ANALISI E PROGETTAZIONE OBJECT ORIENTED

UNIFIED MODELLING LANGUAGE (UML)

a cura di

Giacomo PISCITELLI
Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari

Questi appunti sono ricavati da una monografia redatta da Lorenzo Saladini e pubblicata dalla Scuola Superiore della Pubblica Amministrazione in un volume su "Sistemi informativi per la Pubblica Amministrazione: Metodologie e Tecnologie" realizzato a cura di Carlo Batini e Gaetano Santucci.
Il testo integrale è ricavabile dal sito dell'Autorità Italiana per la Pubblica Amministrazione (AIPA): http://www.aipa.it/servizi[3/pubblicazioni[5/monografie[2/sisteminfo/33.pdf

Bari giugno 2002

1. Introduzione

Vengono presentati alcuni degli elementi necessari al corretto sviluppo di sistemi informativi, secondo una metodologia orientata agli oggetti.

Tali elementi [**Larman97**] sono:

- un **metodo** che consente di descrivere il sistema da sviluppare organizzando tale descrizione per mezzo di oggetti;
- un **linguaggio** che indica il flusso di lavoro nel corso delle fasi del progetto di sviluppo, specificando quando produrre determinati modelli.

In particolare vengono di seguito presentati i fondamenti della progettazione ad oggetti, rimandando quindi all'UML come linguaggio di modellazione.

1.1. Lo sviluppo di sistemi

Lo sviluppo di un sistema informatico è un processo complesso. Gli elementi critici di tale processo sono: la comprensione del dominio di interesse, i problemi di gestione del processo di sviluppo e la capacità di descrivere e controllare il comportamento del sistema sviluppato.

Ciascun sistema informatico viene costruito per risolvere un problema nell'ambito di un certo dominio di interesse. Tali problemi presentano spesso numerosi elementi che accrescono la loro complessità, come la presenza di numerose esigenze spesso in contrapposizione fra di loro e la presenza di requisiti non funzionali, quali quelli di usabilità, prestazioni, costo e affidabilità. Inoltre accade spesso che gli utilizzatori del sistema e i suoi progettisti parlino un linguaggio completamente differente, così che i primi abbiano difficoltà ad esprimere con precisione le proprie necessità, mentre i secondi non riescano a spiegare correttamente cosa si apprestano a realizzare.

Questa difficoltà di comprensione è difficilmente attenuata da specifiche prolisse espresse in forma testuale, mentre di maggiore utilità possono essere modelli del sistema espressi in un linguaggio formale o prototipi del sistema stesso.

I sistemi informatici inoltre possono avere una struttura interna complessa ed essere composti da migliaia di moduli, ciascuno dei quali contiene migliaia di linee di codice.

La produzione di tali sistemi richiede il contributo di numerosi sviluppatori, il cui lavoro deve essere gestito, coordinato e integrato per arrivare al prodotto finale: la comunicazione fra i diversi sviluppatori è un elemento di successo importante nello sviluppo del sistema, e la descrizione del sistema per mezzo di modelli formali può portare un contributo significativo in tale senso.

Un sistema software è un sistema che può assumere uno fra un determinato numero di stati: tale numero è però incredibilmente alto a causa dell'esplosione combinatoria derivante dal numero di oggetti che compongono il sistema.

La complessità *strutturale* del sistema induce quindi una complessità *comportamentale*. Di conseguenza un sistema informatico può eseguire transizioni di stato del tutto inattese in conseguenza di un input esterno non previsto, venendosi a trovare in uno stato di errore, uno stato a partire dal quale il successivo comportamento del sistema risulta imprevedibile.

La risposta a questo problema è nella decomposizione del sistema in componenti più piccole la cui descrizione possa essere raffinata separatamente dalle altre, e il cui comportamento complessivo possa essere descritto in termini semplici e precisi a partire dal comportamento delle singole parti.

Nell'approccio tradizionale la decomposizione di un sistema è realizzata da un punto di vista funzionale: l'elemento atomico di tale decomposizione è la funzione o procedura e ciascuna funzione può essere decomposta in altre più semplici. L'attenzione del progettista del sistema si concentra quindi sul flusso di controllo e sulla decomposizione algoritmica: tale modalità di decomposizione tende a produrre sistemi difficilmente manutenibili, con un grado di resistenza

basso verso l'evoluzione dei requisiti.

1.2. Modellazione Object-Oriented

Una modalità alternativa di decomposizione è quella di un sistema in oggetti e classi; in questo contesto un oggetto nel sistema corrisponde generalmente ad un concetto presente nel dominio del problema o nello spazio della soluzione, mentre una classe corrisponde ad un insieme di oggetti simili. Ogni oggetto è caratterizzato da un'identità, da uno stato e da un comportamento.

I vantaggi che una decomposizione ad oggetti presenta rispetto ad una funzionale si possono così riassumere:

- conduce allo sviluppo di sistemi più piccoli e semplici perché consente di applicare meglio il riutilizzo di componenti preesistenti;
- consente di descrivere il sistema in un modo più comprensibile per gli utenti finali e più vicino al dominio applicativo nel quale il sistema si colloca;
- conduce alla produzione di sistemi più resistenti alle modifiche.

Un modello object-oriented [**Booch94**] è caratterizzato dai seguenti elementi costitutivi:

- ***astrazione di classificazione***, la capacità di descrivere un oggetto mettendo a fuoco le caratteristiche più importanti e ignorando le altre, per mezzo del costrutto di classe;
- ***incapsulamento***, la possibilità di nascondere i dettagli della implementazione di un oggetto;
- ***modularità***, la possibilità di decomporre un sistema in un insieme di elementi caratterizzati da un basso grado di accoppiamento e da un elevata coesione interna;
- ***ereditarietà***, la capacità di specificare una gerarchia fra le classi definite;
- ***aggregazione***, la possibilità di definire un oggetto come composto da altri oggetti.

L'astrazione di classe consente di descrivere oggetti con caratteristiche simili per mezzo del costrutto di classe: una classe rappresenterà solamente le caratteristiche desiderate per gli oggetti che la compongono.

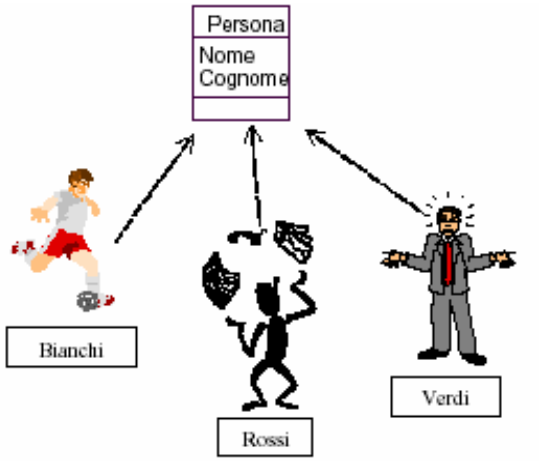


Figura 1 - Classificazione

Un esempio ereditarietà è quella mostrata in figura fra le classi IMPIEGATO e DIRIGENTE: la seconda classe può ereditare dalla prima caratteristiche strutturali (nome, cognome, ecc.), comportamentali (la capacità di svolgere un compito assegnato) e di relazione con altre classi (entrambi fanno parte di un'organizzazione).

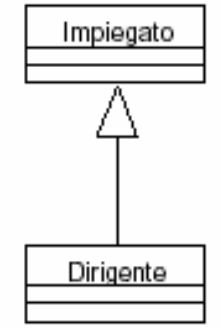


Figura 2 - Ereditarietà

La relazione esistente fra un IMPIEGATO e il GRUPPO in cui lavora costituisce un esempio di aggregazione: il gruppo risulta definito come oggetto composto da un certo numero di impiegati.

Figura 3 - Aggregazione

1.3. Linguaggi di modellazione

La realizzazione di un modello del sistema che si intende sviluppare prima della sua costruzione o della sua ristrutturazione è essenziale così come lo è il progetto per la costruzione di un edificio.

L'importanza di un modello e il grado di dettaglio necessario crescono proporzionalmente con la complessità dell'oggetto che si intende costruire. Una cuccia per un cane può essere costruita da una sola persona senza la necessità di descrivere in alcun modo l'obiettivo che si intende raggiungere. Una casa avrà bisogno almeno di un progetto di massima per verificare le necessità per cui viene costruita ed eventualmente affidare una parte dei lavori a ditte specializzate. La costruzione di un palazzo di molti piani avrà bisogno di un progetto dettagliato, un piano di realizzazione e di spesa, una chiara visione degli obiettivi da raggiungere e del modo in cui raggiungerli [**Booch98**].

Un modello consente di semplificare la realtà da descrivere mettendo in evidenza alcuni aspetti e scartandone altri: lo stesso sistema può essere modellato a diversi livelli di astrazione in diversi momenti del suo sviluppo. Una vista semplificata del sistema da sviluppare consentirà di visualizzare e comprendere meglio gli aspetti essenziali del sistema.

Si desidera qui mettere in evidenza l'importanza del modello fra gli elementi che compongono il progetto di un sistema. Un modello infatti assolve a compiti diversi:

- costituisce un elemento di comunicazione fra le persone coinvolte nella costruzione del sistema
- permette di verificare le caratteristiche del sistema da costruire
- permette di confrontare le caratteristiche del sistema con standard di costruzione
- facilita la stima dei tempi e dei costi necessari a completare la costruzione del sistema.

1.4. UML e modellazione OO

UML (Unified Modelling Language) è un linguaggio per descrivere, specificare e documentare i prodotti del processo di sviluppo del software. Esso può essere utilizzato per sistemi di tipo e grandezza differenti ed è caratterizzato da un alto potere espressivo.

UML non specifica un processo per la modellazione e intende anzi essere indipendente dal processo di sviluppo all'interno del quale è utilizzato. I vantaggi di tale divisione fra linguaggio e modellazione e processo di sviluppo sono principalmente:

- la maggiore probabilità di accettazione di un linguaggio di modellazione rispetto ad un processo di sviluppo;
- la sostanziale invarianza di un linguaggio di modellazione rispetto agli elementi che caratterizzano un sistema software, differentemente da quanto avviene per il processo di sviluppo che può essere profondamente influenzato da tali fattori.

UML fornisce il potere espressivo necessario a modellare un sistema informatico da una prospettiva object-oriented: UML infatti consente di rappresentare le classi e gli oggetti che le compongono, lo stato di tali oggetti e il loro comportamento. UML inoltre permette di mantenere diverse viste di uno stesso sistema, descrivendo gli oggetti che lo compongono da differenti punti di osservazione (strutturale, comportamentale) e a diversi livelli di astrazione.

UML è il risultato delle sforzo di unificazione condotto a partire dal 1994 per individuare un linguaggio standard nella modellazione orientata agli oggetti. Il linguaggio è stato sviluppato da Rational Software come il successore ai linguaggi di modellazione specificati da Booch, Jacobson (OOSE) e Rumbaugh (OMT) pur attingendo alcuni concetti anche da altri linguaggi di modellazione.

La specifica del linguaggio UML ha raggiunto i seguenti obiettivi:

- ha stabilito un linguaggio di modellazione visuale, fornendo un mezzo di comunicazione standard per la modellazione object-oriented;
- ha individuato una base formale per la definizione del significato dei modelli supportati;
- ha favorito la proliferazione di strumenti di sviluppo e componenti per la realizzazione di sistemi object-oriented;
- ha specificato un modello visuale non legato ad un particolare linguaggio o processo di sviluppo;
- ha integrato alcuni fra gli approcci più interessanti alla modellazione object-oriented.

UML è stato proposto per la standardizzazione da parte dell'OMG all'inizio del 1997 e, dopo un processo di revisione che ha coinvolto le più importanti fra le industrie software, ha ottenuto la approvazione nella sua versione 1.1. UML è ora di dominio pubblico e grazie al processo di standardizzazione in cui è stato coinvolto, ai contributi forniti dalle industrie e dal settore scientifico esso potrà avere una ampia accettazione portando alla nascita di strumenti di modellazione, sviluppo e integrazione basati su di un modello di riferimento comune.

