Lo standard UML

Mattia Grosso
Dipartimento di Informatica
Università Ca' Foscari di Venezia
mgrosso@dsi.unive.it

30 ottobre 2002

1 Introduzione a UML

UML è una notazione grafica per la rappresentazione, tramite diagrammi, di architetture software ad oggetti [11, 12]. Nacque nella seconda metà degli anni Novanta dallo sforzo congiunto di tre ricercatori della Rational Software [13], Rumbaugh, Jacobson e Booch, come fusione ed estensione di altri formalismi e linguaggi preesistenti [1, 2, 5, 6, 9, 10]. Lo scopo del lavoro era porre fine ad una disputa che si protraeva da qualche anno nella comunità degli sviluppatori su come impostare al meglio i progetti software orientati agli oggetti. Dopo una iniziale diffidenza verso un prodotto che cercava di imporsi come standard *de facto*, la versione 1.1 del 1997 ricevette alla fine il riconoscimento come standard *de jure* da parte dell'*Object Management Group* (OMG) [7], che, da quel momento in poi, gestì l'evoluzione di UML. Attraverso nuove modifiche ed estensioni, UML è attualmente giunto alla versione 1.4.

In questo documento sono esposti ed illustrati i vari diagrammi definiti da UML. Data la vasta gamma di opzioni disponibili nella notazione specificata dallo standard, vengono mostrati gli aspetti fondamentali di ogni diagramma, le componenti essenziali e il tipo di utilizzo. I diagrammi presentati sono:

- Diagrammi di casi d'uso;
- Diagrammi di classe, degli oggetti e di package;
- Diagrammi di interazione: sequenza e collaborazione;

- Diagrammi fisici: componenti e deployment;
- Diagrammi di stato;
- Diagrammi di attività.

Per illustrare i vari tipi di diagramma, è stato preso a modello un recente studio sulle prestazioni di CORBA [3].

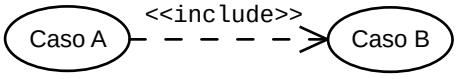
Come testo di riferimento per lo standard UML è stato usato il libro di Fowler e Scott, *UML Distilled* [4], adottandone la classificazione e la terminologia italiana di UML. Per una trattazione completa di tutti i dettagli del linguaggio, si vedano i testi di Rumbaugh, Jacobson e Booch [11, 12] e la documentazione ufficiale dell'OMG su Internet [8].

2 Diagrammi di casi d'uso

Lo scopo dei diagrammi di casi d'uso è descrivere ad alto livello l'interazione fra il sistema e uno o più *attori* che richiedono un servizio. Un attore rappresenta una qualsiasi entità (una persona reale, un altro sistema o una sua componente) che ha il diritto o la possibilità di entrare in contatto con il sistema software. Ogni attore è presente nel diagramma come un omino stilizzato che può interagire col sistema per vie diverse, generando così casi d'uso differenti. Ogni caso d'uso è una raccolta di una o più situazioni (*scenari*) logicamente raggruppabili che possono svilupparsi a partire da un operazione iniziale. I casi d'uso sono rappresentati da forme ovali, e sono collegati agli attori da linee rette: il collegamento può rappresentare, a discrezione dello sviluppatore, partecipazione o generazione di un caso d'uso.

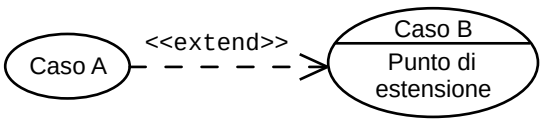
Per estendere il livello di dettaglio e la quantità di informazioni fornite dal diagramma, è possibile strutturare i casi d'uso in gerarchie:

- *inclusione*: indica se un caso d'uso ne include logicamente un altro. È rappresentata da una freccia tratteggiata, marcata con `<<include>>` e diretta verso il caso incluso:

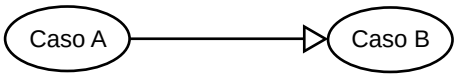


- *estensione*: indica una variazione precisa allo sviluppo normale di un caso d'uso. È rappresentata da una freccia tratteggiata, marcata con

`<<extend>>` e diretta verso il caso che viene esteso, nel quale vengono specificati i punti di estensione:



- *generalizzazione*: indica se un caso estende informalmente il comportamento di un altro. È rappresentata da una freccia non tratteggiata a punta piena verso il caso esteso:



Un diagramma di casi d'uso presenta tutti i possibili attori e casi presenti nella realtà del software studiato. Gli scenari interni ad ogni caso d'uso sono invece analizzati da altri diagrammi. Questo tipo di schema risulta molto comodo per dare una prima organizzazione alle proprie idee, iniziando a creare una struttura di alto livello sulla base di considerazioni molto pratiche.

Nel diagramma in Figura 1 sono illustrati i due attori tipici del sistema CORBA, le applicazioni cliente e servente, e quattro casi d'uso rappresentanti le situazioni di base che si possono verificare: l'inizializzazione dei lati cliente e servente di CORBA, l'invocazione di un metodo remoto con e senza dato di ritorno. In questo diagramma gli attori sono collegati ai casi d'uso generati: da questo punto di vista, l'unica attività autonoma dell'Applicazione servente è l'inizializzazione del suo lato di CORBA, mentre il suo lavoro di risposta alle richieste è una conseguenza interna agli altri casi d'uso generati dall'Applicazione cliente. Il caso Invocazione metodo include (funzionalmente) le due inizializzazioni ed è esteso dall'Invocazione metodo con dato di ritorno. Quest'ultimo eredita dal precedente le due inclusioni, anche se esse non vengono riportate esplicitamente nel diagramma (vedi anche Fig. 10).

