Capitolo 9

Nozioni generali sulla modularizzazione

La modularizzazione è la tecnica di progettazione che consente di costruire programmi costituiti da parti indipendenti e interagenti. Essa costituisce uno strumento essenziale per razionalizzare il progetto del software, e per avere un buon controllo del processo di sviluppo anche quando il programma è di grandi dimensioni.

La modularizzazione sta assumendo sempre più importanza nella progettazione dei programmi, specialmente in relazione alla sua rilevanza nella realizzazione di programmi riusabili. In questo capitolo discutiamo la nozione di modulo, e illustriamo i principali criteri per giudicare la bontà di un processo di modularizzazione, mettendoli in relazione con le qualità dei programmi. Successivamente, presentiamo una classificazione dei possibili tipi di modularizzazione, e discutiamo delle caratteristiche del processo di modularizzazione dei programmi nelle due fasi della progettazione.

9.1 La nozione di modulo

L'idea basilare della modularizzazione è di strutturare il programma in parti autonome, chiamate appunto *moduli*. Un modulo è un'unità di programma con una sua struttura interna, definito con un determinato scopo, che offre all'esterno un certo insieme prefissato di servizi utilizzabili da altri moduli, e che ha precise relazioni con altri moduli. Un modulo è quindi caratterizzato da:

1. l'insieme dei servizi che esso esporta, ovvero che offre agli altri moduli;
2. la modalità con cui tali servizi possono essere effettivamente utilizzati; l'insieme di tali modalità costituisce quella che viene chiamata l'*interfaccia* del modulo;
3. l'insieme dei servizi che esso importa dagli altri moduli, ovvero l'insieme dei servizi offerti da altri moduli ed utilizzati dal modulo per le sue funzioni;
4. la struttura interna, cioè l'insieme dei tipi, delle variabili e delle funzioni definiti nel modulo stesso, ma non visibili ed utilizzabili all'esterno.

Un modulo è perciò allo stesso tempo un *fornitore* di servizi ed un *cliente*, in quanto utilizzatore di servizi forniti da altri moduli[1].

L'uso dei moduli permette di procedere nello sviluppo del programma decidendo prima l'architettura globale, e concentrandosi poi su ogni singolo modulo, sulle sue funzioni e sulle sue caratteristiche. Ciò favorisce anche uno sviluppo razionale da parte di un gruppo di progettisti. Ogni progettista può essere responsabile di un modulo, e può procedere sulla base della specifica di quali sono i servizi che il suo modulo deve offrire, e quali invece sono i servizi che gli altri moduli offriranno una volta progettati e realizzati. Se la suddivisione in moduli è stata effettuata con criterio, ogni modulo può effettivamente essere sviluppato indipendentemente dagli altri, e può essere analizzato e verificato autonomamente, per mezzo di test che prevedano di attivare i servizi da esso offerti.

La modularizzazione favorisce lo sviluppo di programmi di alta qualità. In particolare:

- La verifica di correttezza si giova della modularizzazione. Se il programma complessivo rivela errori durante l'esecuzione, è in genere più facile isolare e quindi individuare l'errore stesso, perché è possibile risalire al servizio che ha dato origine al comportamento errato, e quindi al modulo responsabile di tale servizio.
- La riusabilità del software aumenta nella misura in cui si riesce a progettare moduli ben strutturati, con servizi ben definiti ed utilizzabili con facilità. Il programma non è un blocco monolitico, ma è costituito da un insieme di parti che sono potenzialmente, utilizzabili in contesti diversi.

Il concetto di modularizzazione si ritrova non solo nella progettazione del software, ma in tutte le organizzazioni e in tutti i sistemi. In questi contesti, un modulo è semplicemente una parte dell'organizzazione o del sistema, con un suo insieme preciso di funzioni, e con determinate modalità di interazione con gli altri moduli. Ad esempio, una università è strutturata in facoltà, dipartimenti, e settori, che possono essere visti come parti del sistema complessivo, con determinate funzioni e compiti. Una banca è organizzata in agenzie, ed ognuna di esse è strutturata in sottoparti, responsabili dei vari servizi alla clientela. Una nazione è strutturata in regioni, province e comuni. Ciascuna di queste parti ha determinati compiti e responsabilità, ed interagisce con le altre strutture in modi prefissati. Un libro è suddiviso in capitoli, in paragrafi, sottoparagrafi, e così via. Ogni parte, ad esempio un capitolo, svolge una sua funzione, allo stesso tempo interrelata e indipendente rispetto a quella delle altre parti.

