Capitolo 6

UML – Unified Modeling Language

Introduzione

In un mondo costantemente in fermento ed in evoluzione come quello dello sviluppo di applicazioni Object Oriented, diviene sempre più difficile costruire e gestire applicazioni di alta qualità in tempi relativamente brevi. Tutto ciò, unito all' esigenza di avere un linguaggio universale per modellare gli oggetti utilizzabile da ogni industria produttrice di software, ha dato origine al linguaggio UML, la cui sigla significa Unified Modeling Language.

La forza dell'Unified Modeling Language è nel fatto che il processo di disegno del sistema può essere effettuato in modo tale che i clienti, gli analisti, i programmatori e chiunque altro coinvolto nel sistema di sviluppo, possano comprendere ed esaminare in modo efficiente il sistema e prendere parte alla sua costruzione in modo attivo.

UML è una notazione object-oriented utilizzabile durante le fasi di analisi e design di un sistema.

Il vantaggio di osservare il programma sotto due distinte "viste" del sistema, la vista statica, che rappresenta informazioni che non evolvono nel tempo, tramite class diagram e object diagram e la vista dinamica, in cui si mostra l'evoluzione delle parti del sistema nel tempo, mediante use case diagram, sequence diagram, collaboration diagram, statechart diagram, activity diagram e implementation diagram.

In particolare, si andavano sviluppando tre di queste metodologie:

1. OMT (Rumbaugh)
2. Booch 1991

3. OOSE (Jacobson)

Ognuno di questi metodi aveva, naturalmente, punti di forza e punti deboli: ad esempio, l'OMT si rivelava ottimo in analisi e debole nel disegno. Booch 1991, al contrario, eccelleva nel disegno e peccava in analisi, mentre Jacobson aveva il suo punto di forza nell'analisi dei requisiti e del comportamento di un sistema ma si rivelava debole in altre aree. Successivamente, Booch scrisse il suo secondo libro che adottava i principi di analisi utilizzati da Rumbaugh e Jacobson nelle loro rispettive metodologie. A sua volta, Rumbaugh pubblicò una serie di articoli (conosciuti come OMT-2) che riprendevano parecchie delle tecnologie di disegno di Booch. In sostanza, i tre metodi stavano convergendo verso un'unica visione che incorporasse le qualità migliori che ognuno di essi aveva mostrato, con il solo limite che ogni metodo portava ancora con sé la propria notazione di origine. Tale problema non era da sottovalutare, in quanto l'uso di differente simbologia, tendeva facilmente a confondere sul mercato analisti e disegnatori, a causa della diversa interpretazione che essi potevano dare ai vari simboli.

La prima versione ufficiale, prodotta dall'OMG (Object Management Group) fu rilasciata nel Luglio del 1997 e nel Novembre dello stesso anno l'UML venne adottato come standard.

Elenchiamo, qui di seguito, alcuni dei benefici derivanti dall'utilizzo del linguaggio UML (www.programmazione.html.it/html):

1. Disegno professionale e documentato della soluzione informatica prima di scrivere il codice: si sarà così in grado di conoscere in anticipo il risultato finale del progetto su cui si sta lavorando;
2. Scrittura del codice stessa è resa più agevole, efficiente e riutilizzabile in futuro: i costi di sviluppo, dunque, si abbassano notevolmente con l'utilizzo del linguaggio UML;
3. Migliore previsione di eventuali bugs nel sistema: il software che si scrive, si comporterà esattamente come ci si aspetta senza spiacevoli sorprese finali;
4. Definizione chiara dell'idea, a tutti coloro che fanno parte dello sviluppo, del progetto nel suo insieme: in questo modo, si potranno sfruttare al meglio anche le risorse hardware in termini di memoria

ed efficienza, senza sprechi inutili o, al contrario, rischi di sottostima dei requisiti di sistema;

5. Semplificazione dello sviluppo ulteriore del codice: comporta ovviamente un beneficio in termini di mantenimento del sistema.

La comunicazione e l'interazione tra tutte le risorse umane che prendono parte allo sviluppo del sistema è quindi molto più efficiente e diretta. Avere la possibilità di trasmettere le idee ed i lavori svolti in maniera più semplice da comprendere, facilita il lavoro di definizione, sviluppo, ma anche la commercializzazione dei prodotti.