3 Diagrammi di classe, di package e degli oggetti

La modellizzazione della programmazione orientata agli oggetti è il punto forte di UML: in questa sezione sono presentati tre tipi di diagrammi di diverso livello, che permettono di esprimere in maniera chiara ed efficiente un vasto insieme di concetti e proprietà, dai più semplici ai più avanzati.

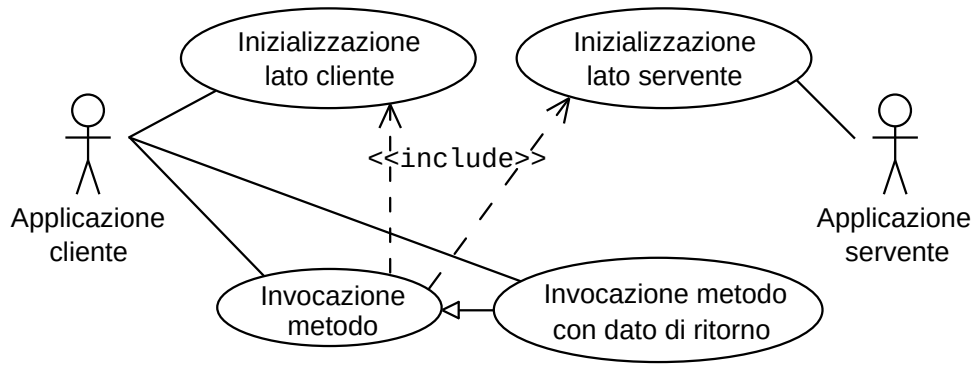
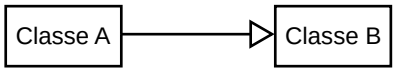


Figura 1: Diagramma UML di casi d'uso per il sistema cliente-servente definito in CORBA

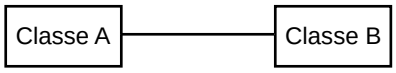
3.1 Diagrammi di classe

I diagrammi di classe sono il nucleo portante di UML. Il loro compito è mostrare le relazioni fra le classi che lo sviluppatore definisce per il proprio progetto. Ogni classe è rappresentata da un rettangolo diviso al più in tre parti, per il nome, gli attributi e i metodi. Le regole di visibilità sono + (pubblico), # (protetto) e – (privato), e vanno indicate prima del nome di un elemento di classe. È possibile definire un'ampia gamma di relazioni statiche e dinamiche nella struttura delle classi, in particolare:

- *generalizzazione*: quando una classe è sottoclasse di un'altra, la relazione viene indicata da una freccia con punta a triangolo bianco:



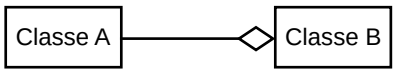
- *associazione*: due classi sono associate se nel sistema sono in relazione fra di loro. Il legame viene espresso da una linea retta che congiunge le classi associate:



  Ai due capi della linea possono anche essere indicate delle molteplicità diverse dalla "uno a uno" di base, tramite un numero (molteplicità esatta), un asterisco (m. infinita) o un intervallo (m. opzionale, ad es.,

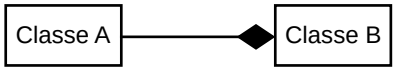
`0..2`). Le rette di congiunzione possono diventare frecce per indicare la direzione del legame.

- *aggregazione*: due classi sono aggregate se una rappresenta logicamente una parte dell'altra. Il legame è segnalato da una retta con un piccolo rombo bianco dal lato della classe includente:



  Le opzioni di molteplicità sono le stesse del punto precedente.

- *composizione*: come per l'aggregazione, ma la classe parte è vincolata ad un'unica classe contenitore. La notazione prevede un rombo nero al posto di quello bianco:



- *stereotipazione*: UML permette di definire nuovi costrutti di modellazione che vengono chiamati *stereotipi*. Quando una classe ne implementa uno (ad esempio, un'interfaccia), questo deve essere indicato fra $\ll$ e $\gg$ assieme al nome della classe stessa.

In Figura 2 viene illustrato un diagramma di classe ad alto livello per le componenti di CORBA. Ogni classe ha un suo preciso stereotipo, che in questo caso deve essere letto come una prima classificazione informale delle parti del sistema: un `Componente` è una parte software che svolge una certa attività all'interno di CORBA, un `Connettore` è un componente specializzato nella comunicazione, mentre una `Risorsa` è una parte hardware che viene utilizzata dal software. Le relazioni fra le classi sono tutte associazioni "uno a uno", a parte quelle "uno a molti" fra connettori e ORB. Queste ultime indicano che ad ogni ORB devono essere collegati almeno un Connettore Cliente e un Connettore Servente.

