4. Modellazione di Sistemi

Cenni sulle diverse tecniche di specifica di sistemi software

Andrea Polini

Ingegneria del Software
Corso di Laurea in Informatica

Sommario

1. Specifiche - generalità
2. Modelli Operazionali
   - Data Flow Diagram (DFD)
   - Finite State Machines (FSM)
   - Petri Nets (PN)
3. Modelli Descrittivi
   - Diagrammi Entità-Relazione
   - Specifiche logiche
   - Specifiche algebriche (Cenni)

Materiale di studio

Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli
Fondamenti di Ingegneria del Software, 2ª Ed. Italiana
Prentice Hall, 2004
(Capitolo 5)

Sommario

Generalità

Una specifica termine piuttosto generale utilizzato in differenti fasi della produzione del software.

In generale una specifica fornisce una precisa definizione delle caratteristiche del “manufatto” che deve essere realizzato nelle fasi successive, fino a giungere all’implementazione.

Specifica dei requisiti contiene dunque una definizione delle caratteristiche e delle funzionalità attese dal cliente.

Specifiche

Stili: Operazionali vs. Descrittive

Le specifiche si distinguono in due grandi categorie:

- operazionali: descrivono un sistema attraverso il comportamento necessario a raggiungere gli obiettivi
- descrittive: descrivono proprietà del sistema che devono essere vere

Un esempio dal mondo matematico.

Un esempio nel mondo “informatico”

Come al solito le distinzioni non sono sempre nette!

Specifiche

Stili: Operazionali vs. Descrittive

Le specifiche si distinguono in due grandi categorie:

- operazionali: descrivono un sistema attraverso il comportamento necessario a raggiungere gli obiettivi
- descrittive: descrivono proprietà del sistema che devono essere vere

Un esempio dal mondo matematico.

Un esempio nel mondo “informatico”

Come al solito le distinzioni non sono sempre nette!

Specifiche

Stili: Operazionali vs. Descrittive

Le specifiche si distinguono in due grandi categorie:

- operazionali: descrivono un sistema attraverso il comportamento necessario a raggiungere gli obiettivi
- descrittive: descrivono proprietà del sistema che devono essere vere

Un esempio dal mondo matematico.

Un esempio nel mondo “informatico”

Come al solito le distinzioni non sono sempre nette!

Specifiche

Stili: Operazionali vs. Descrittive

Le specifiche si distinguono in due grandi categorie:

- operazionali: descrivono un sistema attraverso il comportamento necessario a raggiungere gli obiettivi
- descrittive: descrivono proprietà del sistema che devono essere vere

Un esempio dal mondo matematico.

Un esempio nel mondo “informatico”

Come al solito le distinzioni non sono sempre nette!

Sommario

Sommario

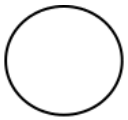
Caratteristiche

DFD sono una notazione molto utilizzata per descrivere le funzioni di un sistema informativo e su come i dati fluiscono all'interno del sistema. Il sistema è in effetti visto come un insieme di funzioni che manipolano dati.

Come possono essere manipolati i dati:

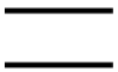
- I dati possono essere immagazzinati in repository
- I dati possono fluire all'interno del sistema
- I dati possono entrare o uscire dal sistema

Sintassi

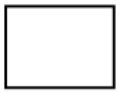


Simbolo utilizzato per rappresentare le funzioni

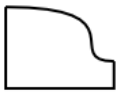
Simbolo utilizzato per rappresentare il flusso del dato