1.5. Schema del processo di sviluppo

Un **processo di sviluppo iterativo** si concilia con la metodologia object-oriented. Il processo iterativo riduce in modo significativo i rischi di progetto negli stadi iniziali dello sviluppo e permette di ottenere un maggiore controllo sul sistema prodotto.

Con un processo di sviluppo iterativo il sistema non viene rilasciato in un'unica versione alla fine del progetto, ma viene sviluppato e rilasciato in porzioni, ciascuna delle quali contiene un sottoinsieme progressivamente crescente di funzionalità.

Lo sviluppo risulta così suddiviso in **iterazioni**: durante ogni iterazione si realizza una porzione del sistema rispondente ad un determinato sottoinsieme delle specifiche. Ogni iterazione si articola a sua volta in analisi, progetto, codifica e sperimentazione. Al termine di ogni iterazione è prevista una verifica di quanto sviluppato con i futuri utenti ed i responsabili del progetto.

Nel corso del processo di sviluppo vengono prodotti diversi modelli del sistema utilizzando il linguaggio di modellazione prescelto. La **prospettiva** in cui tali modelli sono prodotti cambierà durante il corso dello sviluppo: essa sarà infatti inizialmente orientata all'analisi (definire formalmente cosa sviluppare) quindi al progetto (definire formalmente come svilupparlo, in termini di struttura e comportamento del software) e infine all'implementazione (come realizzare, per

mezzo degli strumenti di sviluppo individuati le soluzioni progettuali identificate nella fase precedente).

In linea generale, i diagrammi potranno essere disegnati con varie prospettive indipendentemente dal loro tipo. Infatti il tipo di diagramma dipende dal particolare aspetto del sistema (utilizzo, struttura, comportamento) che si desidera descrivere, mentre la prospettiva utilizzata per disegnarlo dipende dal grado di astrazione (concettuale, specifica, implementazione) con cui si è in grado di descriverlo allo stadio corrente del progetto.

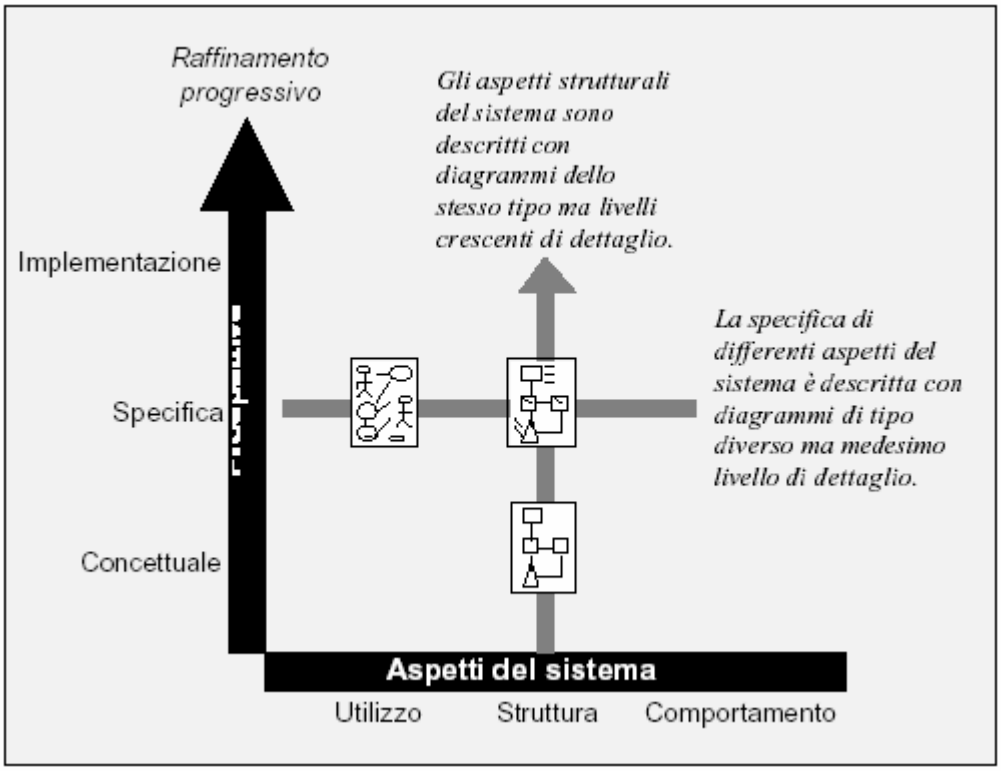


Figura 4 - Prospettiva di utilizzo di un modello nei diversi momenti del processo di sviluppo

Per utilizzare in modo efficace lo sviluppo iterativo è necessario adottare:

- un linguaggio di modellazione che garantisca la possibilità di accrescere il progetto del sistema attraverso il riutilizzo di componenti sviluppati precedentemente e sia resistente a successivi raffinamenti delle specifiche;
- un linguaggio di modellazione in grado di fornire viste del sistema da sviluppare a differenti livelli di astrazione, favorendo la costruzione di una nuova vista a partire da quelle già sviluppate.

Il primo requisito è soddisfatto dai linguaggi di modellazione orientati agli oggetti perché, come accennato, questi offrono i migliori strumenti per costruire modelli flessibili e modificabili nel tempo attraverso i meccanismi dell'ereditarietà, dell'aggregazione e dell'incapsulamento. I primi due consentono di costruire nuovi oggetti a partire da quelli preesistenti, sfruttando le relazioni *is-a* e *part-of*. Il secondo consente di modificare l'implementazione di un determinato oggetto, ad esempio per migliorare le prestazioni di un'operazione che esso definisce, senza per questo richiedere la modifica delle classi che utilizzano tale operazione, sfruttando il meccanismo dell'incapsulamento.

Il secondo requisito è soddisfatto dal linguaggio UML, che integra diversi modelli, permettendo di descrivere diversi aspetti dei componenti del sistema (struttura , comportamento, architettura) a diversi livelli di astrazione (concettuale, logico, implementativo).

1.6. Un primo esempio

Viene di seguito proposto un primo esempio di specifica di un sistema per la prenotazione degli esami universitari. Tale esempio è utile per dare una prima vista generale alle caratteristiche del linguaggio di modellazione UML e identificare una sequenza di modelli prodotti nel corso del processo di sviluppo.

In questo esempio sono descritti alcuni strumenti per la cattura dei requisiti e l'analisi del sistema, come i diagrammi **use case** o il modello concettuale del dominio rappresentato per mezzo di un **class diagram**, e altri utili nella fase di progetto, come l'**interaction diagram** e il modello di progetto delle classi del sistema descritto per mezzo di un **class diagram**.

1.6.1. Use cases

Un primo *use case* per il sistema indicato è il seguente:

Use case:	Prenotazione di un esame universitario
Attori:	Studente
Descrizione:	Lo use case inizia quando lo studente fornisce al sistema la propria matricola e il proprio codice fiscale per identificarsi. Il sistema verifica che lo studente sia regolarmente iscritto e quindi fornisce un elenco degli esami che lo studente può sostenere. Lo studente seleziona un esame, quindi il sistema suggerisce le prossime date per sostenere l'esame. Lo studente seleziona la data prescelta e conferma la prenotazione. Lo use case termina quando il sistema fornisce un numero di registrazione allo studente. Il sistema può non riconoscere lo studente fra quelli iscritti regolarmente. Lo studente può interrompere la registrazione senza selezionare un esame o una data.

*Tabella 1- Descrizione testuale di uno **use case***

Lo use case sopra introdotto può essere in parte descritto per messo di uno **use case diagram.**

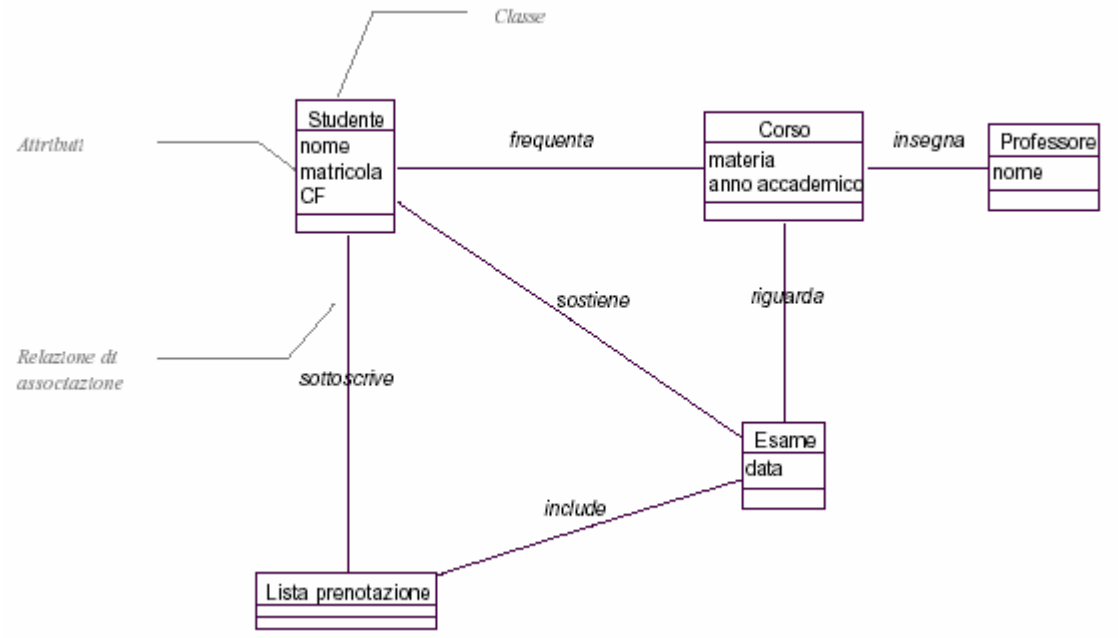


*Figura 5 - Diagramma **use case***

Lo use case illustrato descrive una tipica interazione fra l'utente del sistema ed il sistema stesso per lo svolgimento di un compito specifico. Esso costituisce un importante strumento di comunicazione fra committente e sviluppatore nel corso della elaborazione del progetto e consente una migliore comprensione dei processi da automatizzare.

1.6.2. Modello concettuale

Il modello concettuale è un altro importante prodotto risultante dall'analisi del dominio: mentre gli use cases descrivono i requisiti funzionali del sistema, il modello concettuale illustra il modo in cui gli oggetti vengono classificati e consente di mettere a fuoco il significato dei termini usati nella realtà di interesse. Esso permette di identificare i concetti, gli attributi e le associazioni considerate importanti nel dominio e può essere descritto per mezzo di diagrammi di classi quale quello che segue.

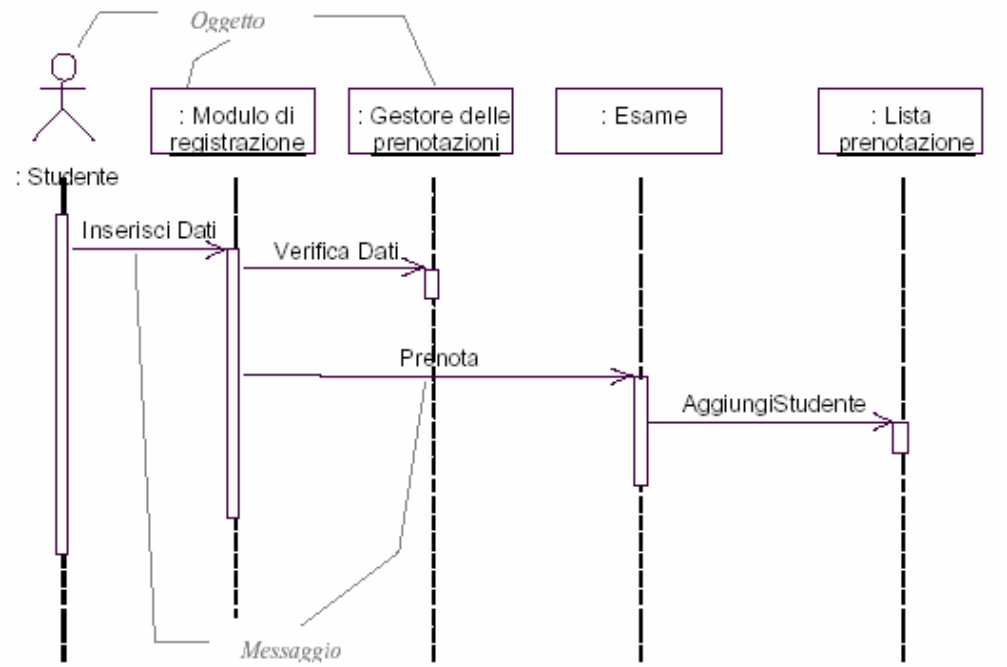


*Figura 6 - Modello concettuale del dominio descritto per mezzo di un **class diagram***

Si noti come il modello concettuale sopra descritto non descrive componenti software, ma piuttosto rappresenta concetti del mondo reale all'interno del dominio di interesse.