3.2 Diagrammi di package

I diagrammi di package modellano il rapporto fra i package di un sistema software. Il termine package in UML indica un qualsiasi insieme di parti che formano un gruppo autonomo. In generale, questo spesso coincide con i package che è possibile trovare nei linguaggi di programmazione come Java (un

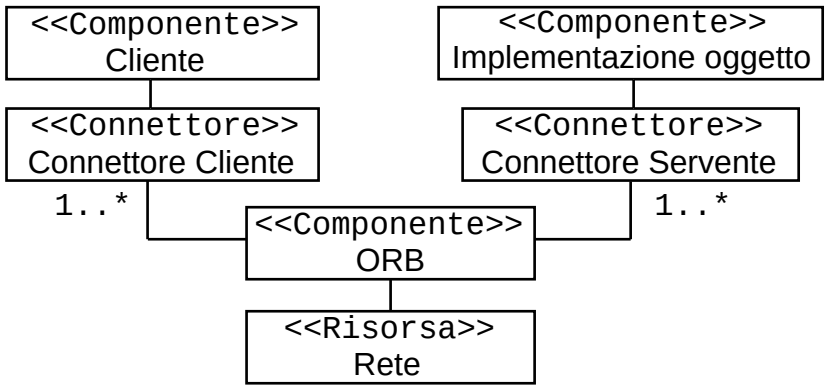
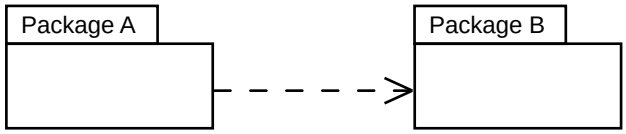


Figura 2: Diagramma UML di classe per le componenti di CORBA

package Java è una raccolta di classi che formano un pacchetto autonomo, riutilizzabile in altri progetti e con uno spazio dei nomi indipendente). I package sono indicati come cartelle e le relazioni di dipendenza vengono rappresentate da frecce tratteggiate:



Spesso questo tipo di diagramma viene presentato assieme a diagrammi di classe, come è illustrato nel prossimo esempio.

Il diagramma di classe in Figura 3 espande una parte della Figura 2, illustrando la composizione interna del Connettore Servente. Tale `Connettore` è in realtà un package contenente quattro classi diverse, POA Manager, POA, Servant e Servente. Le classi sono indicate solo con i nomi, senza attributi e metodi. Possiamo notare numerose relazioni fra le classi; in particolare, ogni POA è incluso in almeno un POA Manager e un Servente (aggregazioni), ha associato almeno un Servant (associazione) e può essere legato ad altri POA (composizione).

3.3 Diagrammi degli oggetti

I diagrammi degli oggetti (o *delle istanze*) sono una rappresentazione dinamica del software, utile per visualizzare lo stato di attività del proprio sistema.

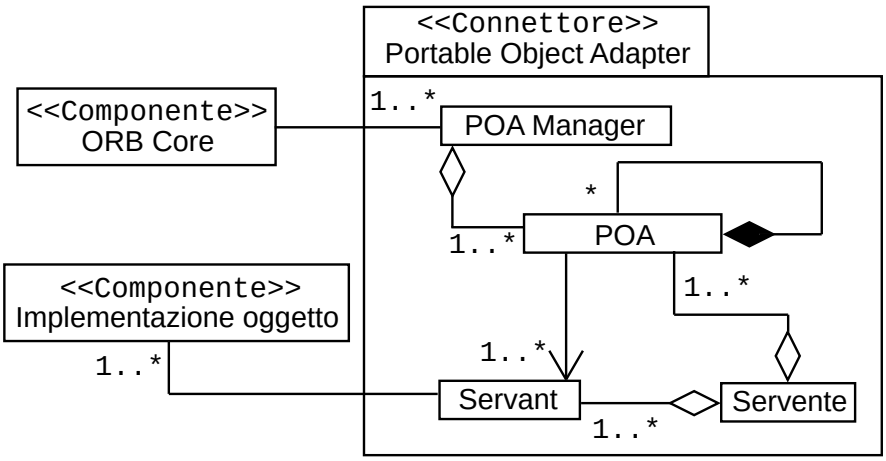


Figura 3: Diagramma UML di classe e package per la componente POA di un sistema CORBA

I diagrammi mostrano, infatti, gli oggetti (implementazioni di classi) attivi in un determinato momento dell'esecuzione. Gli oggetti vengono indicati con la notazione `nome:classe` (anche parziale, solo nome o classe), e possono eventualmente riportare dei valori distintivi di attributo. Fra gli oggetti sono generalmente presenti dei collegamenti tramite linee rette, che derivano dalle relazioni di associazione, composizione o aggregazione definite fra le classi originali (vedi § 3.1).

In Figura 4 viene mostrato un possibile diagramma degli oggetti relativo alle classi definite in Figura 3. In questo caso si suppone che siano presenti e attivi un RootPOA, un Servente, due POA Manager e due POA. Oltre a questi, è presente un numero imprecisato di Servant che gestiscono varie Implementazioni Oggetto, entrambi i gruppi disegnati come pile per semplificare il diagramma. Come si può vedere dalla figura, il diagramma tende a diventare complesso e difficile da leggere anche con pochi elementi.

4 Diagrammi di interazione

I diagrammi di interazione si usano per descrivere la comunicazione e l'interrelazione fra le componenti di un sistema.