Il Visual Modeling prevede una continua interazione tra i vari membri del team di lavoro. A loro volta, i programmatori devono interagire con gli analisti ed i disegnatori per condividere le loro intuizioni, modificare i disegni e consolidare il codice. Il vantaggio di questa metodologia consiste nello sviluppo della conoscenza generale del team: ognuno è in grado di capire a fondo il sistema e il risultato finale non può che essere un sistema più solido. Il RAD Rapid Application Development, lo sviluppo di applicazioni in tempi brevi , consiste di cinque sezioni:

1. Raccolta delle informazioni
2. Analisi
3. Disegno
4. Sviluppo
5. Deployment

Come ogni linguaggio che si rispetti, anche l'UML necessita dell'utilizzo di programmi applicativi appropriati che ne agevolino l'utilizzo. Ci sono sul mercato svariate opportunità che trattano l'UML, il più diffuso è lo strumento della Rational, Il Rose (www.rational.com), disegnato per fornire ai team di sviluppo tutti gli strumenti necessari per il modellamento di soluzioni robuste ed efficienti.

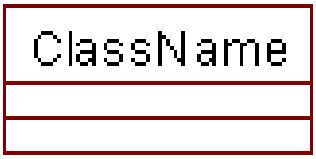
Elementi di una rappresentazione *UML*

Il linguaggio UML contiene svariati elementi grafici che vengono combinati tra loro durante la creazione dei diagrammi, rispettando delle regole precise e standardizzate quanto più possibile. Di seguito è illustrata una panoramica delle tipologie di diagrammi possibili nella creazione di programmi, raggruppandoli per diagrammi statici, i primi due, e dinamici, i successivi.

Un class diagram è un'evoluzione dei diagrammi entità-relazioni (E/R). In un class diagram si rappresentano le entità significative del sistema e le relazioni che esistono fra le entità.

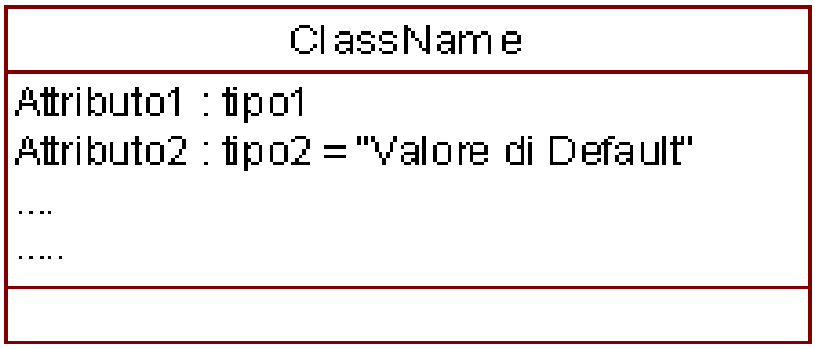
In un object diagram si rappresenta una particolare "configurazione" del sistema, ovvero un particolare insieme di oggetti (istanze di classi) e le rispettive relazioni.

Prima di tutto, però, è importante definire graficamente una classe in UML:

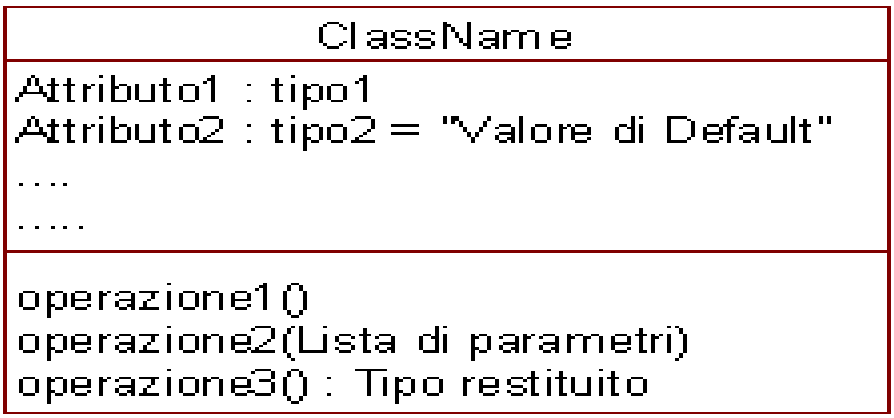


Una classe viene rappresentata da un rettangolo. Il nome della classe, per convenzione, è una parola con l'iniziale maiuscola ed appare vicino alla sommità del rettangolo. Se il nome della classe definita consiste di una parola composta a sua volta da più parole allora viene utilizzata la notazione ungherese, in cui tutte le iniziali di ogni parola sono scritte in maiuscolo.