1.6.3. Diagrammi di interazione

Un diagramma di interazione (**interaction diagram**) descrive il modo in cui gruppi di oggetti collaborano fra loro per svolgere un determinato compito. Tipicamente un diagramma di interazione descrive il flusso di eventi rappresentato in uno use case. Esso permette di assegnare delle responsabilità agli oggetti, mostrando come essi interagiscono fra di loro.

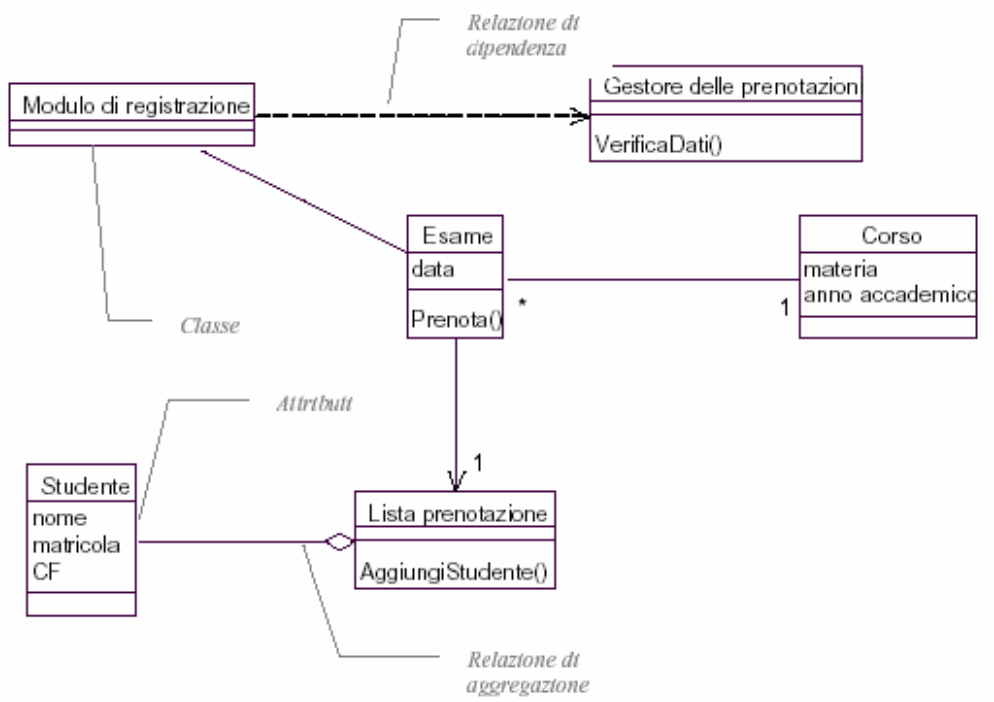


*Figura 7 – Flusso di messaggi fra oggetti del sistema illustrato per mezzo di un **sequence diagram***

1.6.4. Progetto e diagrammi delle classi

Dopo aver descritto gli oggetti del sistema ed il modo in cui essi collaborano per la realizzazione di uno use case, per mezzo di un diagramma di interazione, è ora possibile specificare in dettaglio quali classi comporranno il sistema software. Queste informazioni possono essere catturate per mezzo di una diagramma delle classi di progetto.

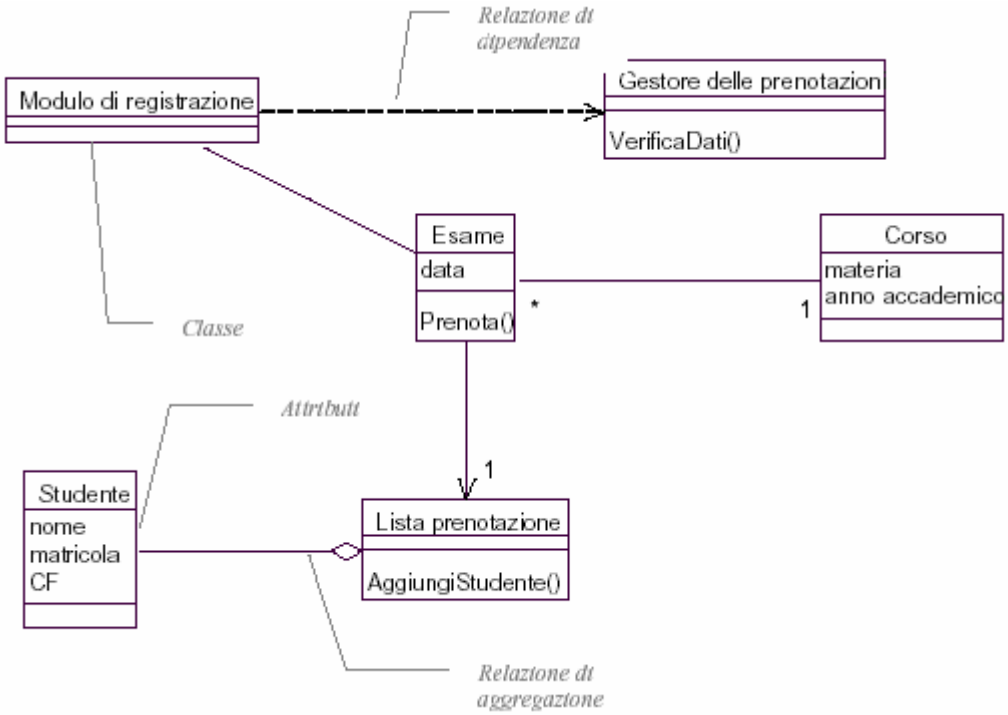


*Figura 8 – Progetto dei componenti software del sistema illustrato per mezzo di un **class diagram***

Questo diagramma illustra in che modo i componenti del sistema software sono connessi e quali metodi realizzano.

2. Il processo di sviluppo

Il processo di sviluppo di un sistema software è costituito da un insieme di passi che consentono di muoversi dalle attività di analisi dei requisiti fino all'implementazione e alla consegna del sistema. Obiettivo di tale processo è quello di sviluppare il sistema in oggetto secondo le esigenze del committente.

Nel classico approccio allo sviluppo dei sistemi il modello seguito è quello a **cascata** in cui si distinguono diverse fasi ordinate linearmente.

Il modello a cascata presenta però notevoli svantaggi. Nel mondo reale infatti i progetti raramente seguono il flusso sequenziale proposto dal modello ed è spesso difficile per il cliente dichiarare tutti i requisiti in modo esplicito nelle prime fasi del progetto. Inoltre il cliente dovrà attendere le ultime fasi del ciclo di vita per poter lavorare con il sistema e verificare quindi che le sue aspettative sono state soddisfatte. Questa situazione può rivelarsi assai penalizzante nello sviluppo di sistemi di grandi dimensioni: infatti tutti i problemi che si riscontreranno nel corso della verifica (**test**), dovuti ad errori commessi in una delle precedenti fasi, comporteranno correzioni che dovranno essere propagate attraverso tutti i precedenti passi che compongono il ciclo di vita. In tal modo potranno aumentare in modo significativo il tempo e i costi per la consegna del sistema, così come il rischio di insuccesso del progetto [**Standish95**].

Di qui la necessità di far uso una modello di sviluppo che consenta un più adeguato controllo dell'evoluzione delle fasi. La modellazione viene perciò effettuata ricorrendo all'UML.

3. Modellazione per mezzo dell'UML

Allo scopo di comprendere come utilizzare l'UML si introduce una descrizione informale del linguaggio; tale descrizione include gli elementi atomici del linguaggio e le regole che permettono di comporre tali elementi. Quindi sono descritti i diversi diagrammi prodotti utilizzando il modello.

3.1. Atomi dell'UML

Gli atomi del linguaggio UML sono gli **elementi**, le **relazioni** fra gli elementi e i **diagrammi** che raggruppano elementi e relazioni di interesse.

Gli elementi si articolano a loro volta in:

- elementi strutturali, che descrivono la parte statica del modello;
- elementi comportamentali, che descrivono i suoi aspetti dinamici;
- elementi di raggruppamento, che suddividono un modello in gruppi di oggetti;
- elementi di annotazione, che consentono di inserire commenti all'interno del modello.

3.1.1. Elementi strutturali

Gli elementi strutturali sono elementi grafici che descrivono la parte più statica di un modello.

Un primo esempio di elemento strutturale è la CLASSE, che descrive un insieme di oggetti, con i medesimi attributi, operazioni e relazioni. Una classe è rappresentata per mezzo di un rettangolo all'interno del quale sono elencati il nome, gli attributi e le operazioni.

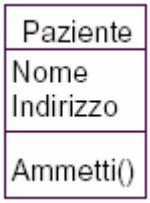


Figura 9 - Classe

Un secondo elemento strutturale è l'INTERFACCIA: esso definisce un insieme di operazioni che costituiscono un servizio offerto da una classe o da un componente.

L'interfaccia descrive quindi il comportamento di un elemento, così come visibile all'esterno, in modo completo o parziale. Un'interfaccia definisce un'insieme di operazioni in termini della loro segnatura, ma non implementa mai tali operazioni.

Graficamente l'interfaccia è resa per mezzo di un cerchio con il suo nome ed è generalmente collegata all'elemento che implementa le operazioni che essa definisce.ì

Figura 10 - Interfaccia

Un ulteriore elemento strutturale è lo USE CASE, una descrizione di una sequenza di azioni che il sistema è in grado di svolgere fornendo un risultato significativo per un particolare attore (si noti che per attore si intende un particolare tipo di classe). Uno use case è rappresentato con un ellisse che contiene il suo nome.

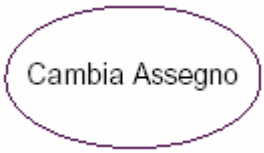


Figura 11 - Use case

Un COMPONENTE è un elemento strutturale che rappresenta una porzione del sistema e implementa una particolare interfaccia. Un componente è un modulo software, in forma di sorgente o eseguibile, e può essere il risultato del processo di sviluppo o essere prodotto all'esterno di tale processo. La rappresentazione grafica di un componente è la seguente.

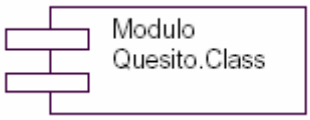


Figura 12 – Componente

Un NODO rappresenta una risorsa fisica che supporta l'esecuzione del sistema; esso può essere un PROCESSORE, e avere quindi una capacità computazionale, o un DISPOSITIVO.

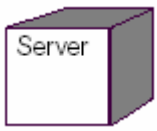


Figura 13 - Nodo

3.1.2. Elementi comportamentali

Gli elementi comportamentali rappresentano gli aspetti dinamici di un modello UML. Vi sono due tipi di elementi comportamentali.

L'INTERAZIONE descrive un insieme di messaggi scambiati fra oggetti in un determinato contesto per svolgere un compito.

Un'interazione è descritta per mezzo di messaggi scambiati, azioni svolte in risposta ai messaggi e relazioni fra gli oggetti che si scambiano tali messaggi. Un messaggio è descritto da una freccia con l'operazione che il messaggio invoca.

