibili all'oggetto generalizzato (genitore). I figli condividono la struttura ed il comportamento del genitore

- ***Realizzazione (Realization)***

  relazione semantica tra elementi, in cui uno specifica un "contratto" con un altro elemento che ne garantisce l'attuazione

I modelli presenti

- ***Modello statico:***
  - ***Fissa le entità e le relazioni strutturali tra le entità***
  - ***Rappresenta gli elementi di un sistema software e le loro relazioni statiche (cioè invarianti al trascorrere del tempo)***
- ***Modello dinamico:***
  - Fissa il modo in cui le entità interagiscono per generare il comportamento richiesto al sistema software
  - Rappresenta il comportamento di un sistema software al trascorrere del tempo

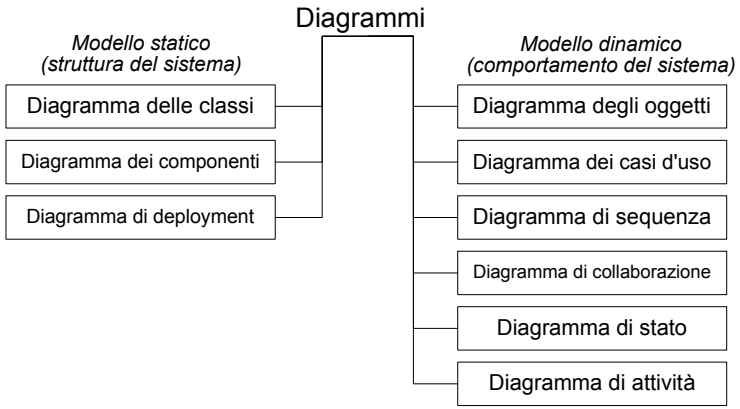
I diagrammi canonici (1.5)

Esistono novi diversi tipi di diagrammi:

- Use case
- Class and Object
- Behavior
  - Statecharts
  - Activity
  - Interaction
    - Sequence
    - Collaboration
- Implementation
  - Component
  - Deployment

I diagrammi canonici (1.5)

Tipicamente divisi tra quelli che modellano la *struttura statica* e quelli che modellano la *struttura dinamica* del sistema



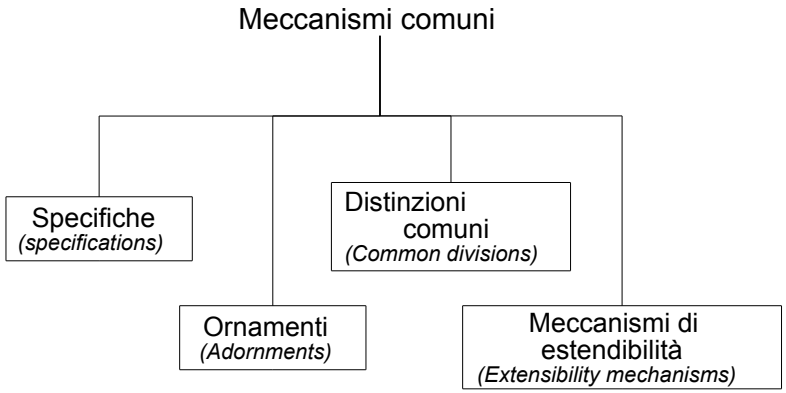
Regole

- Regole per specificare come deve essere “costruito” un modello *ben formato* (*well-formed*)
- Un modello (o parte del modello) *well-formed* è un modello *semanticamente consistente*, ossia un modello (o parte del modello) che rispetta tutte le *regole semantiche* e *sintattiche* che gli vengono applicate.