Un Attributo il cui nome è costituito da una sola parola viene scritto sempre in caratteri minuscoli



La lista delle operazioni, ossia i metodi, viene rappresentata graficamente sotto la lista degli attributi e separata da questa tramite una linea orizzontale.

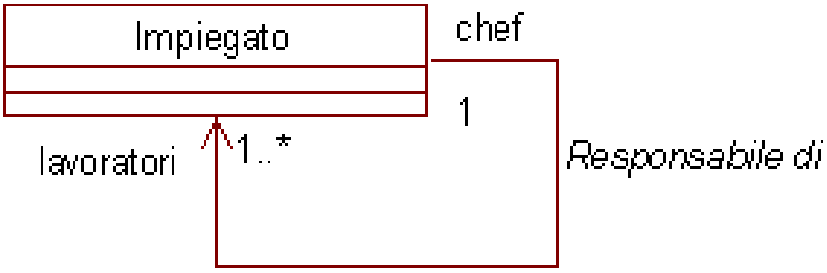


Quando più classi sono connesse l'una con l'altra da un punto di vista concettuale, tale connessione viene denominata associazione.

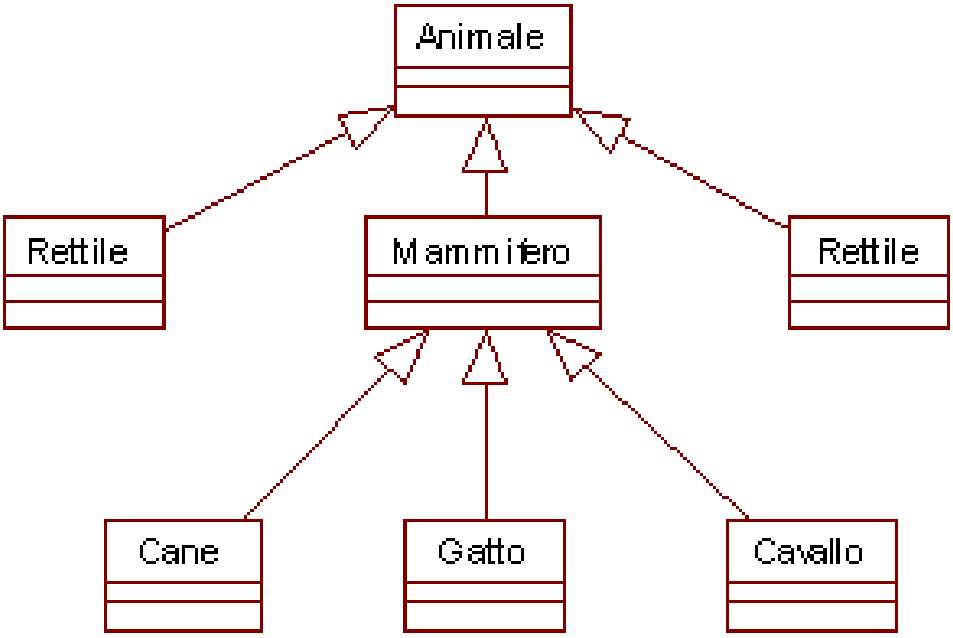
La molteplicità è un tipo di associazione in cui si mostra il numero di oggetti appartenenti ad una classe che interagisce con il numero di oggetti della classe associata. Una classe, in generale, può essere correlata ad una altra nei seguenti modi:

1. Uno ad uno
2. Uno a molti
3. Uno ad uno o più
4. Uno a zero o uno
5. Uno ad un intervallo limitato (es.: 1 a 2 - 20)
6. Uno ad un numero esatto n
7. Uno ad un insieme di scelte (es.: 1 a 5 o 8)