Simbolo utilizzato per rappresentare un repository



Simbolo utilizzato per rappresentare un dispositivo di input



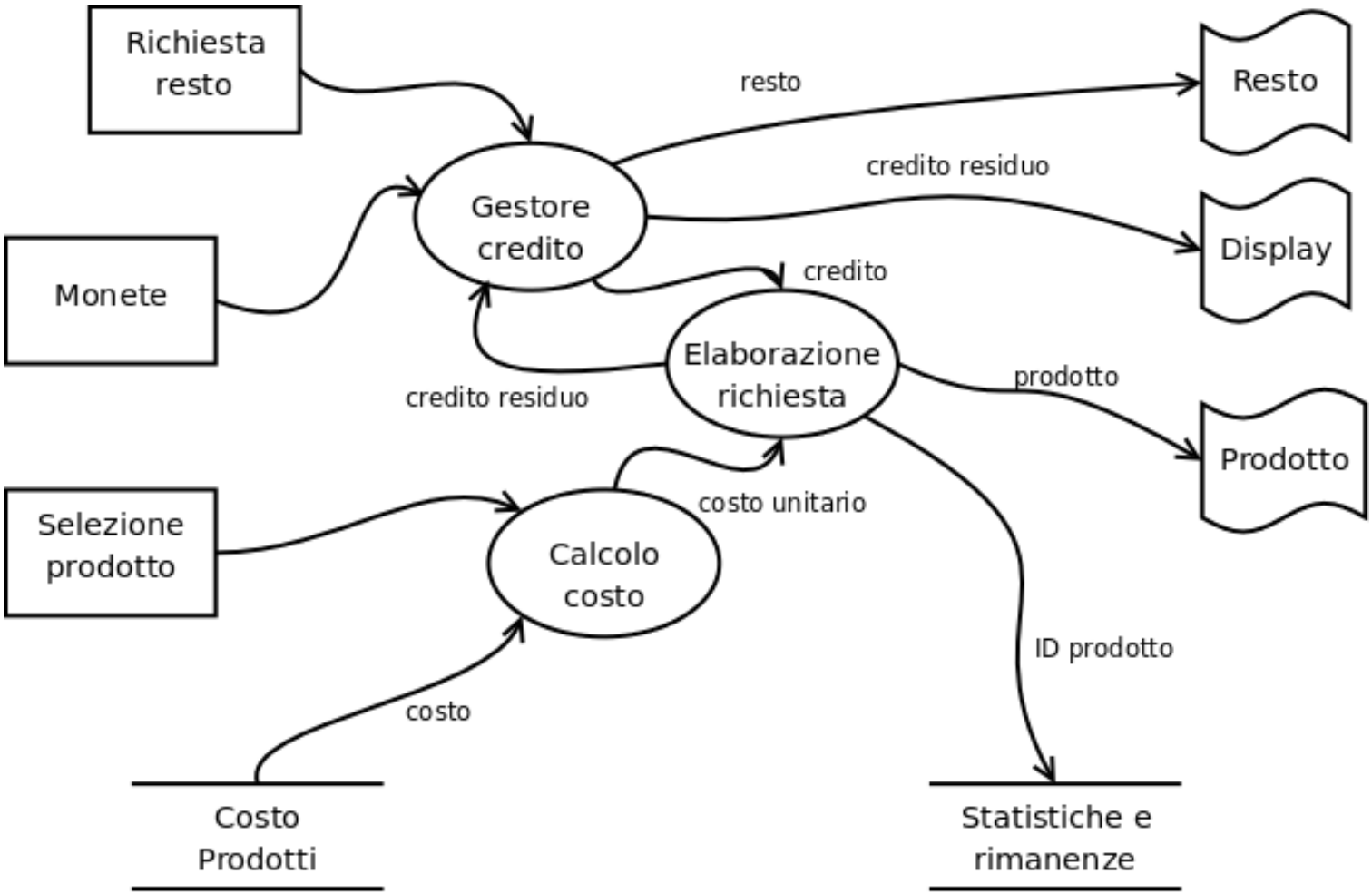
Simbolo utilizzato per rappresentare un dispositivo di output

Regole per combinare i simboli...