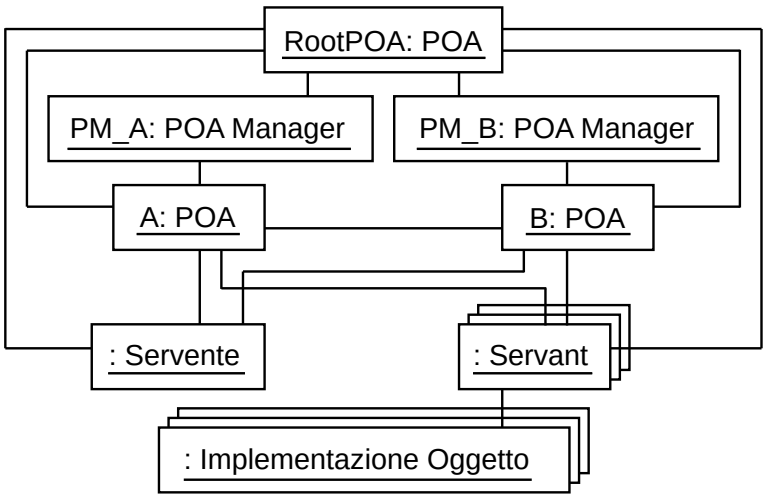


Figura 4: Diagramma UML degli oggetti per le classi coinvolte nei POA

4.1 Diagrammi di sequenza

Un diagramma di sequenza mostra lo scambio di messaggi fra tutti gli oggetti che vengono via via coinvolti in una attività iniziata da un attore. In altre parole, un diagramma di sequenza espande e specifica un caso d'uso (vedi § 2). Se lo sviluppo di una certa attività non è troppo complesso, è possibile visualizzare più possibili percorsi (scenari) in un unico diagramma di sequenza, ponendo delle scelte condizionali. In caso contrario, è consigliabile tracciare un diagramma per ogni possibile strada.

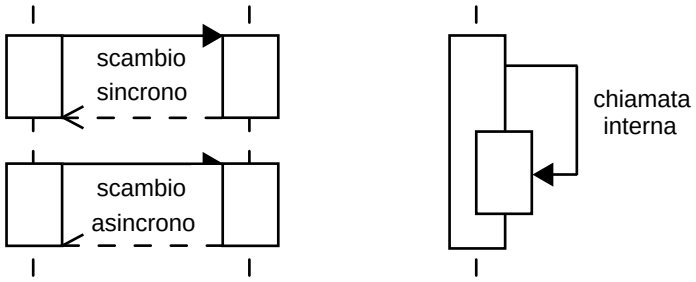
Il diagramma si sviluppa dall'alto verso il basso seguendo un'ipotetica linea temporale. Gli oggetti (istanze di classi) che partecipano all'esecuzione vengono tutti indicati nella parte alta del diagramma con un rettangolo che ne riporta il nome e/o il tipo (la notazione è la stessa dei diagrammi degli oggetti, § 3.3). La vita di ogni oggetto viene rappresentata da una linea tratteggiata verticale: su di essa possono essere tracciati stretti rettangoli più o meno alti, ad indicare i periodi di effettiva attività dell'oggetto.

Gli oggetti possono essere creati o distrutti da altri oggetti in corso di esecuzione. Un oggetto ne può creare un altro attraverso uno speciale messaggio di creazione: diversamente dagli altri, posizionati nella parte alta del diagramma, il nuovo oggetto viene inserito in mezzo al diagramma, all'altezza

del messaggio che lo ha generato. La distruzione di un oggetto viene invece indicata da una grande X sulla linea di vita: un oggetto può deallocarsi autonomamente, oppure essere distrutto tramite un messaggio.

Lo scambio dei messaggi è indicato con delle frecce fra le linee di vita di due oggetti. Le frecce sono generalmente marcate col contenuto (logico) del messaggio, ma prima di questo possono essere presenti delle ulteriori diciture. Una scrittura fra parentesi quadre indica un test condizionale il cui risultato determina l'invio del messaggio. Un'istruzione di assegnazione specifica che il risultato del messaggio diventa il valore di una certa variabile. Infine, un asterisco (*) segnala che l'operazione di invio deve essere iterata.

Le frecce hanno significati diversi in base al modo in cui sono disegnate. Le frecce dirette a punta piena indicano messaggi sincroni, mentre quelle tratteggiate sono risposte di ritorno. Quando hanno solo mezza punta rappresentano invece i messaggi asincroni. Sono possibili anche frecce che ritornano nell'oggetto da cui partono, per indicare chiamate a funzioni interne:



In Figura 5 viene presentato il diagramma di sequenza che espande il primo caso d'uso definito in Figura 1. L'oggetto Servente si occupa di iniziare uno scambio di messaggi che porta all'attivazione di tutte le componenti necessarie al funzionamento del sistema CORBA sul lato servente. Sono chiaramente visibili i messaggi di invio marcati con chiamate di funzioni, e quelli di ritorno con le conferme; tutti i messaggi sono sincroni. Il terzo messaggio inviato da Servente porta alla creazione dell'oggetto Servant, che da quel momento in poi diventa parte del diagramma.