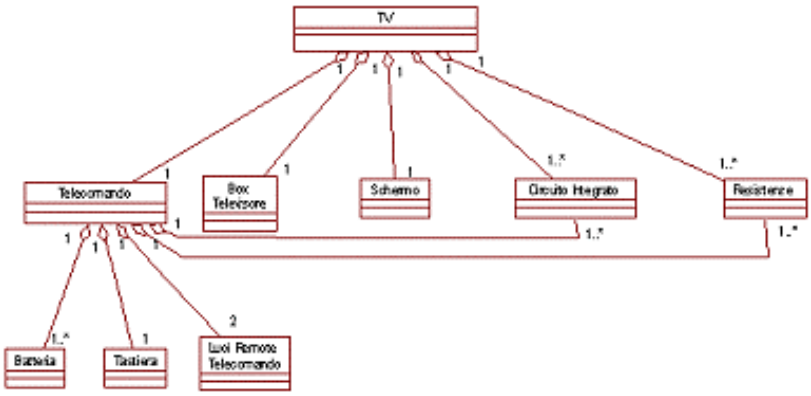
Il linguaggio UML utilizza un asterisco (*) per rappresentare le opzioni molti e più.



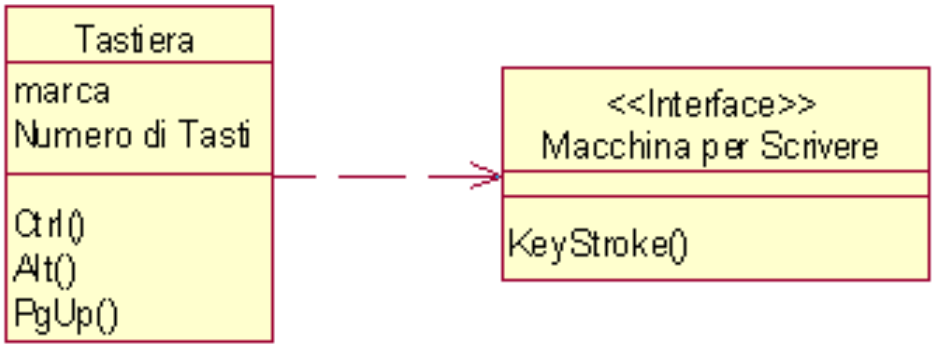
La rappresentazione dell'ereditarietà è espressa facilmente nella figura x .



Qualche volta una classe può rappresentare il risultato di un insieme di altre classi che la compongono. Questo è un tipo speciale di relazione denominata aggregazione. Le classi che costituiscono i componenti e la classe finale sono in una relazione particolare del tipo: parte – intero.



Una interfaccia è una collezione di operazioni che una classe esegue, per distinguere le interfacce dalle classi, dal punto di vista UML si utilizza la scrittura <<Interface>> all'inizio del nome di ogni interfaccia, come mostrato nella figura precedente.



La Visibilità si applica ad attributi o operazioni e specifica la possibilità che hanno le classi di usare gli attributi e le operazioni di un'altra classe. Sono consentiti tre livelli di visibilità:

1. Livello pubblico: L'utilizzo viene esteso a tutte le classi
2. Livello protetto: L'utilizzo è consentito soltanto alle classi che derivano dalla classe originale
3. Livello privato: Soltanto la classe originale può utilizzare gli attributi e le operazioni definite come tali.

A livello grafico vengono utilizzati, generalmente, i seguenti simboli per distinguere i tre livelli:

1. Livello pubblico: +
2. Livello protetto: #
3. Livello privato: -

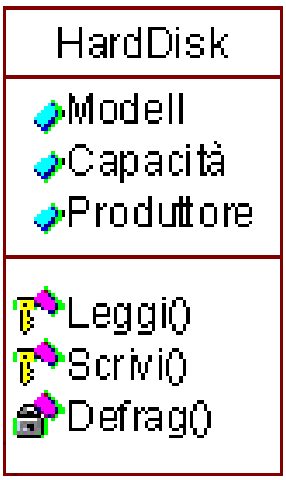
Il Rational Rose, invece, utilizza i seguenti simboli, per le operazioni:

1. Livello pubblico:
2. Livello protetto:
3. Livello privato:

E gli stessi, con il colore differente, per gli attributi:

1. Livello pubblico:

2. Livello protetto:
3. Livello privato:



In un sequence diagram si rappresenta la sequenza temporale delle interazioni che avvengono fra oggetti del sistema. E' un diagramma bidimensionale in cui su un asse si dispongono gli oggetti che interagiscono, ed il secondo asse rappresenta il tempo. Le interazioni fra oggetti sono costituite dallo scambio di messaggi che avviene fra di essi.