Esercizio 9.1 Si consideri un ufficio pubblico (ad esempio un ufficio postale), e lo si descriva in termini di un insieme di moduli. Ogni modulo deve essere caratterizzato secondo quanto descritto in precedenza, ovvero dai servizi che offre, dal modo in cui i clienti possono usufruirne, dalla sua struttura interna, e dal rapporto con gli altri moduli. □

[1] Per questa ragione, per denotare le architetture software basate su moduli si utilizza spesso la dizione inglese *client/server*.

9.3 Tipi di modularizzazione

Possiamo a questo punto rendere più concreto lo scenario dell'uso dei moduli nel progetto dei programmi, e tentare di classificare, seppure in modo molto schematico, le possibilità di utilizzo di moduli. Dal punto di vista metodologico è importante che la struttura a moduli del programma corrisponda ad una decomposizione concettuale significativa del programma stesso. In altre parole, un modulo deve *incapsulare* un preciso insieme di proprietà e caratteristiche, e le deve descrivere completamente ed efficacemente. Per indirizzare il processo di modularizzazione verso scelte razionali, si individuano tre tipi di modularizzazione:

1. modularizzazione *funzionale*, quando in un modulo viene concentrato un insieme omogeneo di funzionalità, ovvero quando il modulo offre all'esterno un certo insieme di funzioni omogenee rispetto ad un obiettivo prefissato;

2. modularizzazione *per oggetto*, quando un modulo gestisce un particolare oggetto, ovvero quando il modulo offre all'esterno un insieme di operazioni che servono ad utilizzare un determinato oggetto in modo corretto, nascondendo all'esterno gli aspetti relativi allo stato interno dell'oggetto stesso che non hanno rilevanza rispetto ai servizi offerti;

3. modularizzazione *per tipo astratto*, quando un modulo offre la definizione e la rappresentazione di un tipo astratto, ovvero quando il modulo offre all'esterno la possibilità di definire oggetti (detti anche *istanze*) di un tipo, e di utilizzarli secondo una opportuna specifica astratta, nascondendo tutte le proprietà che riguardano la rappresentazione del tipo stesso.

Nel seguito, approfondiamo i tipi di modularità descritti, discutendo diversi criteri di modularizzazione per ciascuno di essi. Alla modularizzazione per tipo astratto dedicheremo poi i prossimi capitoli di questa parte.

La modularizzazione funzionale si adotta quando si ritiene utile raccogliere in una unità di programma un insieme di funzioni tra loro omogenee, cioè tutte e sole quelle risolventi un sottoproblema individuato dalla decomposizione del problema in gioco.

Ad esempio, se nel progetto di un programma raccogliamo in un modulo S tutte le procedure di stampa di messaggi in dispositivi di uscita, operiamo una modularizzazione funzionale. In questo caso la coesione è alta perché la nozione che unisce le componenti del modulo riguarda un tipo di operazione (la stampa) a cui gli altri moduli dovranno ricorrere. L'accoppiamento riguarderà in questo caso le modalità di comunicazione tra S e gli altri moduli. Una ragionevole modalità di comunicazione con un modulo M è quella che prevede che M fornisca ad S:

1. il messaggio da produrre,
2. il riferimento logico (ad esempio un codice) del dispositivo su cui il messaggio deve essere prodotto.

Ragionando a livello intuitivo, possiamo concludere che queste sono le informazioni minime e necessarie per utilizzare in modo soddisfacente le funzionalità offerte da S. Basandosi sul riferimento logico del dispositivo, S avrà tutti gli elementi per individuare il dispositivo fisico e realizzare il servizio richiesto. In questo senso, l'accoppiamento tra M e S è basso.

Supponiamo, al contrario, che S richieda, invece del codice, il riferimento fisico (ad esempio l'indirizzo) del dispositivo di uscita. In questo caso i moduli che richiedono un servizio ad S devono fornire le seguenti informazioni:

1. il messaggio da produrre,
2. il riferimento fisico del dispositivo su cui il messaggio deve essere prodotto.