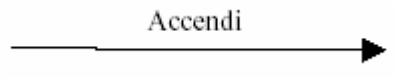


Figura 14 – Messaggio

Una MACCHINA A STATI descrive il comportamento di un oggetto per mezzo della sequenza degli stati che esso può assumere in risposta a determinati eventi. Una macchina a stati è descritta per mezzo di stati, transizioni da uno stato a un altro, eventi che scatenano una transizione e azioni svolte in corrispondenza a una transizione.

Uno stato è rappresentato come un rettangolo con bordi arrotondati che include un nome ed eventualmente un insieme di sottostati.

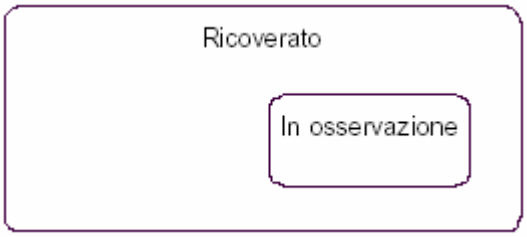


Figura 15 - Stato

3.1.3. Elementi di raggruppamento

Gli elementi di raggruppamento consentono di organizzare porzioni di un modello UML, contenitori di altri oggetti del modello. La funzione di raggruppamento è svolta dal PACKAGE. Esso può contenere oggetti strutturali, di comportamento o anche altri packages. Un package è un oggetto del modello cui non corrisponde alcun oggetto fisico, come invece accade per i componenti.

Un package è rappresentato da una cartellina con il suo nome ed eventualmente il suo contenuto.

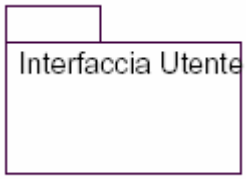


Figura 16 - Package

3.1.4. Relazioni

Le relazioni che possono essere rappresentate in UML sono:
1. dipendenza;
2. associazione;
3. generalizzazione;
4. realizzazione.

Una relazione di DIPENDENZA è una relazione semantica fra due elementi e indica che un cambiamento in uno dei due elementi può influenzare il significato dell'altro. Tale relazione è

rappresentata graficamente da una freccia tratteggiata.

Figura 17 - Dipendenza

L'ASSOCIAZIONE è una relazione semantica che collega due elementi strutturali; un particolare tipo di associazione è l'aggregazione, la relazione fra un oggetto e le sue parti. Un'associazione è rappresentata da una linea continua, con elementi grafici accessori che descrivono vincoli di cardinalità, nomi di ruolo e navigabilità.

Figura 18 – Associazione

Una GENERALIZZAZIONE rappresenta una relazione in cui istanze dell'elemento specializzato (o figlio) condividono le caratteristiche dell'elemento più generale (padre).

Il figlio eredita dal padre struttura, comportamento e relazioni. Tale relazione è resa per mezzo di una freccia.

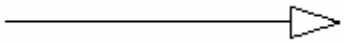


Figura 19 – Generalizzazione

Una REALIZZAZIONE è una relazione fra due elementi del modello in cui il primo elemento definisce un interfaccia che il secondo elemento implementa. Tale relazione è rappresentata per mezzo di una freccia tratteggiata.

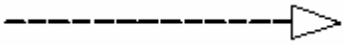


Figura 20 – Realizzazione

3.1.5. Diagrammi

Un DIAGRAMMA in UML è la rappresentazione grafica di un insieme di elementi collegati da relazioni. Un diagramma è una vista sul modello di un sistema e diversi diagrammi possono cogliere aspetti differenti di un medesimo sistema.

UML include nove diversi tipi di diagramma:
1. diagramma delle classi;
2. diagramma degli oggetti;
3. diagramma use case;
4. diagramma di sequenza;
5. diagramma di interazione;
6. diagramma di stato;
7. diagramma di attività;
8. diagramma dei componenti;
9. diagramma di dispiegamento.

3.1.6. Specifiche e adornment

Si consideri che l'UML non è un linguaggio unicamente grafico: per ogni elemento, oltre a una rappresentazione grafica esiste una specifica che descrive in forma testuale l'insieme delle caratteristiche dell'elemento. Per una classe ad esempio la specifica indica il nome, l'insieme degli attributi e delle operazioni, inclusa la segnatura, e altre caratteristiche. Di conseguenza in diversi momenti una classe può avere una rappresentazione grafica differente, in funzione degli elementi che si vuole mettere in evidenza, ma è sempre associata all'insieme completo delle specifiche.

UML offre quindi una notazione grafica semplificata per rappresentare i suoi elementi, anche allo scopo di facilitarne l'utilizzo. D'altra parte essa permette di visualizzare molti dettagli aggiuntivi per mezzo delle variazioni o *adornment*, elementi aggiuntivi della notazione che permettono di arricchire la rappresentazione di un modello. Ad esempio per la relazione di associazione sono definiti i seguenti adornment:

Adornment	Rappresentazione
Cardinalità	0..n
Navigabilità	
Ruolo	manager
Aggregazione semplice	
Aggregazione forte o composizione	

Tabella 2 - Adornment di un'associazione e loro rappresentazione

3.2. Regole dell'UML

Naturalmente non tutte le combinazioni degli elementi sopra descritti sono ammesse e significative.

UML prevede un insieme di regole che indicano il modo in cui gli atomi sopra descritti possono essere composti. Un modello dovrà rispettare tali regole per essere un modello *ben formato*. La specifica del linguaggio include la definizione di un meta-modello descritto in modo semi-formale per mezzo di:

- una ***sintassi astratta***, che descrive quali sono e come possono essere composti gli elementi del modello utilizzando un diagramma delle classi e la sua descrizione in linguaggio naturale;
- le ***regole di consistenza***, che esprimono dei vincoli ulteriori sui modelli che possono essere costruiti, espresse per mezzo del linguaggio formale OCL (Object Constraint Language) e del linguaggio naturale);
- la ***semantica***, che descrive il significato di un costrutto principalmente in linguaggio naturale.

Vediamo ad esempio alcune regole che determinano la costruzione di un modello ben formato per quanto riguarda il costrutto di classe.

La sintassi astratta del costrutto di classe può essere, in parte, descritta dal seguente meta-modello:

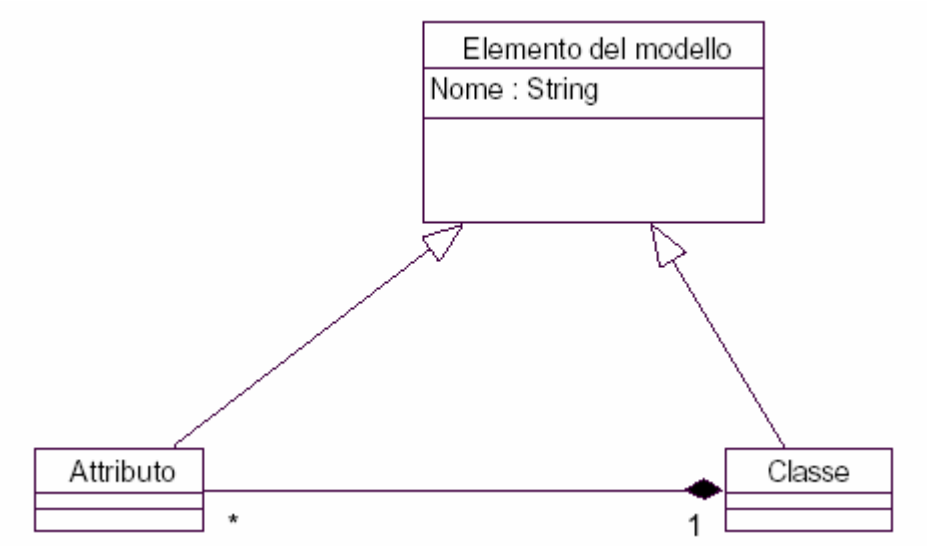


Figura 21 – Meta-modello per la Classe

Tale meta-modello indica che una classe è un elemento del linguaggio, e quindi possiede un nome, e che una classe può contenere un insieme di attributi, ciascuno dei quali è a sua volta un elemento del linguaggio.

Si noti che UML definisce alcuni aspetti della sintassi di un modello per mezzo dei suoi stessi costrutti.

Fra le regole di consistenza valide per la Classe c'è quella che indica che non possono esistere più attributi all'interno della medesima classe; la sua espressione in OCL è la seguente:

```
self.attribute->select (a | a.oclIsKindOf (Attribute))-
>forAll (p, q |
p.name = q.name implies p = q )
```

La semantica per la Classe stabilisce infine che ogni oggetto istanziato da una classe conterrà valori di attributo corrispondenti agli attributi definiti per la classe.

3.3. Modellare la struttura

UML consente di modellare la struttura del sistema per mezzo di diversi tipi di diagramma: class diagram, object diagram, packages.

Un class diagram mostra un insieme di classi e le loro relazioni e rappresenta il più comune diagramma nella modellazione object-oriented. Esso è utilizzato per mostrare gli aspetti statici del progetto del sistema.

Un object diagram mostra un insieme di oggetti e le loro relazioni e rappresenta una istantanea delle istanze degli elementi di un class diagram.

3.3.1. Class diagram

I **class diagram** sono un elemento centrale nella notazione UML e più in generale nella modellazione object-oriented.

Un class diagram visualizza un insieme di classi, le caratteristiche delle singole classi (quali attributi e le operazioni) e le relazioni che intercorrono fra le classi.

Un class diagram rappresenta la struttura statica di un sistema a diversi livelli di astrazione:

- un class diagram **concettuale** permette di specificare i concetti che fanno parte della realtà che si vuole modellare;
- un class diagram può essere utile per specificare il **progetto** del sistema, descrivendo i tipi che compongono il sistema software;
- un class diagram può rappresentare una specifica dettagliata delle classi del sistema da **implementare** in un determinato linguaggio.

Un **class diagram** è composto da classi, relazioni (di dipendenza, associazione, generalizzazione e realizzazione) e interfacce.
Nel class diagram di esempio che segue viene descritta una catena di negozi che effettua vendite di prodotti in un catalogo.
La Catena viene rappresentata come un aggregazione di Negozi, ciascuno dei quali è descritto per mezzo di un Nome e di un Indirizzo: la associazione di aggregazione fra Catena e Negozio indica che un negozio è logicamente (anche se non fisicamente) contenuto in una Catena. Le relazioni di associazione, come la relazione *utilizza* fra Negozio e Catalogo, rappresentano relazioni strutturali, ad indicare nell'esempio che un'istanza di Negozio utilizza un'unica istanza di Catalogo (che si trova presso l'ufficio vendite come indicato nella nota). La relazione *utilizza* lega una sola istanza di Negozio ed una sola istanza di catalogo, come indicato dalle cardinalità espresse nel diagramma.

Ad un Catalogo sono associati diversi Tipi di prodotto: si noti che l'attributo di navigabilità della relazione indica che è possibile, o più efficiente, ottenere un Tipo di prodotto a partire da un Catalogo, piuttosto che il contrario.

Tutte le volte che il Negozio effettua una Vendita essa può essere composta da più Voci, ciascuna delle quali può contenere più istanze di Prodotto. Si noti che fra i prodotti è definita la sottoclasse dei prodotti che hanno una scadenza per mezzo di una relazione di generalizzazione-specializzazione fra Prodotto e Prodotto con scadenza.