Regole

- UML permette di specificare regole semantiche per
  - ***Names,*** per entità, relazioni, diagrammi
  - ***Scope,*** il contesto che dà uno specifico significato ad un nome
  - ***Visibility,*** come i nomi possono essere visti ed usati da altri
  - ***Integrity,*** come le entità sono relazionate propriamente e consistentemente
  - ***Execution,*** per eseguire o simulare un modello dinamico

Meccanismi comuni di UML



Meccanismi comuni di UML

- *Specifiche*: *per ciascun elemento oltre la notazione grafica è fornita una specifica testuale*
- Ornamenti: *dettagli rappresentati graficamente o testualmente che sono aggiunti ai simboli base per rappresentare entità*
- Distinzioni comuni: *meccanismi per distinguere tra astrazioni e loro istanze*

Meccanismi comuni di UML

- *Meccanismi di estendibilità* includono
  - *Stereotipi (stereotypes): estendono il vocabolario di UML permettendo la definizione di un nuovo elemento di modellazione basato su uno esistente*
  - *Valori etichettati (tagged values): estendono le proprietà di un elemento permettendo di creare nuove informazioni nella specifica di quell'elemento*
  - *Vincoli (constraints): estendono la semantica di un elemento consentendo di aggiungere nuove regole*

Architettura

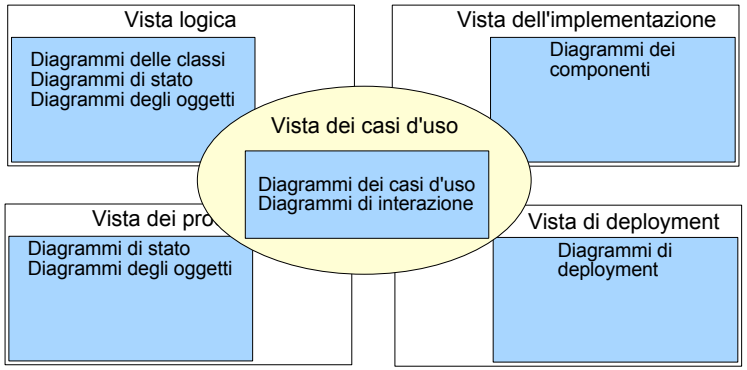
Quattro viste del sistema:

- logica: *fissa la terminologia del dominio del problema sotto forma di insiemi di classi e oggetti*
- dei processi: *modella i processi del sistema sotto forma di classi attive*
- di implementazione: *modella i file e i componenti che sono la base di codice del sistema*
- di deployment: *modella lo sviluppo fisico del sistema*

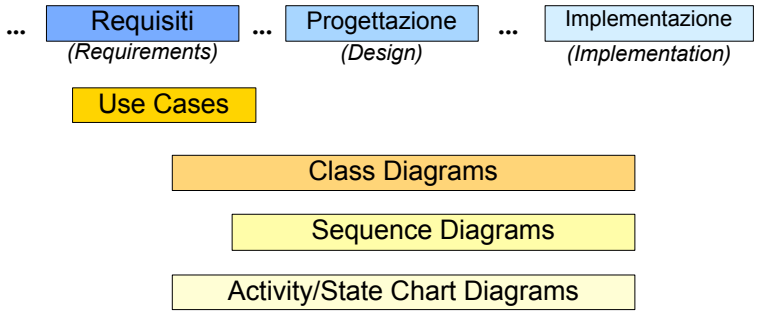
più una:

- dei casi d'uso: *fissa i requisiti di base del sistema*

Architettura



Diagrammi UML e ciclo di vita



I requisiti utente

- Per requisiti utente si intende ciò che il cliente vorrebbe poter fare con ciò che ci commissiona
- Questi requisiti vanno analizzati in modo da
  - poter separare le informazioni utili da quelle che non lo sono
  - e poterne raccogliere alcune in singole funzionalità
- Con i *casi d'uso* l'analista si propone di dare forma ai *desiderata* del cliente

Esempio

Cliente: “

- *vorrei vendere i manufatti che realizzo...*
- *non vorrei solo un mercato locale...*
- *mi piacerebbe che gli acquirenti potessero visionare un catalogo da cui scegliere...*
- *vorrei gestire gli ordini da qualunque posto perché viaggio molto...* ”

Cosa proponete?