Sotto queste ipotesi, i moduli clienti sono perciò responsabili della conversione tra il riferimento logico ed il dispositivo fisico: l'accoppiamento tra essi ed S è maggiore rispetto al caso precedente. Lo svantaggio più evidente è che, nel caso in cui si cambi il riferimento fisico di un dispositivo, sono necessari gli opportuni cambiamenti nei moduli clienti, pregiudicando la riusabilità e l'estendibilità. Un altro svantaggio è che ogni cliente che utilizza il modulo per la stampa è obbligato ad eseguire delle operazioni preliminari di conversione da riferimento logico a riferimento fisico prima di potere usare i servizi di stampa. Queste operazioni di conversione hanno bassa coesione con le altre caratteristiche dei moduli cliente.

Passiamo ad analizzare la modularizzazione per oggetto. In questo caso, il modulo viene dedicato alla gestione di un particolare oggetto che deve essere utilizzato dai moduli clienti. Nella modularizzazione per oggetto la coesione aumenta nella misura in cui siamo in grado di concentrare nel modulo che gestisce l'oggetto O tutte e sole le funzionalità necessarie per la rappresentazione e la gestione di O, mentre l'accoppiamento diminuisce nella misura in cui riusciamo a nascondere nel modulo che gestisce O tutti gli aspetti irrilevanti per i moduli clienti, e riusciamo invece a rendere opportunamente visibili le modalità di utilizzo di O.

Ad esempio, supponiamo che diversi moduli di un programma debbano essere in grado di effettuare operazioni di scrittura su e lettura da un video. Una possibilità di impostazione del progetto è quella in cui ciascun modulo è responsabile delle operazioni effettuate sul video. Ciò implica che la struttura del video deve essere nota a tutti i moduli, e che le modalità con cui il video stesso viene gestito sono condivise da tutti i moduli. Osserviamo subito che questa condivisione crea un potenziale accoppiamento tra i moduli. Ragionando con l'ottica della modularizzazione per oggetto, imposteremmo il progetto in modo diverso, dedicando un modulo V alla gestione del video, in particolare:

- definendo in V tutti i dati necessari per rappresentare le proprietà del video (ad esempio la dimensione, il livello di grigio dei punti ecc.), e nascondendo tali dati ai moduli clienti
- offrendo ai moduli clienti tutte e sole le operazioni necessarie per eseguire i servizi richiesti.

È facile convincersi che questa impostazione aumenta la coesione, sia perché nel modulo V abbiamo concentrato tutti gli aspetti relativi all'entità "video", sia perché abbiamo scorporato dai moduli clienti delle funzionalità che avevano bassa coesione con le altre funzionalità specifiche dei moduli stessi. L'accoppiamento tra i moduli cliente ed il modulo V è basso in quanto gli aspetti relativi alla rappresentazione del video sono ora nascosti ai moduli clienti, che invece comunicano con V mediante tutte e sole le necessarie modalità che servono per eseguire i servizi necessari.

Si noti che nella modularizzazione per oggetto è implicitamente presente il concetto di modularizzazione funzionale: è infatti ovvio che il modulo che gestisce un oggetto O raccoglie necessariamente tutte le funzioni di gestione di O stesso. In questo senso, ogni volta che operiamo una modularizzazione per oggetto, operiamo anche implicitamente una modularizzazione funzionale (si noti che non è invece vero il contrario).