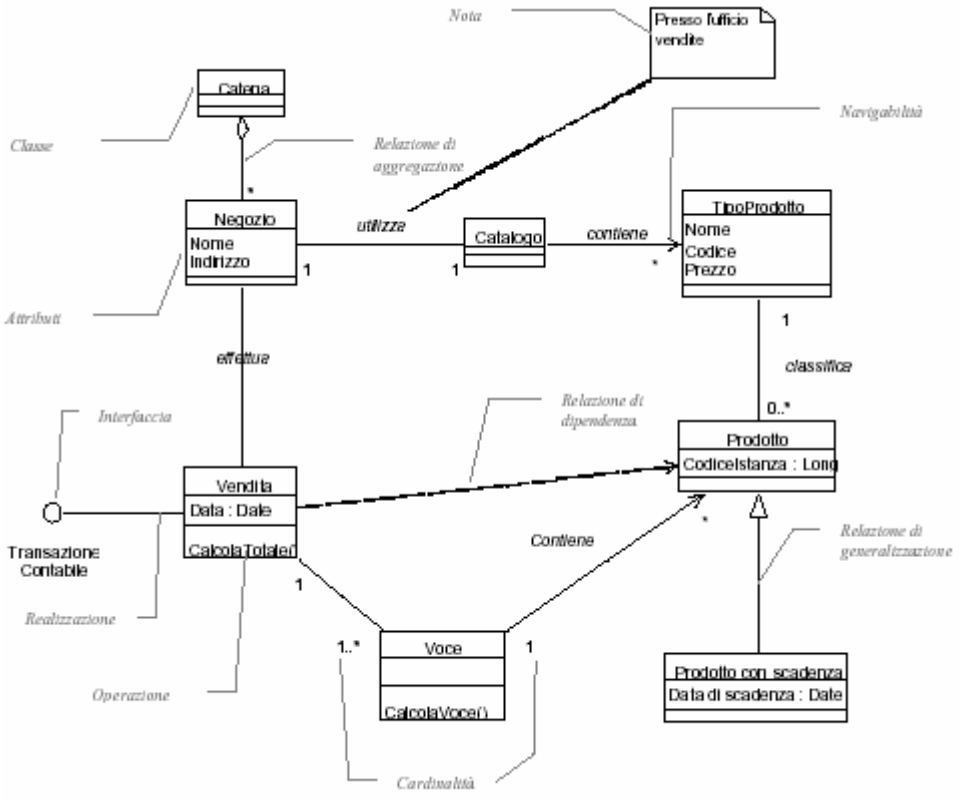


*Figura 22 - **Class diagram** per la rappresentazione di una catena di negozi*

I class diagram sono un elemento fondamentale per l'utilizzo di tutte le metodologie object-oriented e verranno utilizzati in diversi momenti del ciclo di vita di un progetto.

Si noti che la loro ricchezza di notazione richiede un utilizzo attento: non è importante può anzi essere dannoso utilizzare tutti i dettagli della notazione. Inoltre è importante utilizzare un class diagram al giusto livello di astrazione: per descrivere modelli concettuali nelle attività di analisi, per definire delle specifiche nel corso del progetto, per definire particolari tecniche di implementazione.

Infine si tenga presente che non è necessario modellare ogni aspetto del sistema che si intende sviluppare, mentre è utile piuttosto averne pochi, costantemente aggiornati e utilizzati spesso.

3.3.2. Packages

La specifica, la visualizzazione e la documentazione di un sistema di grandi dimensioni richiede la manipolazione di un grande numero di elementi del modello UML, quali classi, componenti, diagrammi e altri elementi. UML definisce il costrutto di PACKAGE per organizzare la descrizione del sistema in gruppi di oggetti. I package dovrebbero essere utilizzati per raggruppare elementi semanticamente vicini e che evolvono similmente nel corso dello sviluppo del sistema. Un package ben strutturato deve essere scarsamente accoppiato con altri packages e altamente coesivo, offrendo un accesso limitato ai suoi elementi interni.

Un package è caratterizzato da un nome e dagli elementi che esso contiene; per ciascuno di tali elementi può essere definita una visibilità. Un elemento può essere:
- privato, visibile solamente agli altri elementi contenuti nel package;
- protetto, visibile ai packages che specializzano il package che lo contiene;
- pubblico, visibile in tutto il modello.

Si supponga ad esempio che la classe Client dipenda dalla classe Server (perché accede ad uno dei suoi membri pubblici). Se le due classi sono nello stesso package ciascuna ha visibilità dell'altra.

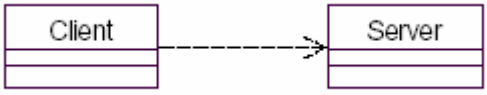


Figura 23 - Dipendenza nel medesimo package

Se invece le due classi appartengono a package diversi esse non hanno mutua visibilità a meno che non si verifichino le due condizioni:
- il package contenente Server lo abbia dichiarato pubblico;
- il package contenente Client importi l'altro server.

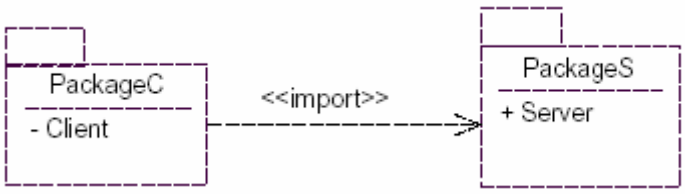


Figura 24 - Dipendenza fra packages e import

La relazione di *importa* è un particolare tipo di relazione di dipendenza fra packages (relazione di dipendenza con stereotipo <<import>>), che fornisce all'interno del package dipendente visibilità su tutti gli elementi pubblici presenti nel package importato.

Il vantaggio di questa seconda soluzione risulta evidente per sistemi composti da un numero molto elevato di elementi: infatti invece di descrivere le relazioni di dipendenza all'interno del sistema in

termini di dipendenza fra singoli elementi esse possono essere descritte in termini di dipendenza fra gruppi elementi. Inoltre fornendo un meccanismo per schermare la visibilità di un elemento all'esterno di un gruppo si rende più difficoltoso instaurare relazioni di dipendenza non necessarie. Infine la dipendenza di tipo importa consente di specificare una visibilità in un solo verso, rendendo più difficoltoso l'instaurarsi di relazioni di dipendenza ciclica fra elementi del modello del sistema.

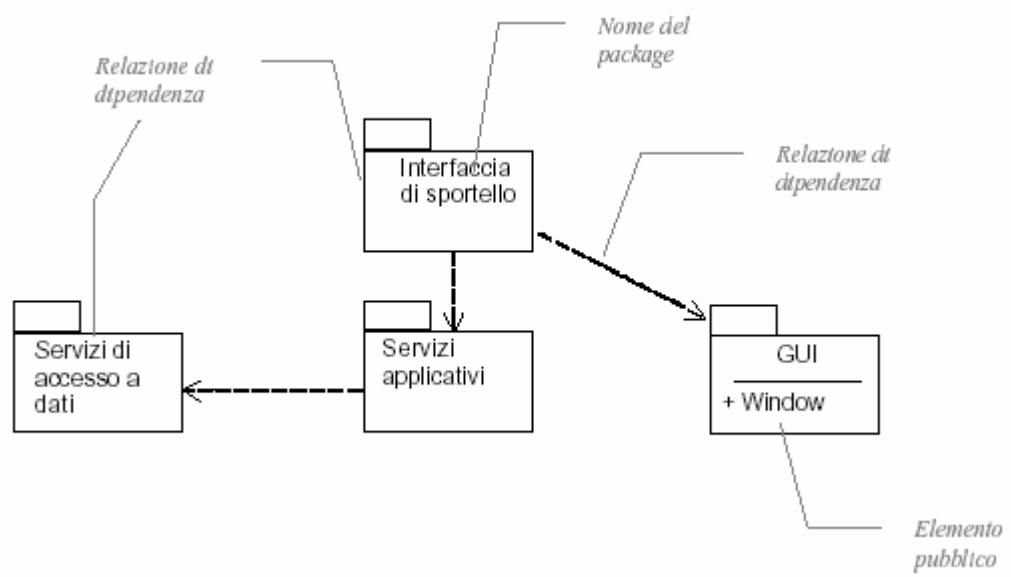


Figura 25 - Package diagram

Il costrutto di package è fondamentale per la gestione di grandi progetti: quando un class diagram che rappresenta l'intero progetto non è più gestibile e maneggevole da utilizzare è richiesto l'utilizzo di un package diagram.

I packages sono importanti anche per organizzare le attività di test, offrendo anche in questo un livello di granularità più appropriato rispetto alle singole classi.

3.3.3. Diagrammi di oggetti

I **diagrammi degli oggetti** modellano una vista statica del sistema, fornendo un'istantanea del sistema in un particolare momento della sua vita descrivendo insiemi di oggetti, le loro relazioni e il loro stato.

Tale diagramma può essere usato in due modi: validare il progetto degli aspetti strutturali del sistema fornendo un esempio di quelli che saranno gli oggetti del sistema, le loro relazioni e il loro stato. Dal punto di vista di un utente può essere più semplice in alcuni casi ragionare su istanze di oggetti del sistema che sulle classi cui tali istanze appartengono.

Inoltre la possibilità di osservare lo stato di una porzione del sistema e la configurazione che tale porzione assume in un determinato momento del suo ciclo di vita è molto importante per analizzare il funzionamento del sistema stesso. Per mezzo di tale diagramma è possibile verificare in un dato momento dell'esecuzione del sistema quali oggetti sono presenti, quale stato tali oggetti assumono e quali sono le relazioni fra di loro.

Un diagramma degli oggetti conterrà: oggetti e collegamenti fra di essi.

Un oggetto è rappresentato da un rettangolo contenente il nome dell'oggetto, il nome della classe cui esso appartiene eventuali valori degli attributi.

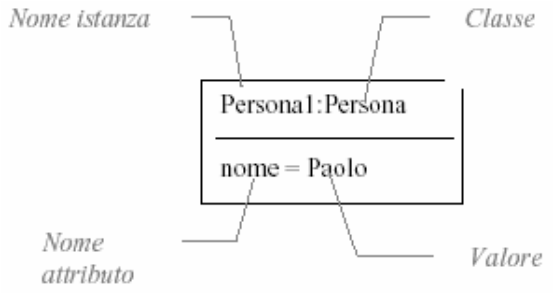


Figura 26 - Rappresentazione di un oggetto

Il diagramma degli oggetti mostrato in figura rappresenta il check-up effettuato dal paziente Rossi in data 1/1/99, insieme agli esami che esso prevede e ai risultati di tali esami.

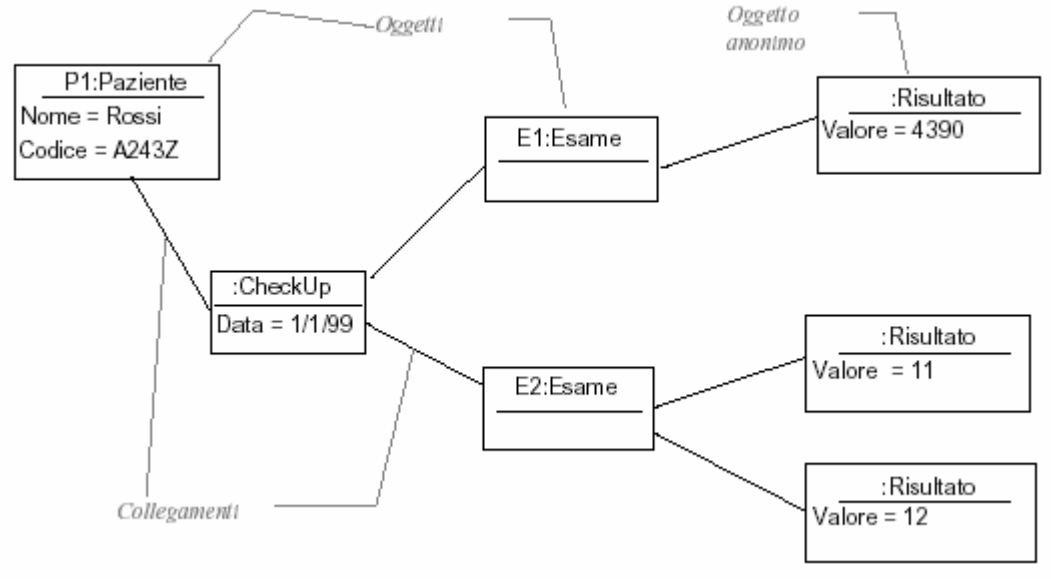


Figura 27 - Diagramma deegli oggetti

3.4. Modellare il comportamento

UML mette a disposizione i diagrammi use case, i diagrammi di interazione, i diagrammi di stato e di diagrammi di attività per la descrizione del comportamento del sistema.

Un diagramma use case consente di specificare il comportamento del sistema evidenziando le relazioni fra gli attori (un particolare tipo di classe) e gli use case (un insieme di azioni con cui il sistema svolge un compito).