4.2 Diagrammi di collaborazione

I diagrammi di collaborazione analizzano le stesse situazioni dei diagrammi di sequenza, evidenziando maggiormente i legami fra le varie componenti. Si possono considerare come un'estensione dei diagrammi degli oggetti (§ 3.3),

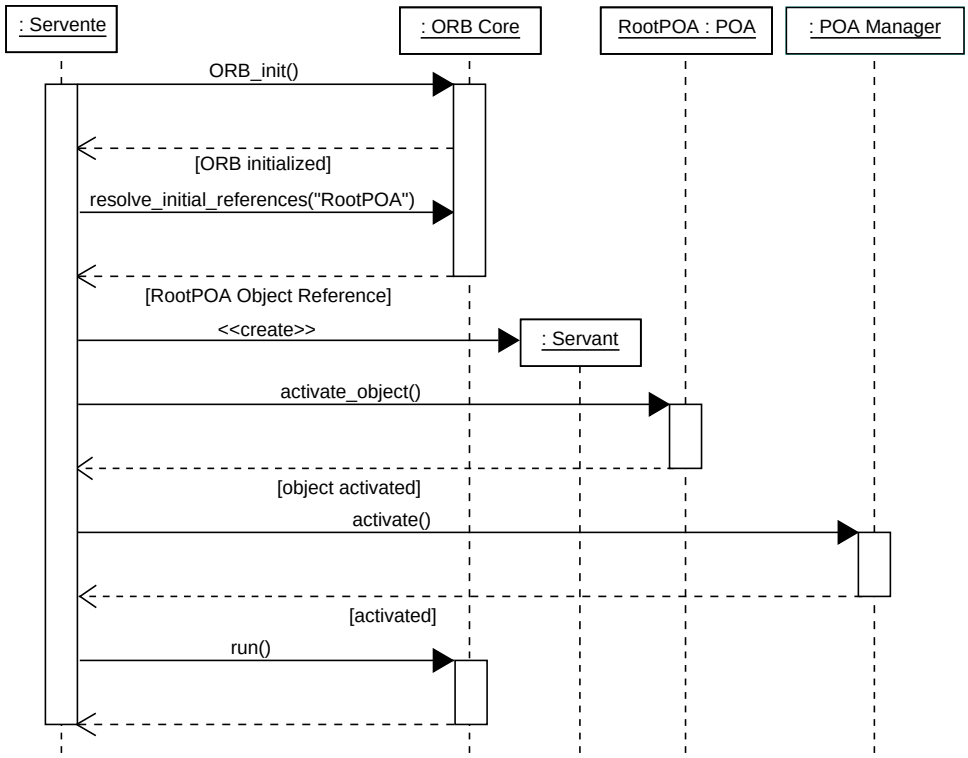


Figura 5: Diagramma UML di sequenza per l'inizializzazione del lato servente del sistema CORBA

dai quali ereditano la notazione. Vicino ad ogni collegamento fra oggetti sono presenti i messaggi scambiati, preceduti da una numerazione decimale (eventualmente gerarchica) che permette di ricostruire la storia della collaborazione. Piccole frecce indicano la direzione dei messaggi, che possono essere scritti con le stesse opzioni disponibili per i diagrammi di sequenza.

Per sua natura, questo tipo di diagramma tralascia una serie di informazioni sulla vita degli oggetti che troviamo invece all'interno dei diagrammi di sequenza. Perciò è sempre possibile ricavare un diagramma di collaborazione da un diagramma di sequenza, osservando che una qualsiasi comunicazione fra due oggetti di un diagramma di sequenza genera uno dei legami riportati nel diagramma di collaborazione.

In Figura 6 è possibile vedere il diagramma di collaborazione ottenuto da

quello di sequenza in Figura 5. Gli oggetti coinvolti e i messaggi scambiati sono rimasti gli stessi, ma l'attenzione del diagramma è chiaramente spostata sul collegamento fra le componenti generato dalla comunicazione.

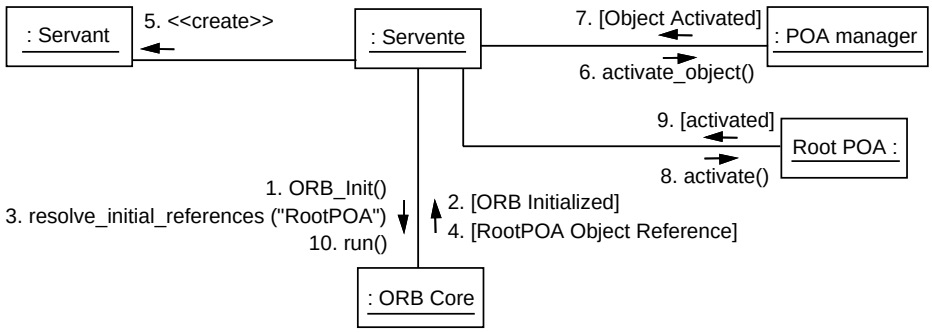


Figura 6: Diagramma UML di collaborazione ricavato dal diagramma di sequenza in Figura 5

5 Diagrammi fisici

I diagrammi fisici sono diagrammi di alto livello utilizzati per mostrare l'implementazione del software su una certa architettura hardware. Diventano essenziali per tutte quei progetti in cui il sistema software è strettamente dipendente dall'architettura fisica dove dovrà funzionare.