Un messaggio che viaggia da un oggetto ad un altro, viene disegnato a partire dalla lifeline dell'oggetto da cui parte il messaggio e arriva sulla lifeline dell'oggetto a cui il messaggio è diretto.

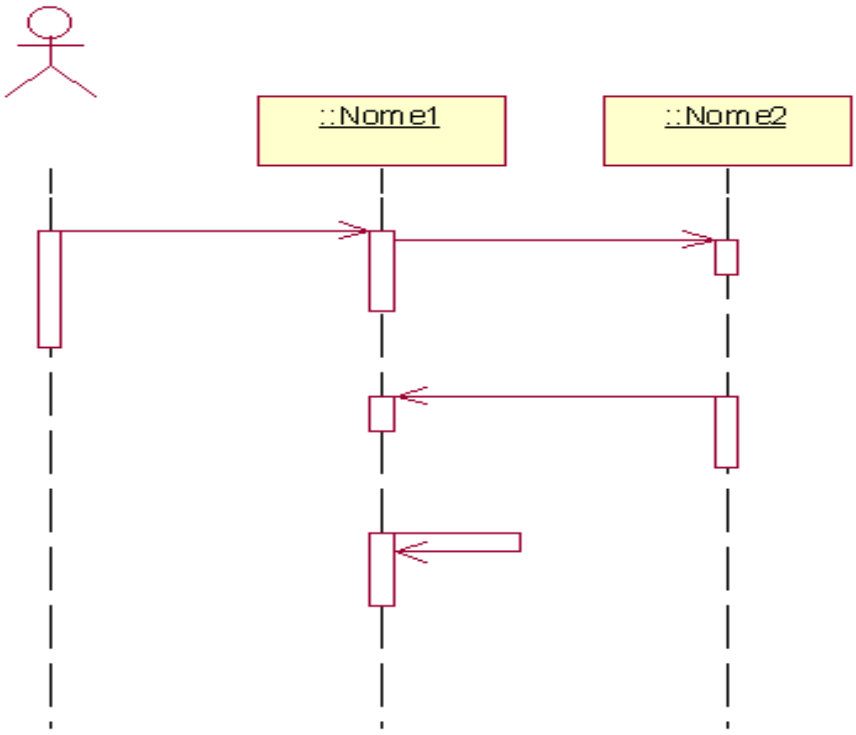
Si può anche verificare il caso in cui un oggetto mandi un messaggio a se stesso. Tale tipo di comportamento viene definito ricorsione.

Nella tabella seguente vengono identificati i possibili messaggi definiti in UML, per i sequenze diagrams:

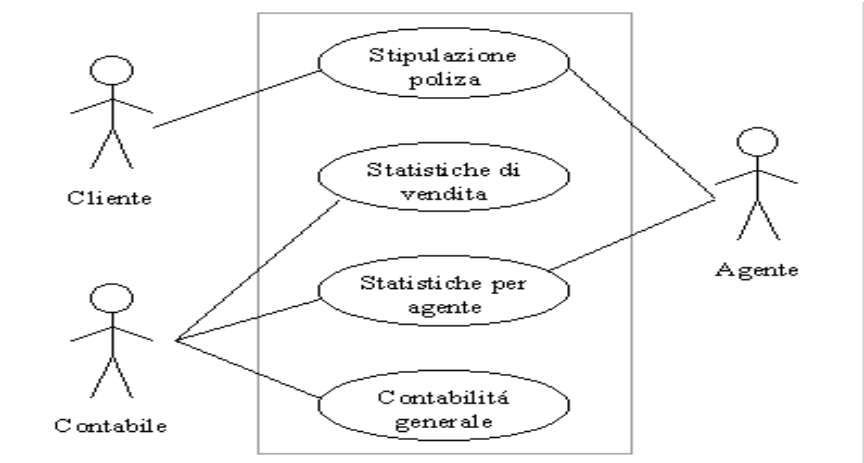
simple	Rappresenta il trasferimento del controllo da un oggetto ad un altro.	→
synchronous	Se un oggetto invia un messaggio sincrono, allora si attende che gli venga restituita una risposta al messaggio stesso prima di poter continuare con altre operazioni.	➡

asynchronous	Diversamente dai messaggi sincroni, se un oggetto invia un messaggio asincrono, non attende che gli venga inviata alcuna risposta prima di continuare con altre operazioni	→