I diagrammi di interazione, che si differenziano ulteriormente in sequence e collaboration diagram, descrivono un insieme di oggetti, le relazioni fra tali oggetti, i messaggi che essi si scambiano nello svolgere un determinato compito. I sequenze diagram evidenziano maggiormente la sequenza temporale dello scambio di messaggi fra gli oggetti, mentre i collaboration diagram pongono l'accento sulla struttura risultante dalle relazioni fra gli oggetti stessi.

I diagrammi di stato descrivono una macchina a stati, le transizione da uno stato al successivo, gli eventi che le scatenano e le azioni in corrispondenza di ciascuna transizione. Sono particolarmente adatti a modellare il comportamento di un singolo oggetto e il modo in cui esso reagisce alle sollecitazioni esterne.

I diagrammi di attività sono un tipo di diagramma di stato che mostra il flusso di controllo da un'attività all'altra all'interno del sistema. Essi sono adatti a modellare il funzionamento del sistema, il flusso di controllo fra gli oggetti e l'esecuzione concorrente di attività.

3.4.1. Use cases e diagrammi use case

Uno **use case** descrive l'interazione fra un **attore** e il sistema da sviluppare. Lo use case specifica il comportamento del sistema, elencando le azioni da compiere per fornire ad un determinato utente un risultato atteso.

Un modo per arrivare alla specifica degli use cases è di intervistare gli utenti per chiedere loro quali funzionalità dovranno essere supportate dal sistema, chiedendogli di specificare in dettaglio crescente in che modo tali funzionalità si decompongono in sottofunzioni più semplici. Uno use case può essere descritto a diversi livelli di dettaglio specificando così: gli obiettivi che un utente intende raggiungere nell'utilizzo del sistema oppure il modo in cui l'utente interagisce con il sistema stesso.

La parola **attore** indica il ruolo che un utente, un essere umano o un sistema diverso da quello che si intende realizzare, gioca nel corso dell'interazione con il sistema stesso.

Use Case: *VERSAMENTO DI UN ASSEGNO SU DI UN CONTO CORRENTE BANCARIO*

Attori: Possessore dell'assegno, Presentatore, Cassiere

Descrizione: Un cliente, o presentatore, arriva allo sportello bancario e consegna al Cassiere un assegno compilato e firmato dal Possessore dell'assegno. Il Cassiere prende l'assegno, verifica che il conto corrente cui fa riferimento l'assegno abbia una disponibilità sufficiente e registra l'operazione.

Lo use case precedente è in una versione semplificata e descrive le azioni che diversi attori compiono allo scopo di versare un assegno su di un conto corrente bancario.

UML definisce un tipo di diagramma per rappresentare e visualizzare gli use case denominato **use case diagram**.

Il diagramma che segue descrive il precedente use case.

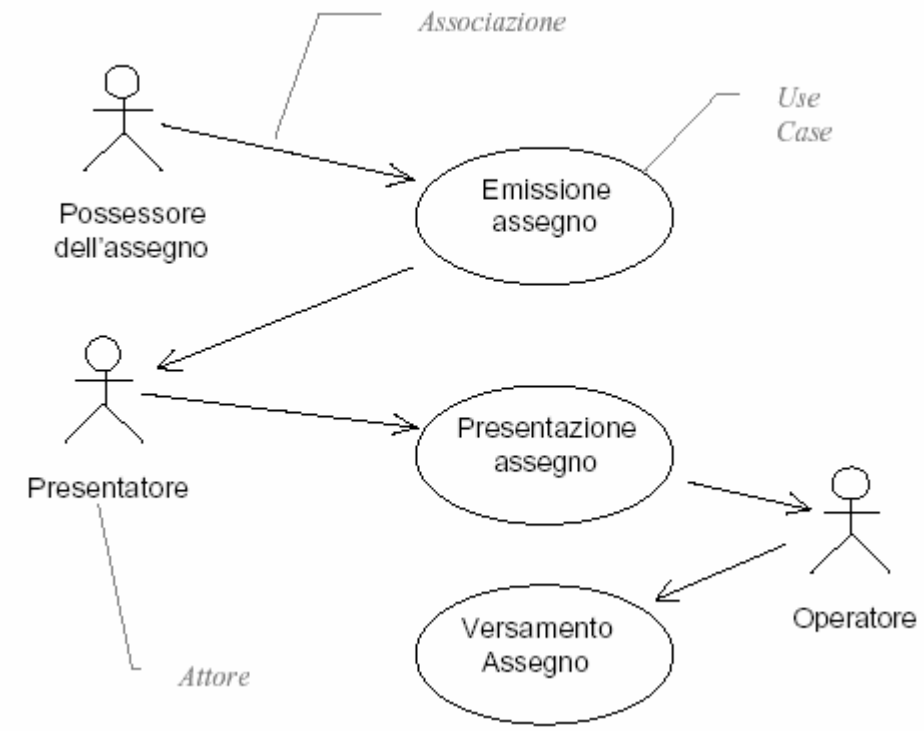


Figura 28 - Diagramma use case del versamento di un assegno

Uno use case può avere diverse varianti; esso può infatti descrivere funzionalità:
- che rappresentano un'**estensione** di quelle già descritte in un altro use case;
- che **includono** quelle descritte in un altro use case;
- che rappresentano una **specializzazione** di quelle già descritte in un altro use case.

Un'estensione dello use case precedente *VERSAMENTO DI UN ASSEGNO SU DI UN CONTO CORRENTE BANCARIO* può essere quella di versamento di un assegno con priorità dell'accredito.

Use Case: *VERSAMENTO DI UN ASSEGNO CON PRIORITÀ DI ACCREDITO*

Attori: Possessore dell'assegno, Presentatore, Cassiere

Descrizione: Un cliente, o presentatore, arriva allo sportello bancario e consegna al Cassiere un assegno compilato e firmato dal Possessore dell'assegno. Il Cassiere prende l'assegno verifica che il conto corrente cui fa riferimento l'assegno abbia una disponibilità sufficiente e autorizza il versamento con priorità di accredito, fornendo al sistema i dati per il proprio riconoscimento.

Il seguente use case diagram descrive la relazione di estensione fra i due use cases.

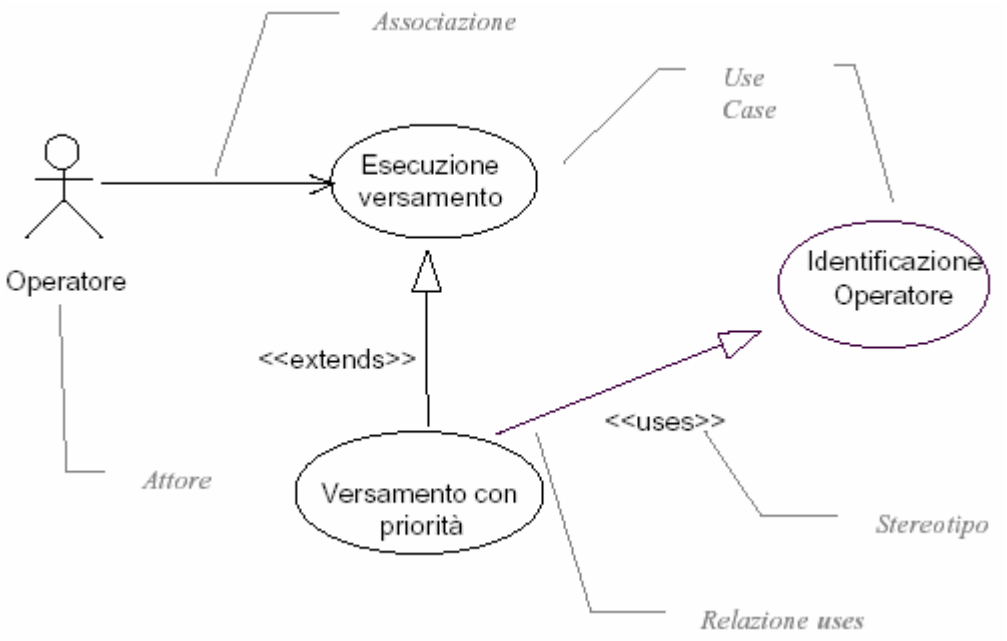


Figura 29 - Use case diagram di dettaglio del versamento con sconfino

Nell'esempio precedente lo use case VERSAMENTO CON PRIORITÀ svolge la stessa funzione dello use case ESECUZIONE VERSAMENTO, aggiungendo a questo la predisposizione per un accredito con una determinata priorità. Naturalmente questa è solo una possibile estensione dello use case di base; altre potrebbero riguardare il caso in cui il conto di provenienza dell'assegno non abbia fondi sufficienti o il caso in cui l'assegno non sia compilato correttamente. Più in generale la relazione **extends** permette di catturare tutte quelle relazioni fra due use case in cui uno dei due rappresenta una situazione di eccezione rispetto all'altro o mostra un comportamento più complesso.

Lo use case VERSAMENTO CON PRIORITÀ d'altra parte include al suo interno lo use case di IDENTIFICAZIONE OPERATORE. Tutte le volte in cui uno use case descrive un insieme di attività che sono un sottoinsieme di quelle descritte in altri use case è possibile utilizzare la relazione **uses**.

Gli use cases sono utili in diverse fasi di un progetto: nella fase di raccolta dei requisiti, in quella di pianificazione e per il controllo del progetto stesso. La maggior parte degli use cases sarà generata nel corso della fase di elaborazione.

3.4.2. Diagrammi di interazione

I diagrammi di interazione descrivono il comportamento di più oggetti per la realizzazione di una specifica attività; essi permettono di catturare il comportamento dinamico del sistema nel suo insieme. Un tipico utilizzo di un diagramma di interazione è quello di descrivere il flusso di controllo fra gli oggetti per la realizzazione di una funzione del sistema descritta in uno use case.

La specifica dei diagrammi di interazione comporta l'assegnazione di responsabilità agli oggetti: il fatto stesso che un oggetto sia in grado di rispondere a un messaggio inviato da un altro oggetto implica una decisione progettuale di assegnazione delle responsabilità.

UML definisce due diversi formalismi per la descrizione delle interazioni fra gli oggetti del sistema: i **collaboration diagram** e i **sequence diagram**.

Un **collaboration diagram** rappresenta l'interazione fra oggetti del sistema ponendo una maggiore enfasi sull'organizzazione degli oggetti che scambiano i messaggi.

Un **sequence diagram** pone l'accento sulla sequenza temporale dei messaggi inviati.

3.4.2.1. Sequence diagram

I **sequence diagram** sono caratterizzati dalla rappresentazione di una linea temporale sulla quale sono disposti gli oggetti che interagiscono. A ciascun oggetto è associata una linea tratteggiata verticale, o **lifeline**; su di una lifeline possono essere posizionate delle barre di **attivazione** utilizzate come punto di partenza e di arrivo dei messaggi da e verso l'oggetto. Ogni messaggio viene rappresentato da una freccia che viaggia dalla barra di attivazione di un oggetto a quella di un altro oggetto. Per ogni messaggio sono indicati il nome ed eventualmente gli argomenti del messaggio.