La modularizzazione per tipo astratto rappresenta in qualche modo una estremizzazione della modularizzazione per oggetto, e consiste nel realizzare moduli che, invece di occuparsi della gestione di un singolo oggetto, offrono all'esterno la rappresentazione e la gestione di una classe di oggetti, cioè di una collezione di oggetti aventi caratteristiche comuni. Questo tipo di modularizzazione costituisce la nozione centrale delle metodologie di progetto e dei linguaggi orientati agli oggetti. Nei linguaggi orientati agli oggetti è esplicitamente presente un costrutto per gestire classi di oggetti e per stabilire diversi tipi di relazioni tra classi. Nei linguaggi imperativi con moduli, il concetto di classe non è esplicitamente presente, ma è disponibile in genere la possibilità da parte di un modulo di nascondere la rappresentazione di un tipo[2]. Questo è il meccanismo minimo che consente di realizzare esplicitamente la modularizzazione per tipo astratto, e di strutturare il modulo MT che gestisce il tipo astratto T secondo il seguente principio: MT offre all'esterno il nome del tipo T e nasconde invece la sua struttura; inoltre MT offre all'esterno tutte e sole le operazioni definite sul tipo astratto T, nascondendo le modalità di realizzazione delle operazioni stesse. Se consideriamo l'insieme dei moduli che utilizzano MT in un programma, è facile rendersi conto che la coesione sarà tanto maggiore quanto più riusciremo ad isolare le caratteristiche di T in MT e scorporarle dai moduli clienti, mentre l'accoppiamento sarà tanto minore quanto più riusciremo a fare interagire MT ed i moduli clienti sulla base delle proprietà astratte di T e non sulla base delle modalità con cui queste proprietà sono realizzate in MT.

Nella modularizzazione per tipo astratto è implicitamente presente sia la nozione di modularizzazione per oggetto, perché un tipo astratto è l'astrazione di un insieme omogeneo di oggetti, sia la nozione di modularizzazione funzionale, perché il modulo che gestisce il tipo T raccoglie tutte le funzioni rilevanti sui valori di tipo T.

9.4 La modularizzazione nella fase di concettualizzazione

La tecnica della modularizzazione viene utilizzata nella fase di concettualizzazione principalmente per descrivere i moduli che si ritengono importanti per lo sviluppo del programma e le relazioni concettuali tra essi.

A partire dal documento di analisi vengono individuati i vari moduli che hanno rilevanza concettuale, e vengono eseguite le seguenti scelte:

- ogni modulo viene identificato mediante un nome e viene caratterizzato secondo il tipo: funzionale, per oggetto o per tipo astratto;
- vengono stabilite le relazioni tra i vari moduli,
- viene poi fornita la specifica dettagliata di ogni modulo.

Come abbiamo già detto nel capitolo 2, il prodotto finale viene chiamato lo *schema concettuale di progetto*, o semplicemente *schema concettuale*. Le caratteristiche essenziali dello schema concettuale vengono spesso raffigurate mediante un diagramma in cui i moduli sono rappresentati da rettangoli, e le relazioni tra moduli da archi che li congiungono.

Le relazioni che si stabiliscono tra i moduli sono diverse a seconda dei tipi di moduli coinvolti.

[2]Un esempio è il "tipo opaco" in MODULA-2 (si veda [56]).

Terminiamo il paragrafo con un esempio di costruzione dello schema concettuale.

Esempio 9.4.1 Consideriamo una applicazione relativa ai dati di un insieme di persone, con i relativi appartamenti di proprietà. Supponiamo che la fase di analisi dei requisiti abbia messo in evidenza che sono di interesse per l'applicazione i seguenti aspetti:

- Le persone, con il loro nome e cognome, il codice fiscale, la data di nascita, la città di nascita, e gli appartamenti di cui sono proprietari. Nel caso in cui le persone siano impiegati, ha interesse anche lo stipendio medio annuo.
- Le città di nascita delle persone, con il nome e la regione di appartenenza.
- Gli appartamenti, con la superficie, il numero di vani, l'indirizzo, il valore, e la città in cui si trovano.
- Il calcolo del valore degli appartamenti di una persona.

Mostriamo in figura 9.4 la rappresentazione diagrammatica dello schema concettuale, con i vari moduli, il loro tipo, e le relazioni tra i moduli.

□

L'esempio appena presentato mette in evidenza che nel diagramma dello schema concettuale sono di interesse principale le relazioni che si individuano tra i moduli, mentre i moduli stessi sono descritti in modo piuttosto scarno. Ad esempio, nel diagramma di figura 9.4 non c'è traccia di alcune proprietà dei moduli per tipo astratto (come nome, cognome, codice fiscale delle persone). Vedremo nel paragrafo 9.6 che, associate al diagramma, vengono riportate altre informazioni che hanno lo scopo di specificare in maggiore dettaglio le caratteristiche di ciascun modulo. Il tipo di specifica che si aggiunge ad un modulo dipende dal tipo del modulo stesso, ovvero dal fatto che sia un modulo funzionale, per oggetto o per tipo astratto.