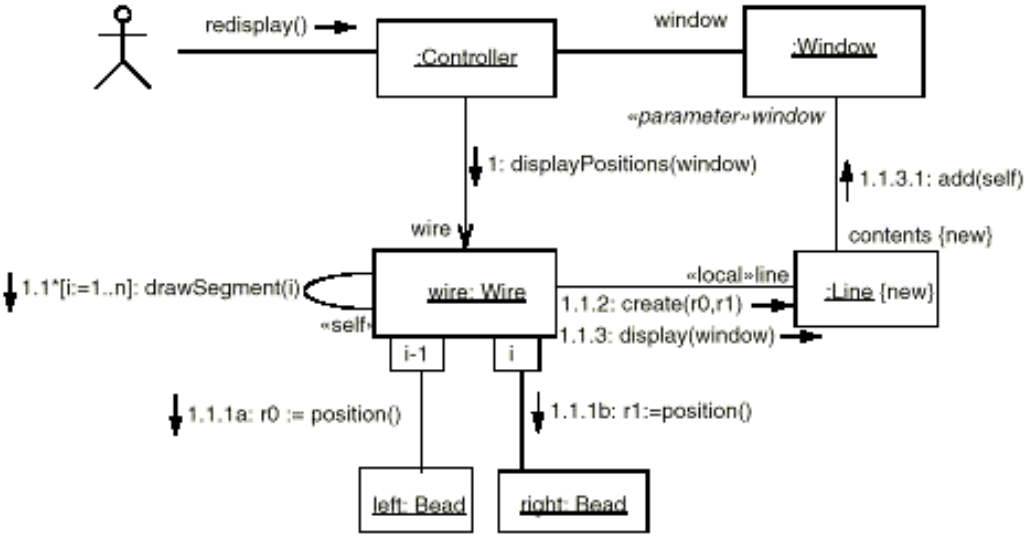
Il sequence diagram rappresenta il trascorrere del tempo se visto in direzione verticale. Il tempo, graficamente, viene fatto partire alla base di ogni oggetto e proseguire fino in basso. Ad esempio, un messaggio che si trovi più vicino all'oggetto rispetto ad un altro (rispetto alla direzione verticale), si verificherà prima nel tempo.



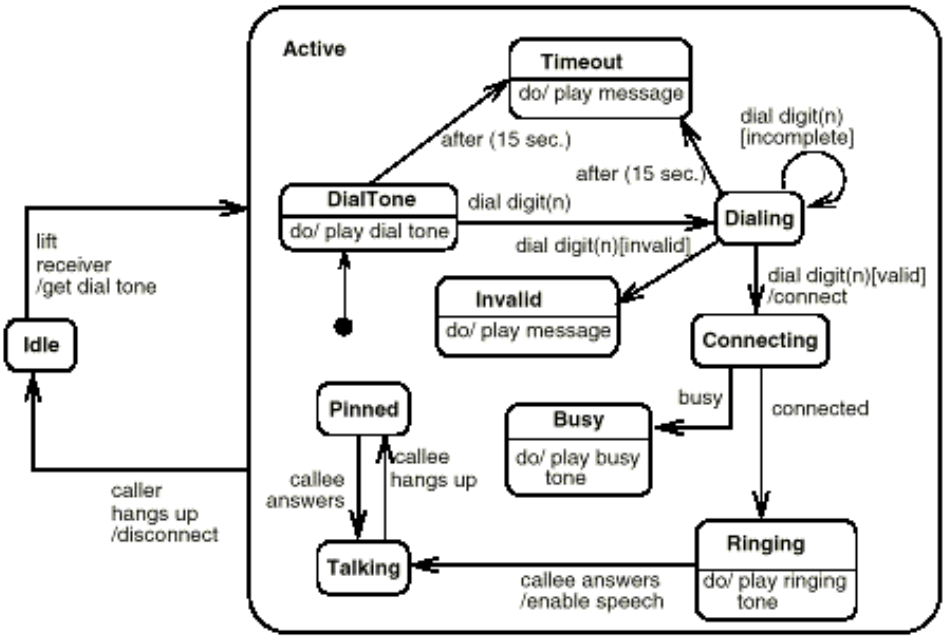
Lo Use case diagram definisce le possibili entità esterne, actor al sistema che interagiscono con esso, nonché i possibili scenari di utilizzo dello stesso e le relazioni, il tutto secondo una vista dinamica.