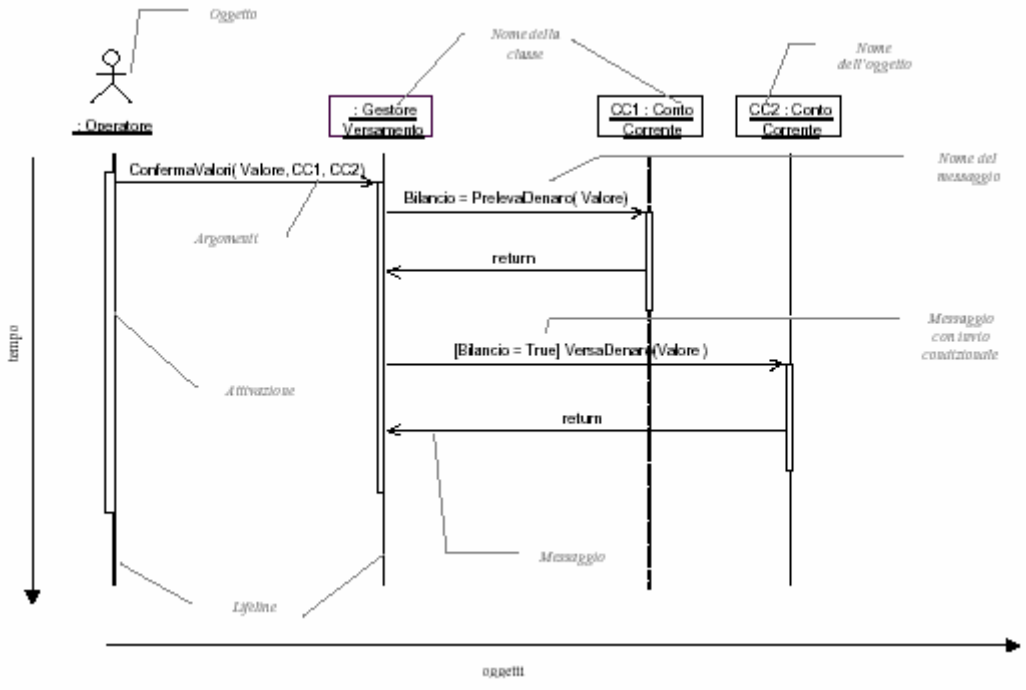


*Figura 30 - **Sequence diagram** per il versamento di un assegno su conto corrente*

Nell'esempio un operatore può fornire ad un oggetto i valori necessari alla gestione di un versamento su conto corrente. Tale oggetto verifica la disponibilità dell'ammontare sul conto di provenienza dell'assegno e in caso il bilancio sia positivo effettua il versamento sul conto corrente di accredito.

3.4.2.2. Collaboration diagram

Un **collaboration diagram** evidenzia le relazioni fra gli oggetti: ciascun oggetto è rappresentato tramite un'icona e collegato ad altri oggetti tramite **links**. Un messaggio è rappresentato tramite una freccia affiancata ad un link. Se due oggetti collegati si scambiano più messaggi, tali messaggi vengono visualizzati vicino al medesimo link, specificando per ciascun messaggio un numero di ordine.

La medesima interazione fra gli oggetti presentata precedentemente può essere illustrata per mezzo di un **sequence diagram**.

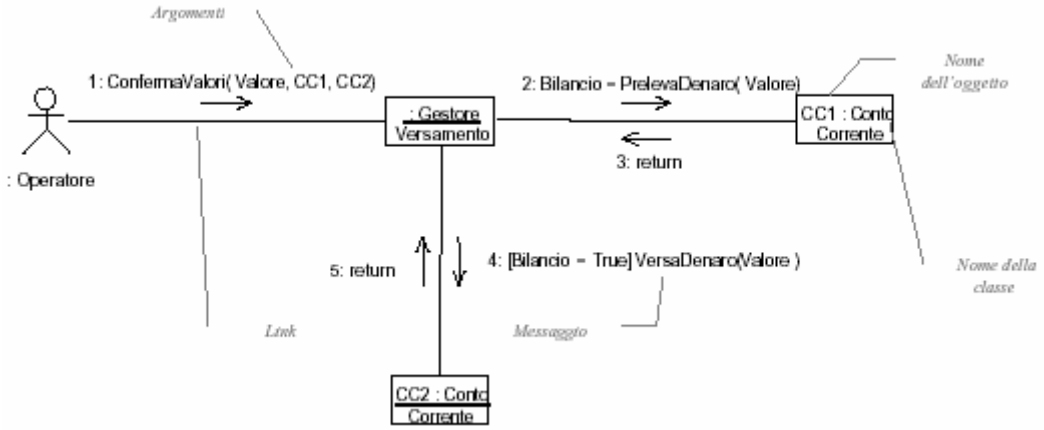


*Figura 31 - **Collaboration diagram** per il versamento di un assegno su conto corrente*

Si noti come la sequenza dei messaggi sia stabilita per mezzo dei numeri d'ordine associati ad ogni messaggio.

I diagrammi di interazione sono utili nel modellare il comportamento di più oggetti nello svolgimento di una determinata attività e il modo in cui più oggetti collaborano. Se è necessario descrivere il comportamento di un singolo oggetto nel corso di diverse attività è più appropriato invece l'utilizzo di uno state diagram, mentre per controllare il comportamento di gruppi di oggetti nel corso di attività differenti o specificare l'esecuzione parallela di attività è più appropriato l'uso di un activity diagram.

3.4.3. Diagrammi di attività

Gli **activity diagrams** sono in grado di modellare gli aspetti dinamici di un sistema. In particolare essi sono adatti a visualizzare il flusso di più oggetti attraverso diversi stati.

Mentre i diagrammi di interazione si concentrano sul modo in cui il flusso di controllo si sposta fra diversi oggetti, un **activity diagram** è in grado di rappresentare il flusso di controllo attraverso diverse attività.

Un primo esempio di **activity diagram** è quello che descrive il cambio di un assegno in contanti, nel caso in cui sia necessaria per la transazione un'autorizzazione da parte del direttore della banca.

Nel diagramma presentato l'attività iniziale consiste nell'apertura del conto corrente su cui

effettuare l'accredito; a questo punto più attività possono essere eseguite in parallelo, ovvero senza necessità di specificare un ordine logico. La suddivisione del flusso di controllo viene indicata per mezzo di una barra di **fork**. A questo punto un primo flusso di attività prevede l'inserimento dei dati dell'assegno da versare e la verifica della disponibilità sul conto corrente dal quale proviene l'assegno; il secondo flusso di attività verifica invece la necessità di un autorizzazione per il versamento di assegni fuori piazza e ottiene eventualmente tale autorizzazione.

La barra di **sincronizzazione** consente quindi di unificare nuovamente il controllo dei diversi flussi di attività e passa all'esecuzione dell'operazione, producendo una registrazione del versamento effettuato.

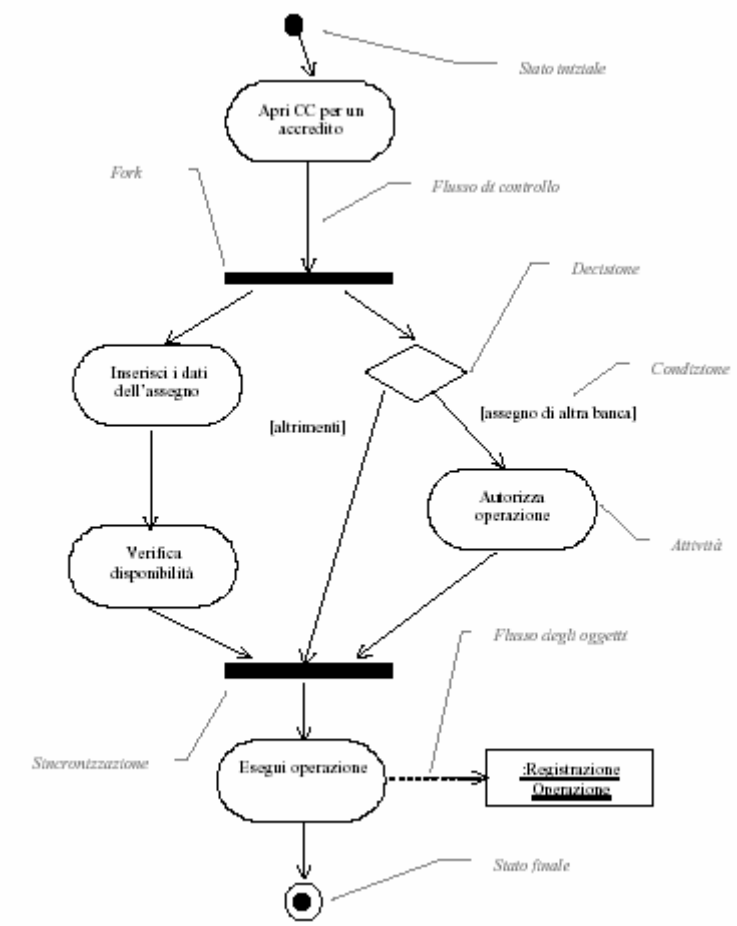


*Figura 32 - **Activity diagram** per la descrizione delle attività collegate al versamento di un assegno*

Questo tipo di diagramma può esser utilizzato per analizzare in dettaglio le attività previste all'interno di uno **use case**, evidenziando i vincoli di sequenzialità fra diverse attività o invece la possibilità di eseguire alcune attività in parallelo. Inoltre un **activity diagram** può essere utilizzato per mostrare il comportamento di organizzazioni in cui il sistema software si trova ad interagire con processi molto complessi.

3.4.4. State diagram

Uno **state diagram** descrive il comportamento di un particolare oggetto indicando gli stati in cui esso può trovarsi e in che modo possono avvenire i passaggi fra uno stato e l'altro: tale diagramma è quindi in grado di mostrare il ciclo di vita di un oggetto.
Gli elementi principali di uno **state diagram** sono gli stati che l'oggetto può attraversare, le

transizioni fra uno stato e il successivo, gli eventi che attivano le transizioni e le azioni compiute nel corso di una transizione. Le transizioni fra uno stato e l'altro possono essere condizionali ad indicare che oltre allo scatenarsi di un evento il cambiamento di stato richiede il verificarsi di una precisa condizione. Inoltre per ciascuno stato possono essere indicate delle azioni da compiere in condizioni specifiche (ingresso, uscita e permanenza, ad esempio). Infine più stati possono essere raggruppati in superstati, allo scopo di indicare in modo semplice un comportamento comune dell'oggetto in stati differenti, ad esempio transizioni che si svolgono in modo identico da o verso un gruppo di stati.

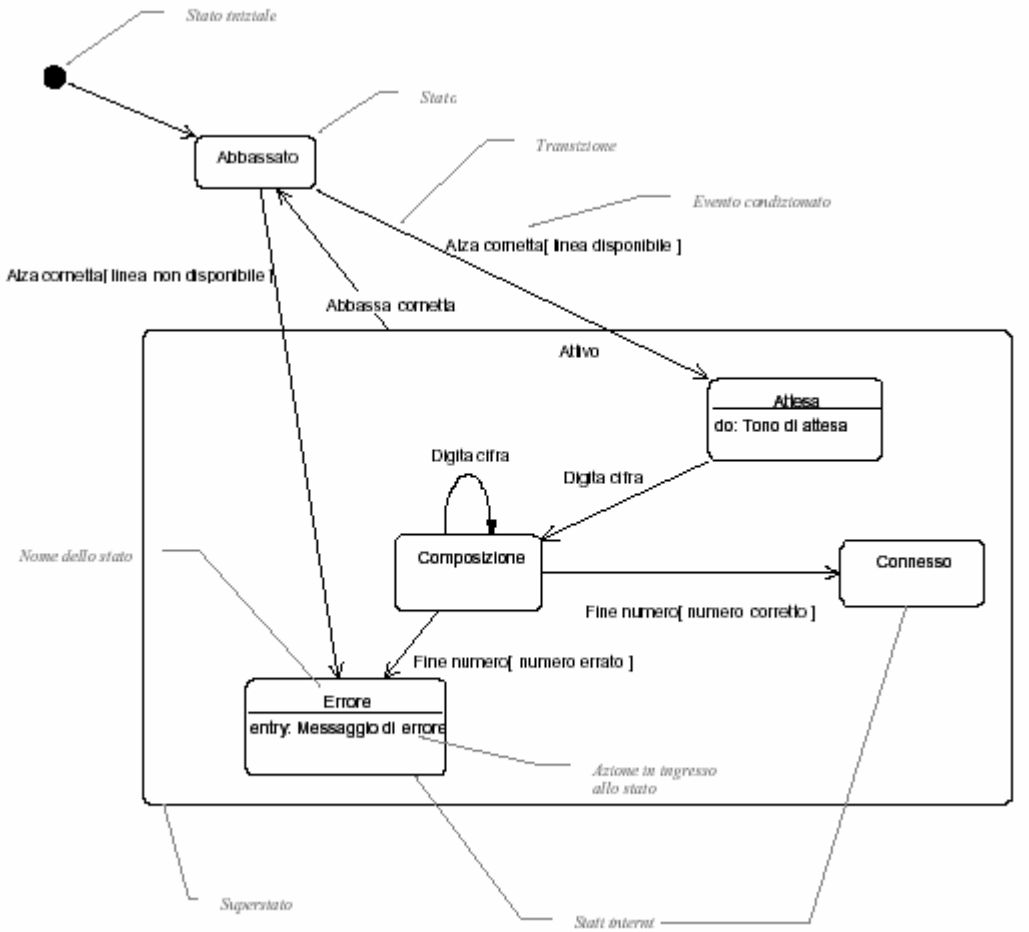


*Figura 33 - **State diagram** per la descrizione di una comunicazione telefonica*

Nel diagramma precedente dallo stato iniziale (abbassato) alzando la cornetta si può ottenere la linea (notificata da un tono di attesa continuo) oppure se la linea non è disponibile ottenere un messaggio di errore. Ottenuta la linea si possono comporre le cifre del numero desiderato; terminata la composizione se il numero è corretto si ottiene la connessione, altrimenti si ottiene un messaggio di errore. Poiché gli stati di attesa, composizione, errore e connessione sono stati specificati come interni al superstato attivo, è possibile indicare un'unica transizione da questo stato allo stato abbassato quando l'operatore abbassa la cornetta.