Esempio: desiderata del cliente

Applicazione web per la vendita di oggetti su internet

Desiderata del cliente: “

- *vorrei avere la possibilità di creare un catalogo dei miei manufatti...*
- *vorrei un catalogo liberamente consultabile da chiunque...*
- *vorrei organizzare i manufatti raccogliendoli in serie...*
- *vorrei che gli interessati all'acquisto possano inviarmi un ordine, che io provvederò ad evadere previa una qualche forma di registrazione...* ”

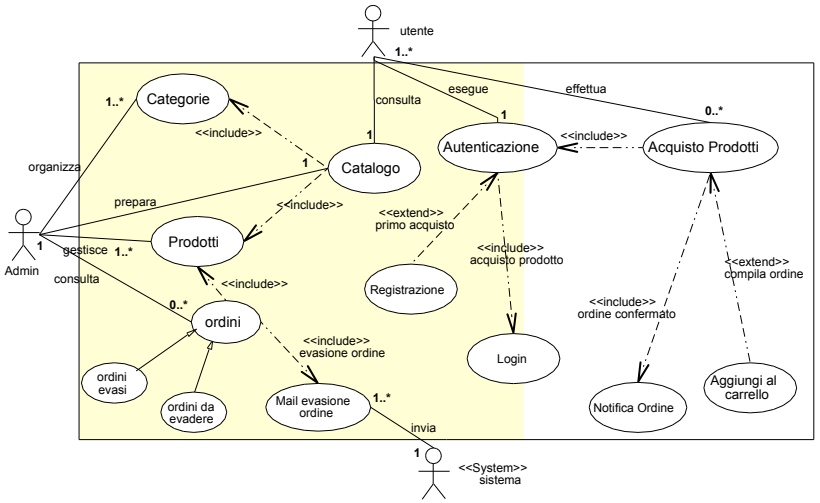
Esempio: formalizziamo tali desiderata

- *Ragionare sulle richieste e realizzare un modello di tale “realtà”: per farlo useremo gli Use Case*
- *Fissiamo le operazioni principali che si desumono dai requisiti:*
  - *si vuole gestire un catalogo in cui gli oggetti possano essere organizzati in categorie*
  - *e che deve essere liberamente consultabile dai clienti*
  - *l'acquirente deve registrarsi fornendo informazioni fondamentali (nome, cognome, indirizzo a cui spedire la merce,...)*
- *Cos'altro aggiungere?*

Esempio: formalizziamo tali desiderata

- *Aggiungiamo i seguenti requisiti che il cliente non ha esplicitamente richiesto:*
  - *quando il cliente compila un ordine, si notifica l'ordine avvenuto*
  - *una volta che l'ordine è stato evaso, si invia una e-mail di conferma dell'evasione al cliente*
- *Alla fine di questo processo, avremo disegnato qualcosa di simile al diagramma nella slide successiva*

Esempio di *Use Case*



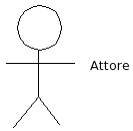
Elementi di uno *Use Case*

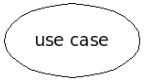
Le Use Case dice fondamentalmente

- chi interagisce con l'applicazione
- cosa fa con essa
- in che modo lo fa

Ovvero

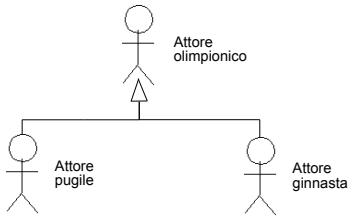
- chi sono gli attori del sistema
- quali sono le funzionalità fornite dal sistema
- che relazioni esistono
  - tra le funzionalità
  - tra le funzionalità e gli attori





Relazioni dello Use Case

- Associazione: *è l'unico legame posibile tra attori e casi d'uso. Ci dice che un attore sta partecipando ad un caso d'uso, cioè interagisce con la funzionalità espressa dal caso d'uso*
- Generalizzazione: *rappresenta una forma di specializzazione tra elementi dello Use Case. Ad esempio:*



Relazioni dello Use Case

- Estensione:
  - *applicabile solo a casi d'uso*
  - *indica che uno use case può includere (non necessariamente) nella sua funzionalità il comportamento di un altro use case*
  - *il comportamento dello use case che riceve la freccia può includere quello da cui la freccia parte*
  - *<<extend>>*