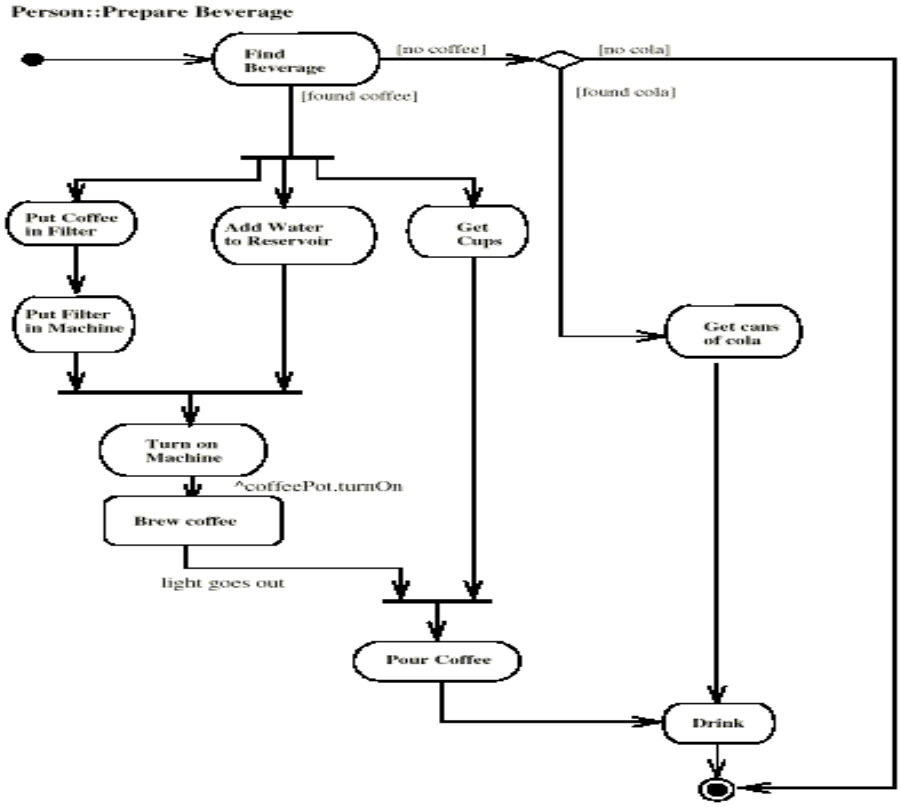
Il collaboration diagram è un modo differente per rappresentare le informazioni di un sequence diagram. In un collaboration diagram si mostrano oggetti e relazioni come fosse un object diagram e si annotano anche le relazioni con l'indicazione dei messaggi scambiati fra gli oggetti.



Uno statechart diagram può essere utilizzato per rappresentare l'evoluzione dello stato degli oggetti di una classe.

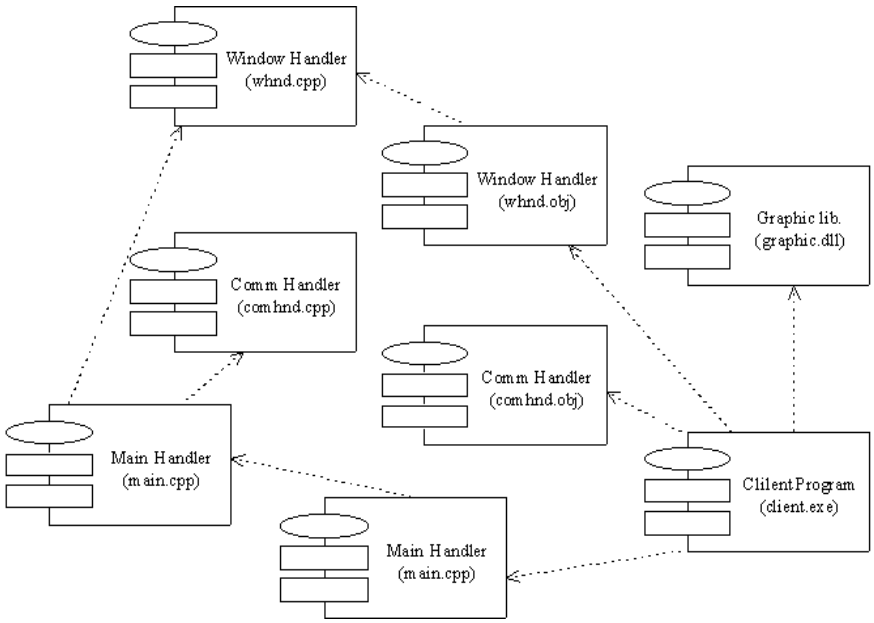


Gli activity diagram sono un'evoluzione dei flow chart e possono essere utilizzati per descrivere il comportamento di una singola operazione di una classe.