5.1 Diagrammi dei componenti

I diagrammi dei componenti individuano le relazioni presenti, a livello di codice, fra componenti software. I componenti sono parti logicamente autonome di un sistema software: possono corrispondere all'implementazione di un package, oppure essere istanze di classi di diversa origine. Fra componenti si possono definire numerose relazioni, ad esempio di comunicazione o di compilazione, e il diagramma può essere esteso mostrando gli oggetti che fanno parte di ogni singolo componente. Sono interessanti per uno studio dei legami fra le varie entità software attive nel sistema.

Il diagramma di Figura 7 mostra i componenti del sistema cliente-servente CORBA (Cliente, Servente, Cliente ORB e Servente ORB) con i rispettivi

oggetti interni. Le direzioni delle dipendenze nascono da una operazione di richiesta da parte del Cliente.

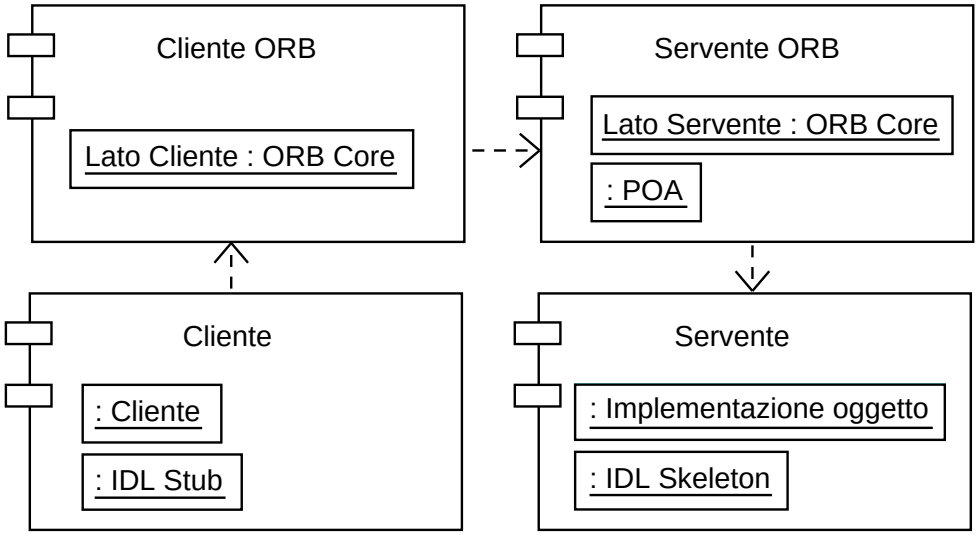


Figura 7: Diagramma UML di componenti per l'architettura cliente-servente CORBA

5.2 Diagrammi di deployment

I diagrammi di deployment (schieramento, distribuzione) presentano le varie risorse hardware coinvolte dall'esecuzione del software. A seconda del tipo di sistema (centralizzato, distribuito o misto), abbiamo naturalmente componenti differenti: le macchine sono rappresentate come parallelepipedi, i collegamenti con linee rette. Per ciascuna parte vengono indicati (in maniera semplificata) la natura e il collegamento con le altre parti del sistema.

I due tipi di diagrammi fisici vengono generalmente usati e presentati assieme (con il nome di *diagramma di deployment* o *di implementazione*), mostrando come le componenti di un software si distribuiscono sul sistema hardware prescelto.

Il diagramma fisico combinato di Figura 8 mostra una generica implementazione ad alto livello del sistema CORBA su due macchine cliente e servente

collegate da una rete. Sono illustrati i componenti senza dettaglio interno (vedi Figura 7) e l'infrastruttura hardware di appoggio.

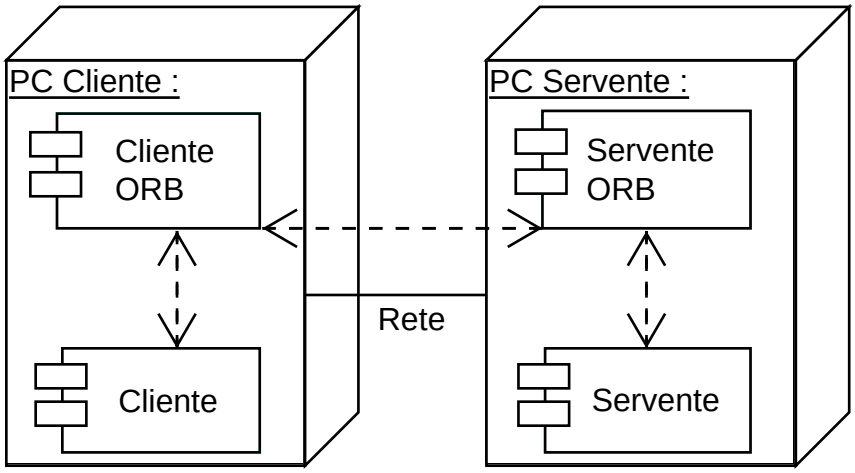


Figura 8: Diagramma UML di deployment per l'architettura cliente-servente CORBA

6 Diagrammi di stato

Questo tipo di diagramma illustra la vita di un oggetto software durante l'attività del sistema, mostrando tutti i possibili stati in cui esso può venirsi a trovare, e come avviene il passaggio fra gli stati. Ogni diagramma inizia da un punto di partenza indicato da un piccolo cerchio nero, e le transizioni sono indicate da semplici frecce. Sono possibili transizioni ricorsive verso lo stesso stato. Gli stati possono essere di due tipi:

- *semplice*: lo stato è caratterizzato solo da un nome. La transizione verso un altro stato è determinata dal verificarsi di un certo evento, segnato sulla freccia di cambio di stato. Se sono definite transizioni verso più stati differenti, allora ogni freccia uscente viene marcata da eventi, test condizionali (fra parentesi quadre) ed eventuali operazioni (precedute da una barra, /);

- *con attività*: lo stato ha una operazione associata, il cui nome inizia con `do/`, che deve essere eseguita quando l'oggetto entra in tale stato, o ci

rimane dopo un test condizionale. Il movimento verso gli altri stati è lo stesso del caso precedente.