Relazioni dello Use Case

- Inclusione:
  - *applicabile solo a casi d'uso*
  - *l'elemento da cui parte la freccia include nella sua funzionalità quella dell'oggetto che riceve la freccia*
  - *lo include necessariamente*
  - *<<incude>>*

Analizziamo l'esempio

Nel diagramma Use Case relativo all'applicazione web:

- *Tra l'attore Admin e lo use case Prodotti vi è un associazione che indica che l'amministratore del sito può gestire una serie di prodotti (N.B.: la molteplicità 1..*)*
- *L'inclusione tra gli use case Ordini e Prodotti significa che un ordine contiene sempre dei prodotti*
- *L'inclusione tra gli use case Ordini e Mail evasione ordine dice che*
  - *per ogni ordine viene inviata una mail di conferma*
  - *tale mail viene inivata all'evasione dell'ordine*

Analizziamo l'esempio

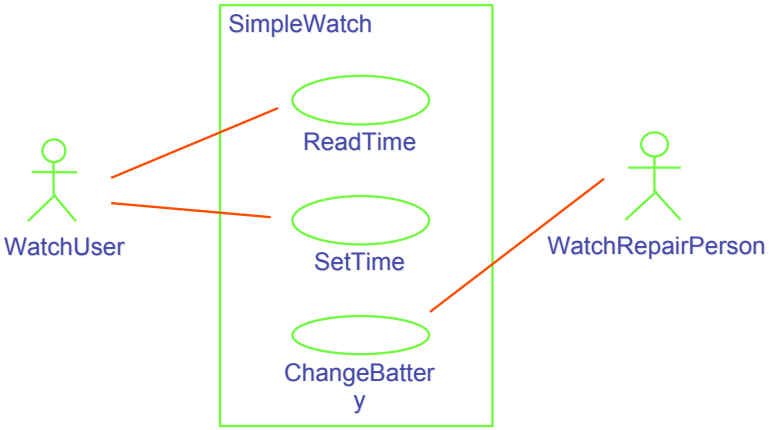
- *Due specializzazioni dello use case Ordini sono: Ordini evasi e Ordini da evadere:*
  - *è una relazione di generalizzazione*
  - *tutte le operazioni applicate su Ordini saranno applicate alle sue specializzazioni*
  - *non vale il contrario: possono esserci particolari funzionalità implementate da Ordini evasi che Ordini non implementa*

Analizziamo l'esempio

- Tra gli use case *Registrazione* e *Autenticazione* vi è la relazione di estensione:
  - *l'estensione non richiede necessariamente di includere la funzionalità di uno use case in un altro.*
  - *possiamo usarla per esprimere un comportamento che deve essere perseguito in una particolare situazione, ma non sempre*
  - *nell'esempio essa dice che non è necessario registrarsi ogni volta che ci si autentica: la registrazione è richiesta solo nel caso del primo acquitso.*

Use Cases: un esempio

Esempio: Use case diagram per un semplice orologio



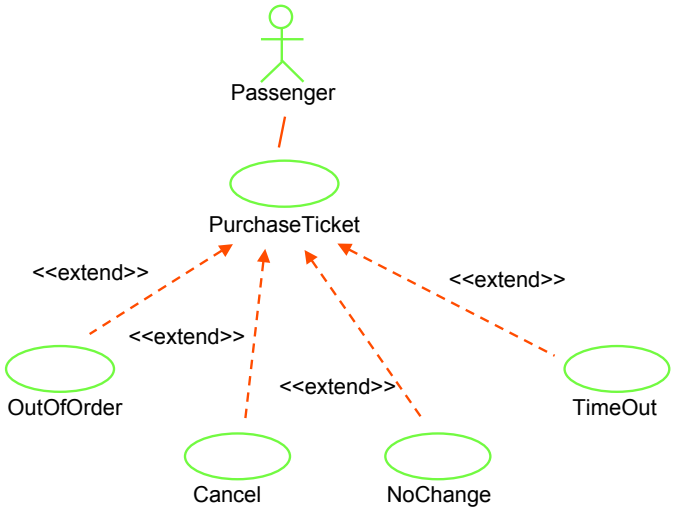
<<extend>> e <<include>>

<<extend>> : use case invocato in situazioni eccezionali
<<include>>: comportamento effettuato esternamente allo use case