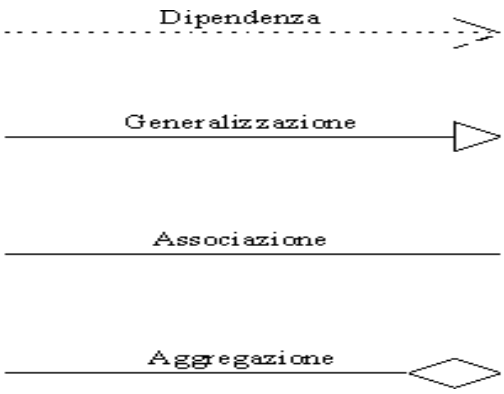


Per ciò che riguarda gli Implementation diagram esistono due tipi fondamentali:

1. Component diagram: mostrano la struttura del codice in termini di moduli, file sorgenti, ecc.
2. Deployment diagram: mostra la prevista distribuzione degli elementi del sistema fra i calcolatori su cui il sistema dovrà funzionare.



Esistono infine i cosiddetti Adornments, ossia elementi grafici che possono essere introdotti all'interno di dei diagrammi e che ne specificano alcune caratteristiche:



Un intricato esempio: jVEFrame-0.9.0.a

Utilizzando il programma Poseidon-1.3.1 (www.gentleware.com) è possibile osservare la creazione di un modello UML di jVEFrame-0.9.0.a, riportato nella figura seguente, il cui funzionamento verrà dettagliato sia nella appendice A, per ciò che riguarda il modello di partenza, sia nel capitolo seguente in cui si esporranno tutti gli sviluppi del modello intervenuti durante la stesura di questa opera.

Inserire figura

Conclusione

La stesura di un progetto mediante l'utilizzo di UML, garantisce sicuramente i vantaggi descritti in precedenza riguardanti la maggior chiarezza, l'effetto di rappresentazione verso l'acquirente od il finanziatore del prodotto finale e il riutilizzo del software. Tutto ciò nella prospettiva di uno staff di analisti, progettisti e programmatori, quindi in ambiti professionalmente avanzati e dinamici. Diversamente l'adozione di un tale strumento in ambiti più ristretti o comunque non finalizzati all'interazione fra più persone coinvolte nel medesimo progetto è ardua.

La difficoltà maggiore da superare rimane, comunque, un ulteriore passo di astrazione che andrebbe fatto, così come è successo con l'assembler, con i linguaggi imperativi e dichiarativi prima e con la programmazione orientata agli oggetti poi.

Infatti nonostante UML si basi su una tecnologia di programmazione orientata agli oggetti e consideri il problema posto nei termini che gli sono propri, pare che richieda un passo ulteriore nel modo di impostare la programmazione.

Sicuramente, per un esperto programmatore di Java, potrebbe sembrare un progetto utile solo al fine di rappresentare il proprio lavoro in una veste più sofisticata, provvedendo a scrivere il codice per ogni ulteriore modifica, ma forse occorrerebbe pensare alla programmazione UML come ad un modo diverso di impostare il ragionamento. Non solo occorre pensare in termini di oggetti,

provvedendo perciò a strutturare i singoli programmi con le rispettive funzionalità, ma le relazioni fra gli stessi ed eventuali ambienti o decisori, se si interpreta nel senso dello schema *ERA*, sono già definite nella struttura del linguaggio in maniera visuale e perciò, radicalmente diverse da ogni altro linguaggio sinora adottato.

Probabilmente solo la maggior diffusione di questo strumento e la sua sperimentazione da parte di un numero sempre maggiore di utenti, potrà smentire od avvalorare questa tesi che è solo il frutto di una analisi teorica del progetto e non tanto la constatazione in sede pratica dei costi e dei benefici.

Bibliografia

Alhir, S. S. (1998), *UML in a nutshell,* O'Reilly & Associates

Fuggetta, A. (2002), *UML, un dattiloscritto*