Il passaggio fra stati è quindi strettamente deterministico. La vita di un oggetto termina quando esso arriva in uno stato finale, cioè senza frecce uscenti. Sono naturalmente possibili più stati finali.

Le transizioni e le azioni degli stati possono essere ulteriormente estese, e gli stati possono essere riuniti in superstati più ampi. Inoltre, all'interno di un superstato possono essere descritti percorsi diversi che devono svolgersi in parallelo. A meno di eventi specificatamente indicati, l'uscita da un superstato con stati concorrenti avviene solo quando tutte le sue transizioni interne arrivano a conclusione.

Questo tipo di diagramma è particolarmente indicato per descrivere il comportamento di un oggetto che si trova coinvolto in casi d'uso diversi (§ 2), perché permette di dare un'immagine globale evidenziando eventuali problemi non visibili localmente. Per descrivere l'attività di più oggetti contemporaneamente, è invece preferibile rivolgersi ad altri tipi di diagramma, quali quelli di interazione (§ 4) o di attività (§ 7).

In Figura 9 è illustrato un diagramma di stato relativo al lato cliente dell'ORB Core di CORBA. I cambiamenti di stato sono tratti dal terzo e dal quarto caso d'uso del diagramma in Figura 1. Si possono vedere uno stato semplice al centro, e due stati con attività. Manca uno stato finale perché si suppone che dopo l'attivazione l'ORB Core rimanga sempre attivo.

7 Diagrammi di attività

I diagrammi di attività vengono utilizzati per illustrare le sequenze di operazioni che possono nascere da una richiesta o un intervento sul sistema. Sono utili per visualizzare a quali conseguenze e sviluppi può portare una certa operazione. I diagrammi di attività nascono come evoluzione dei classici diagrammi di flusso (*flowchart*), dai quali ereditano la possibilità di definire flussi che dipendono da test condizionali. L'attività ha il suo punto di partenza in un piccolo cerchio nero, e termina in un altro cerchio nero ricerchiato. Le operazioni sono indicate da degli ovali e il flusso da frecce semplici. I salti condizionali (*branch*) sono visualizzati come dei rombi, dai quali escono due o più

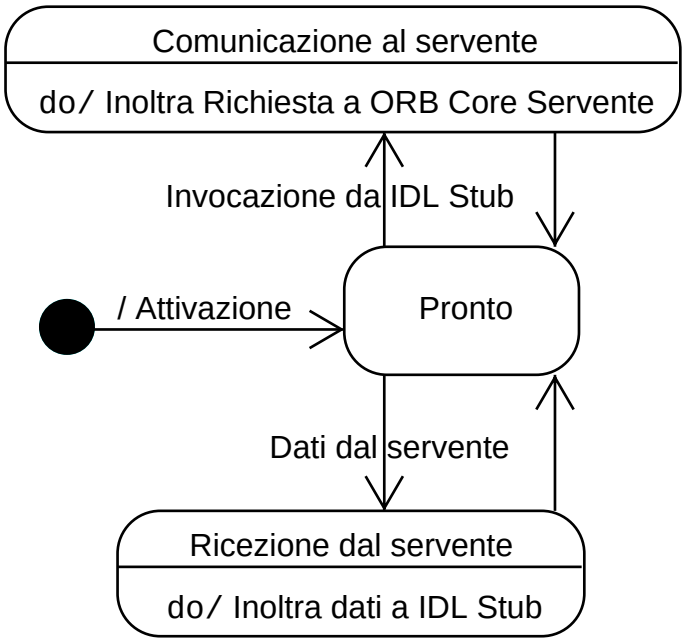


Figura 9: Diagramma UML di stato per il lato cliente dell'ORB Core

nuovi flussi, ognuno marcato da una condizione (come al solito, fra parentesi quadre). I nuovi flussi si riuniscono prima o poi in un altro rombo (*merge*).

A questa normale strutturazione, UML aggiunge un'importante novità: la possibilità di descrivere computazioni parallele. Questi flussi di operazioni si sviluppano contemporaneamente da un certo punto di diramazione (*fork*), indicato da uno spesso segmento nero orizzontale. Da quel punto in poi continuano come flussi normali, fino a quando non raggiungono il punto di riunione (*join*), graficamente uguale al fork, dove vengono ricomposti in un unico flusso. La differenza fra merge e join è che, nel secondo caso, la normale sequenza di operazioni riprende solo quando tutti i flussi paralleli vengono completati.