3.5. Modellare l'architettura

Il linguaggio UML permette di rappresentare l'architettura di un sistema per mezzo dei diagrammi di componenti e dei diagrammi di dispiegamento.

Un diagramma dei componenti mostra l'organizzazione e le dipendenze in un insieme di componenti del sistema, e rappresenta una vista sull'implementazione del sistema stesso.

Il diagramma di dispiegamento mostra la configurazione dei nodi che supportano l'esecuzione del sistema e l'allocazione dei componenti su tali nodi.

3.5.1. Diagramma dei componenti

Un componente è un oggetto fisico che costituisce il sistema e implementa una o più interfacce. Un componente può avere un nome ed è rappresentato da un rettangolo con due bande laterali.

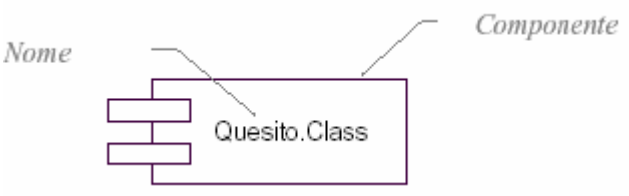


Figura 34 – Componente

Un componente è simile ad una classe dal punto di vista del linguaggio di modellazione: entrambi realizzano interfacce, possono avere relazioni, possono essere istanziati. Essi si collocano però ad un diverso livello di astrazione:

- le classi rappresentano un'astrazione logica;
- i componenti invece sono oggetti fisici che esistono nel sistema reale e sono allocate sui nodi che compongono la piattaforma di esecuzione del sistema.

L'implementazione di un sistema può essere quindi decomposta specificando delle interfacce e realizzando componenti:

- che implementano tali interfacce;
- che accedono ai servizi utilizzando tali interfacce.

In questo modo è possibile strutturare un sistema i cui componenti siano facilmente modificabili e sostituibili.

UML definisce diversi tipi di componenti:

- componenti che fanno parte del sistema in esecuzione, come eseguibili, librerie dinamiche;
- componenti che risultano dal processo di sviluppo, come file sorgenti;
- componenti istanziati nel corso dell'esecuzione del sistema.

Un diagramma composto dal primo tipo di componenti permette di specificare le dipendenze fra gli oggetti in esecuzione su di un sistema.

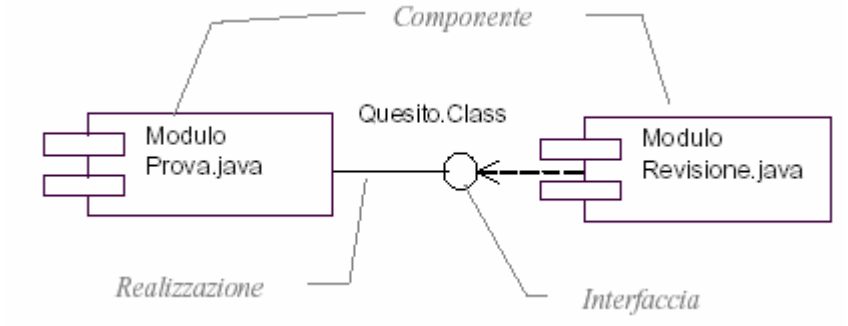


Figura 35 - Dipendenza in un diagramma dei component

Un diagramma composto da file sorgenti permette di specificare, ad esempio, le dipendenze di compilazione fra essi.

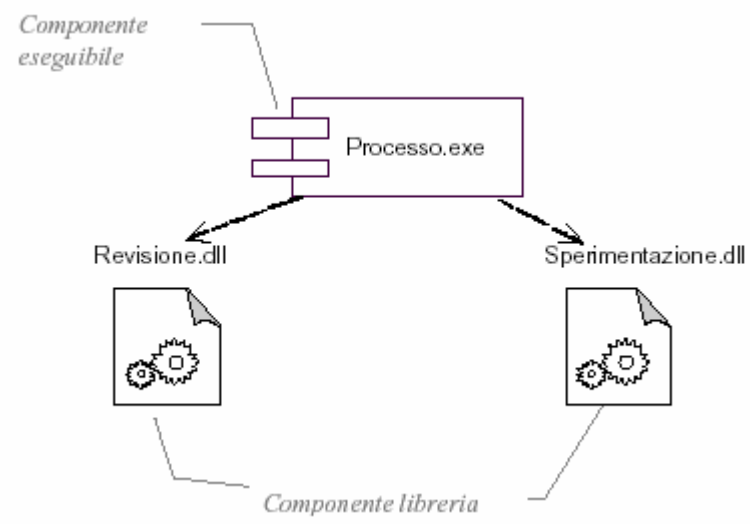


Figura 36 - Diagramma dei componenti in esecuzione su di un sistema

Si noti che i componenti possono essere organizzati in packages.

3.5.2. Diagramma di dispiegamento

Elemento fondamentale di un diagramma di dispiegamento è il nodo. Un nodo è un elemento fisico del sistema che esiste quando il sistema è in esecuzione e rappresenta una risorsa che può avere o meno capacità di calcolo.

Un nodo rappresenta la risorsa fisica sulla quale sono allocati i componenti al momento dell'esecuzione del sistema.

Un nodo è rappresentato graficamente come un cubo con un nome.

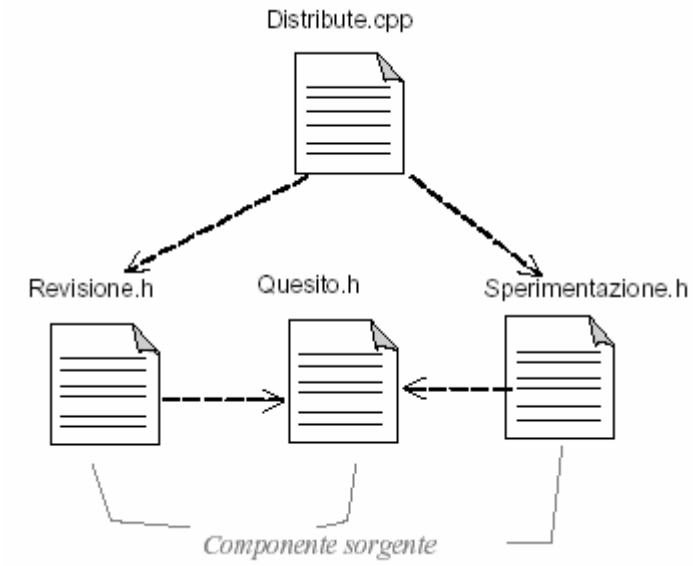


Figura 37 – Dipendenze di compilazione fra sorgenti

Un nodo è la risorsa fisica sulla quale i componenti sono allocati. Se un nodo ha capacità di calcolo esso potrà ospitare componenti che sono è un oggetti eseguibili. In questo caso esiste una relazione di dipendenza fra nodi e componenti.

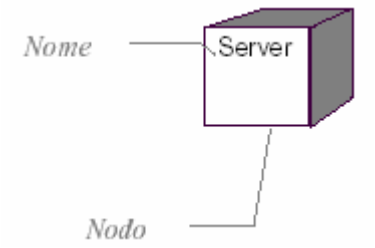


Figura 38 – Nodo

Un nodo è la risorsa fisica sulla quale i componenti sono allocati. Se un nodo ha capacità di calcolo esso potrà ospitare componenti che sono è un oggetti eseguibili. In questo caso esiste una relazione di dipendenza fra nodi e componenti.

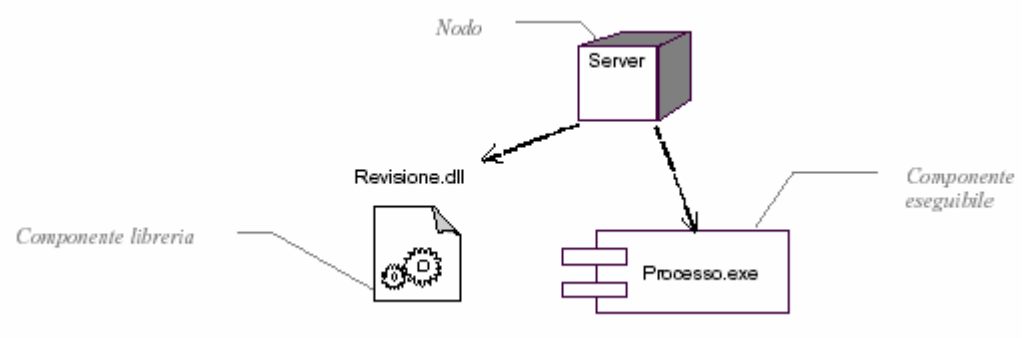
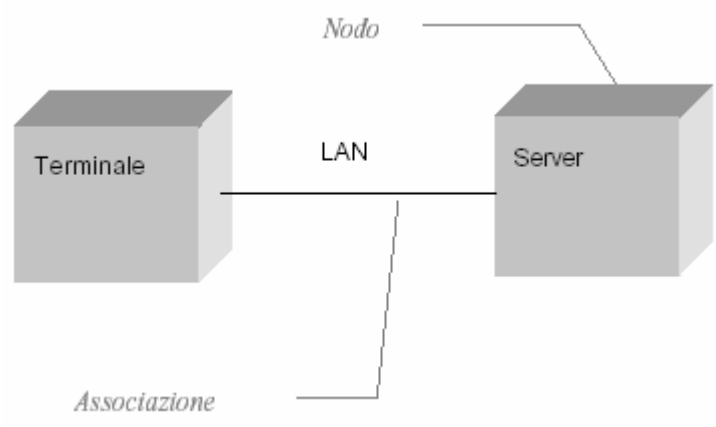


Figura 39 - Dipendenza di un nodo dai componenti

Un nodo può essere associato ad altri nodi per indicare una connessione fisica, oltre che altri tipi di relazione, come aggregazione, dipendenza e generalizzazione.

Il diagramma seguente mostra un terminale collegato via rete locale ad un server di calcolo.



4. Bibliografia

Larman97 Larman, C., 1997. *Applying UML andPatterns*. Prentice Hall

Booch94 Booch, G. 1994. *Object Oriented Analisys and Design with Applications*. Benjamin/Cummings

Booch98 Booch, G., 1998. *The Unified Modeling Language User Guide*. Addison-Wesley

Fowler97B Fowler M., 1997. *Analisys Patterns*. Addison-Wesley

Standish95 Standish Group, 1995. *Charting the Seas o Information Technology: Chaos*. Rapporto Standish Group

Jacobson94 Jacobson, I. et al., 1994. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley

Fowler97 Fowler, M. et al., 1997. *UML Distilled*. Addison-Wesley

Gamma94 Gamma, E. et al., 1994. Design Patterns: Elements of Reusable object- Oriented Software. Addison-Wesleys