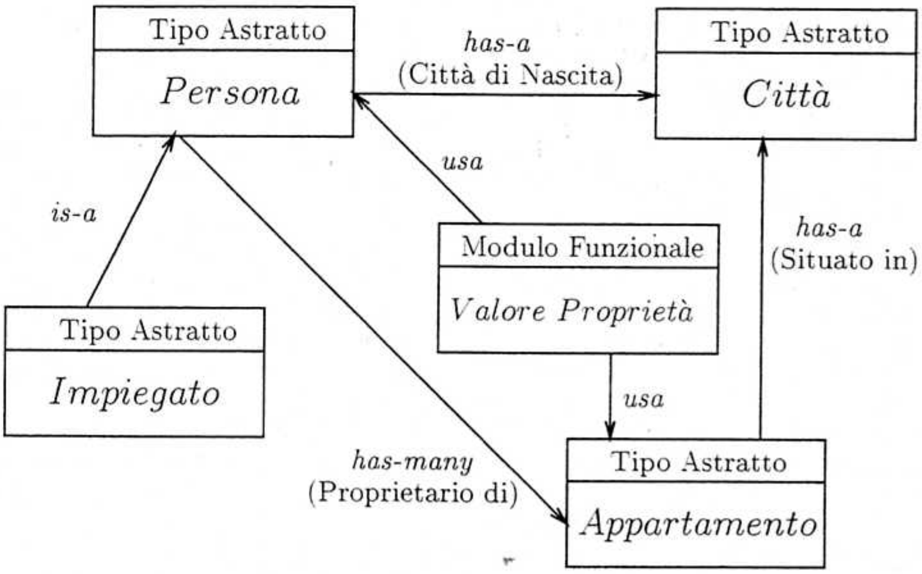

Figura 9.4 Lo schema concettuale dell'applicazione dell'esempio 9.4.1

9.5 La modularizzazione nella fase di realizzazione

La tecnica della modularizzazione viene utilizzata nella fase di realizzazione per partizionare il programma in diverse unità. La nozione di unità di programma varia nei diversi linguaggi di programmazione. Ad esempio, in PASCAL standard modulo significa semplicemente sottoprogramma, poiché l'unico modo per partizionare un programma è di definire ed utilizzare funzioni e procedure.

Altri linguaggi di programmazione sono molto più sofisticati, e rendono disponibili opportuni costrutti che consentono di delimitare sintatticamente un segmento di codice, contenente un insieme di dichiarazioni e di definizioni (procedure, funzioni, variabili, costanti, tipi ecc.). In tali linguaggi un programma si può quindi considerare come un insieme di moduli interagenti, dei quali uno, il modulo principale, specifica il flusso di esecuzione. I meccanismi con cui si definisce un modulo e si specifica come un modulo può utilizzare i servizi offerti dagli altri moduli sono governati da due insiemi di primitive messe a disposizione dal linguaggio di programmazione:

- le primitive per specificare i moduli,
- le primitive di definizione della interazione tra moduli.

Il MODULA-2 è un linguaggio particolarmente accurato nell'idea del partizionamento di un programma in moduli. Altri linguaggi offrono meccanismi meno espliciti e sofisticati. In C++ la modularizzazione si può ottenere con i seguenti meccanismi.

- Partizionando il programma in funzioni. Ogni funzione è infatti un modulo, in cui le variabili locali realizzano l'information hiding, e i parametri l'interfacciamento esplicito.
- Definendo ed utilizzando le classi. Ogni classe è infatti un modulo, con i meccanismi di information hiding e interfacciamento esplicito visti nel capitolo 4.
- Partizionando il programma in più file, e sfruttando la possibilità di includere un file in un altro.
- Partizionando il programma in più file, e sfruttando la possibilità di compilare i file separatamente e collegarli con il *linker* per ottenere un programma eseguibile (si veda il paragrafo 3.7). In questo caso si utilizzano specifici meccanismi del linguaggio, quali `static` e `extern`, per realizzare l'information hiding e l'interfacciamento esplicito.