In maniera simile a quanto visto per i diagrammi di stato (§ 6), nei diagram-

mi di attività è possibile isolare parti del flusso in sotto-attività indipendenti. Questo permette di semplificare diagrammi complessi e di favorire il riutilizzo di parti del diagramma, rendendole indipendenti dal diagramma padre.

Questo tipo di diagrammi risulta efficace per rappresentare contemporaneamente l'attività complessiva di più parti del sistema, ad esempio quelle coinvolte in un caso d'uso (§ 2). La possibilità di descrivere flussi paralleli è molto pratica per descrivere sistemi che usano al loro interno meccanismi di *multithreading*.

La Figura 10 illustra un diagramma di attività relativo all'operazione di richiesta di un cliente CORBA. La sequenza di operazioni parte dall'applicazione cliente e, dopo un controllo sullo stato dell'ORB Core, coinvolge oggetti e componenti differenti disposti sul lato cliente e servente. La seconda scelta condizionale individua le richieste che necessitano di una risposta, e quindi di un'ulteriore sequenza di operazioni.

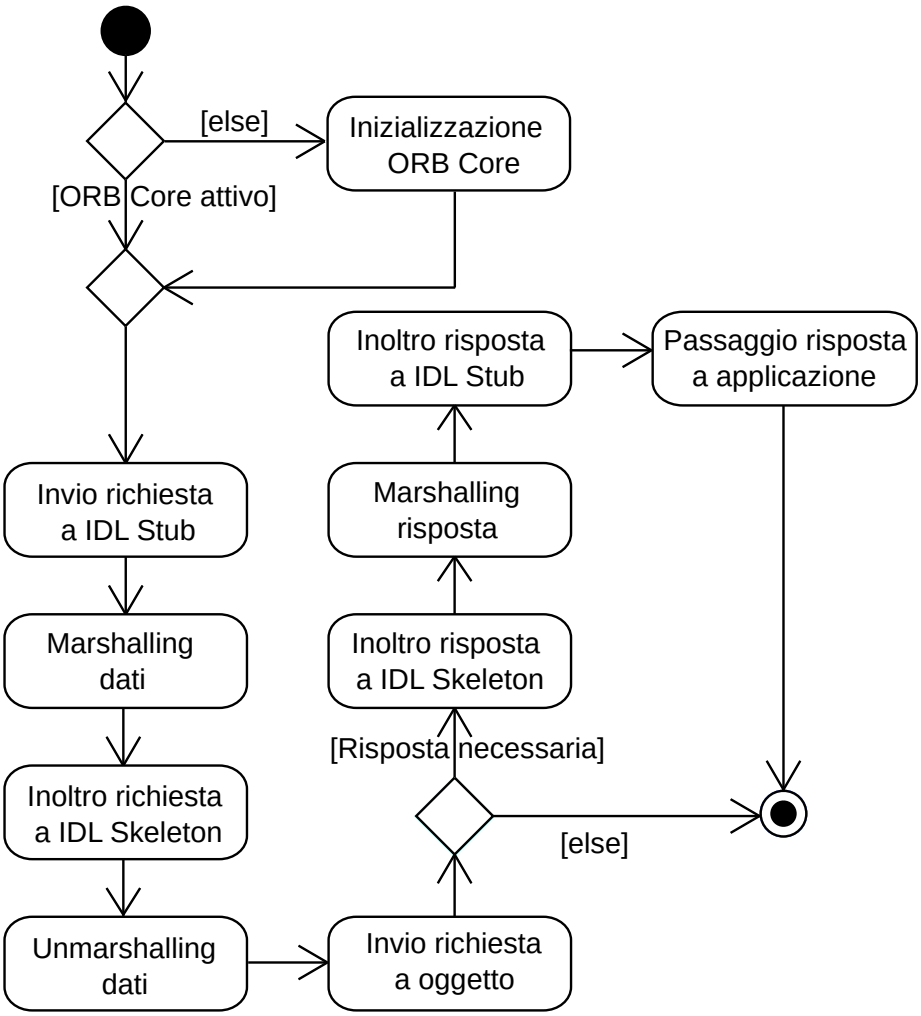


Figura 10: Diagramma UML di attività per una richiesta di un cliente CORBA

Riferimenti bibliografici

[1] G. Booch. *Object-Oriented Analysis and Design with Applications. Second Edition.* Addison-Wesley, 1994.

[2] G. Booch. *Object Solutions: Managing the Object-Oriented Project.* Addison-Wesley, 1996.

[3] F. Chicchiriccò. Performance evaluation of a corba-based distributed object system – design. Technical report, Nokia Research Center, Helsinki, Finland, 19/10/2001.

[4] M. Fowler and K. Scott. *UML Distilled. Prima edizione italiana.* Addison-Wesley, 2000.

[5] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Enginnering: A Use Case Driven Approach.* Addison-Wesley, 1992.

[6] I. Jacobson, M. Ericsson, and A. Jacobson. *The Object Advantage: businness Process Reengineering with Object Technology.* Addison-Wesley, 1995.

[7] Object Management Group (OMG). `http://www.omg.org/`.

[8] Object Management Group (OMG). Unified Modeling Language (UML). `http://www.uml.org/`.

[9] J. Rumbaugh. *OMT Insights.* SIGS Books, 1996.

[10] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenzen. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[11] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1999.

[12] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language User Guide.* Addison-Wesley, 1999.

[13] Rational Software. `http://www.rational.com/`.