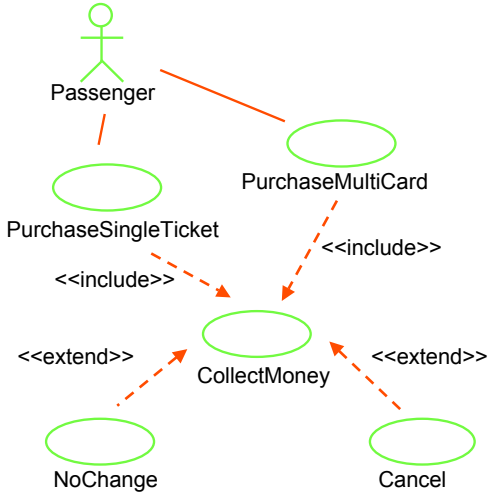
- Inclusione: si usa quando si vuole evitare la ripetizione della stessa descrizione all'interno di piu' casi d'uso
- Estensione: si usa per descrivere una variazione al comportamento normale

Arrow direction:
<<extend>> : ExceptionalCase ---> BaseCase
<<include>> : IncludingCase ---> IncludedCase

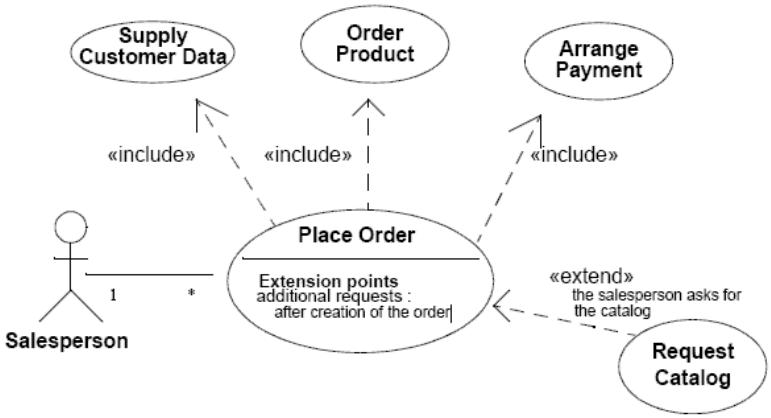
The <<extend>> Relationship



The <<include>> Relationship



Esempio: relazioni tra casi d'uso



Modellazione dei casi d'uso

Modellazione di caso d'uso:

Una *vista* che concentra l'attenzione su come viene percepito il comportamento di un sistema *sw* da parte di un utente esterno.

Le funzionalità di un sistema vengono suddivise in *transazioni* (casi d'uso) utili per ciascuna delle classi di utilizzatori (attori)

Specifiche del caso d'uso

- Ogni caso d'uso ha un nome e una specifica.
- La specifica è composta da:
  - Precondizioni *(entry conditions)*: condizioni che devono essere vere prima che il caso d'uso possa essere eseguito (sono vincoli sullo stato iniziale del sistema)
  - Sequenza degli eventi *(flow of events)*: i passi che compongono il caso d'uso
  - Postcondizioni *(exit conditions)*: condizioni che devono essere vere quando il caso d'uso termina l'esecuzione

Esempio

Nome del caso d'uso	**Caso d'uso: PagamentoIVA**
Identificatore univoco	**ID:** UC1
Gli attori interessati dal caso d'uso	**Attori:** Tempo, Fisco
Lo stato del sistema prima che il caso d'uso possa iniziare	**Precondizioni:** 1. Si è concluso un trimestre fiscale
I passi del caso d'uso	**Sequenza degli eventi:** 1. Il caso d'uso inizia quando si conclude un trimestre fiscale. 2. Il sistema calcola l'ammontare dell'IVA dovuta al Fisco. 3. Il sitema trasmette un pagamento elettronico al Fisco.
Lo stato del sistema quando l'esecuzione del caso d'uso è terminata	**Postcondizioni:** 1. Il Fisco riceve l'importo IVA dovuto.

Sequenza degli eventi

- La sequenza degli eventi elenca i passi che compongono il caso d'uso
- Comincia sempre con un attore che fa qualcosa per dare inizio al caso d'uso
- Un buon modo per iniziare la sequenza degli eventi è:
  1. Il caso d'uso inizia quando un <attore> <funzione>
- Ogni passo del caso d'uso dovrebbe avere la struttura:

  <numero> Il <qualcosa> <qualche azione>

Esempio: sequenza degli eventi

Pensiamo al caso d'uso **NuovoOrdine**. I seguenti passi della sequenza degli eventi sono corretti?

1. Incomincia quando si seleziona la funzione “ordina libro”
2. Il caso d'uso inizia quando il cliente seleziona la funzione “ordina libro” **sbagliato**
3. Vengono inseriti i dati del cliente **corretto**
4. Il cliente inserisce nel form il suo nome e indirizzo **sbagliato**
5. Il sistema verifica i dati del Cliente **corretto** **corretto**

Ramificazione di una sequenza

- UML usa parole chiave per esprimere *ramificazione*, *ripetizione* o *sequenze alternative*
- È bene non eccedere con le ramificazioni
- *parola chiave **Se***: indica una ramificazione della sequenza degli eventi
- *Sequenze alternative*: ramificazioni che non possono essere espresse utilizzando il ***Se***. Ad esempio ramificazioni dovute a condizioni che si possono verificare in un qualunque momento
- *Ripetizioni all'interno di una sequenza*:
  - *Parola chiave **Per** (**For**)*
  - *Parola chiave **Fintantoché** (**While**)*

Esempio

Caso d'uso: AggiornaCarrello
ID: UC2
Attori: Cliente
Precondizioni: 1. Il contenuto del carrello è visibile
Sequenza degli eventi: 1. Il caso d'uso inizia quando il Cliente seleziona un articolo nel carrello. 2. Se il Cliente seleziona “rimuovi articolo” 2.1 Il Sistema elimina l'articolo dal carrello. 3. Se il Cliente digita una nuova quantità 3.1 Il Sistema aggiorna la quantità dell'articolo presente nel carrello
Postcondizioni: 1. Il contenuto del carrello è stato aggiornato
Sequenza alternativa 1: 1. In qualunque momento il Cliente può abbandonare la pagina del carrello
Postcondizioni:

Modellazione dei casi d'uso

- Uno scenario e' una sequenza di passi che descrivono l'interazione tra un utente ed un sistema
- Un caso d'uso e' un insieme di scenari legati da un obiettivo comune
- Gli *scenari non hanno ramificazioni*, quindi ogni percorso della *sequenza degli eventi* genera uno scenario distinto
- Nello scenario principale succede tutto quello che ci si aspetta e si desidera dal sistema, senza errori, interruzioni o deviazioni
- Ogni *caso d'uso* può anche avere scenari secondari: sono i percorsi alternativi per attraversare la sequenza degli eventi

Modellazione dei casi d'uso

- La *specifica del caso d'uso* deve contenere lo *scenario principale* e l'elenco degli *scenari secondari*, riportato in un'apposita sezione
- Ogni *scenario secondario* deve essere riconducibile al proprio *caso d'uso*
- Gli *scenari secondari* all'interno della loro sequenza di eventi possono anche fare riferimento allo *scenario principale*

Individuare gli scenari secondari

Possono essere individuati analizzando lo *scenario principale*:

A ogni passo della sequenza potrebbero presentarsi:

•Ramificazioni del percorso

•Errori o eccezioni

•Interruzioni della sequenza: eventi che possono capitare in qualunque momento

Ciascuno di questi può dare origine a un nuovo *scenario secondario*

Quanti scenari?

Esistono due strategie per limitare il numero degli scenari:

•Scegliere gli *scenari secondari* più importanti e documentare solo quelli

•Se esiste un insieme di *scenari secondari* molto simili, documentarne uno solo come esempio e, se necessario, aggiungere delle annotazioni per spiegare come gli altri scenari differiscano dall'esempio.

Ricordiamo che la descrizione dei casi d'uso e degli scenari deve eservire per capire il comportamento desiderato del sistema

Quando applicare la modellazione dei casi d'uso?

I casi d'uso sono il miglior modo di fissare i requisiti se:

•Il sistema è dominato da requisiti funzionali

•Il sistema fornisce diverse funzionalità a molti tipi di utente (esistono molti attori)

•Il sistema ha molte interfacce con altri sistemi (esistono molti attori)

Ricordiamo che

•Requisiti funzionali: descrivono cosa debba fare il sistema

•Requisiti non-funzionali: descrivono proprietà o vincoli specifici che il sistema deve rispettare (prestazioni, affidabilità, ecc)

Scenari: negozio on line

- **Actors**: cliente, sistema
- **Preconditions**:
  - il cliente accede alla pagina Web
- **Basic Course**:
  1. Il cliente naviga nel catalogo e seleziona degli articoli
  2. Il cliente si avvia alla "cassa"
  3. Il cliente indica le informazioni relative alla spedizione
  4. Il sistema presenta il prospetto
  5. Il cliente inserisce I dati della carta di credito
  6. Il sistema autorizza l'acquisto
  7. Il sistema conferma l'acquisto
- **Alternative Course**:
  - carta di credito non valida….
  - cliente abituale → pagina personalizzata…

Scenari: cambio di volo

- **Actors**: viaggiatore, client account db, airline reservation system
- **Preconditions**:
  - Il viaggiatore ha fatto login sul sistema e ha selezionato l'opzione 'cambio itinerario di volo'
- **Basic Course**:
  - Il sistema trova l'account utente e l'itinerario di volo dal database degli account utenti
  - Il sistema chiede al viaggiatore di scegliere la tratta che vuole cambiare; il viaggiatore sceglie la tratta
- **Alternative Course**:
  - Se non ci sono voli disponibili allora…

Scenari: update benefits

- **Actors**: employee, employee account db, healthcare plan system, insurance plan system
- **Preconditions**:
  - Employee has logged on to the system and selected 'update benefits' option
- **Basic Course**:
  - System retrieves employee account from employee account db
  - System asks employee to select medical plan type; **include** (UpdateMedicalPlan)
  - System asks employee to select dental plan type; **include** (UpdateDentalPlan).
- **Alternative Course**:
  - Employee requests reimbursement option

Perche' utilizzare i diagrammi di use case?

- Per modellare i requisiti di un utente
- Per modellare gli scenari di testing
- Per derivare modelli strutturali e comportamentali del sistema da realizzare

Sommario

- UML è utile per la modellazione di sistemi software complessi
- UML permette la specifica di sistemi software in modo indipendente dall'implementazione
- La modellazione strutturale specifica uno "scheletro architetturale" che può essere raffinato ed esteso da strutture addizionali e specifiche comportamentali
- La modellazione dei casi d'uso specifica i *requisiti funzionali* di un sistema software

Per ulteriori informazioni

- Web:
  - UML 1.4 RTF: www.celigent.com/omg/umlrtf
  - OMG UML Tutorials: www.celigent.com/omg/umlrtf/tutorials.htm
  - UML 2.0 Working Group: www.celigent.com/omg/adptf/wgs/uml2wg.htm
  - OMG UML Resources: www.omg.org/uml/
- Email
  - uml-rtf@omg.org

Riferimenti

- [UML 1.5] *OMG UML Specification* v. 1.5.
- [RJB 99] Rumbaugh, Jacobson, Booch, *The UML Reference Manual*, AW 1999
- [AN02] Arlow, Neustadt, *UML e Unified Process*, McGraw-Hill, 2002
- [D04] Dattilo, *UML: Use Case,* `http://www.devspy.com/Art/Misc/Art.aspx?area=tech&id=00036`, 2004