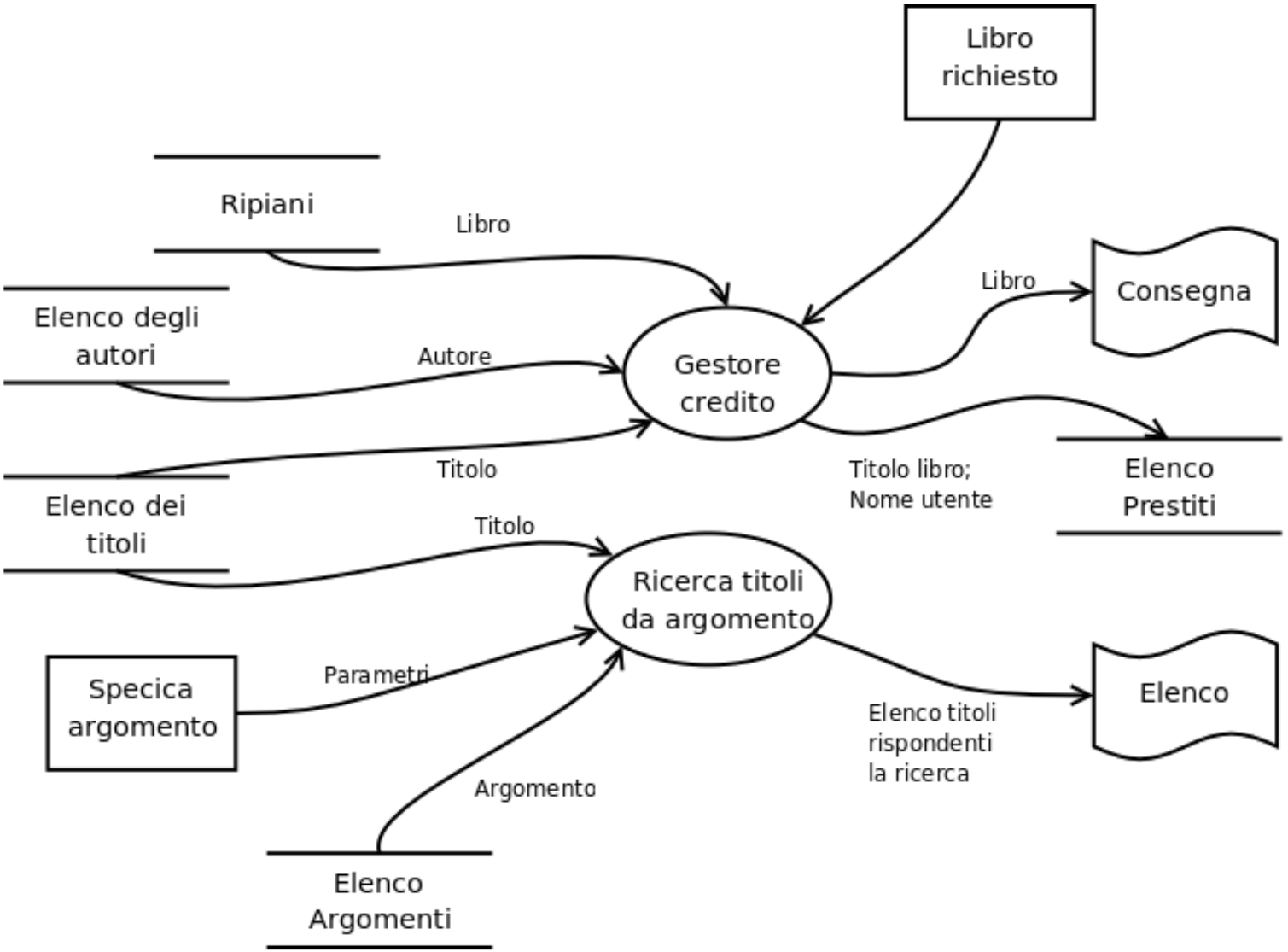
Qualche esempio

- dal mondo matematico $(a + b) * (c + a * d)$
- Il distributore del caffè
- La biblioteca

Distributore del caffè



Biblioteca


Libro richiesto
Ripiani
Libro
Elenco degli autori
Autore
Gestore credito
Libro
Consegna
Titolo
Titolo libro; Nome utente
Elenco Prestiti
Elenco dei titoli
Titolo
Ricerca titoli da argomento
Specica argomento
Parametri
Elenco titoli rispondenti la ricerca
Elenco
Argomento
Elenco Argomenti

Carenze dei DFD

Semantica evocata dai nomi ma non c'è definizione precisa

Carenza nella definizione del controllo

Semantica ed arrichimento delle informazioni in un DFD

Abbiamo visto molti esempi di ambiguità

Formali? Semiformali? Informali?

Sommario

Caratteristiche

Una macchina a stati finiti è formalmente definita da una tupla $<\mathcal{Q}, \mathcal{I}, \delta, q_0>$ dove:

- $\mathcal{Q}$ è un insieme di stati
- $\mathcal{I}$ è un insieme finito di input
- δ è la funzione di transizione – $\mathcal{Q} \times \mathcal{I} \rightarrow \mathcal{Q}$
  δ può anche essere definita soltanto parzialmente
- $q_0 \in \mathcal{Q}$ è lo stato iniziale

Caratteristiche

Formali? semiformali? informali?

- FSM sono molto usate per verificare formalmente proprietà dei sistemi
- Una FSM si trova sempre in un solo stato e permette di specificare soltanto uno stato successivo per una stessa azione (caso deterministico). Allo stesso tempo il sistema è un sistema sincrono che non ha attività concorrenti.
- Strumento piuttosto limitato. Memoria limitata dal numero degli stati.
  - Esplosione degli stati (e.g. somma di due interi) ed estensione del formalismo di base.
- È necessario focalizzarsi sul aspetti importanti del sistema del sistema ed eventualmente arricchire il formalismo (formalmente o informalmente)

Molte sono le possibili estensioni proposte *I/O FSM, EFSM, . . .*
Transizioni con guardia ed azioni.

Sommario

Reti di Petri

Le reti di Petri sono un formalismo ideato negli anni 60 per modellare sistemi concorrenti, asincroni, distribuiti, paralleli, non deterministici, e/o stocastici.
Formalmente sono definite da una tupla $<\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{W}, \mathcal{M}_0>$:

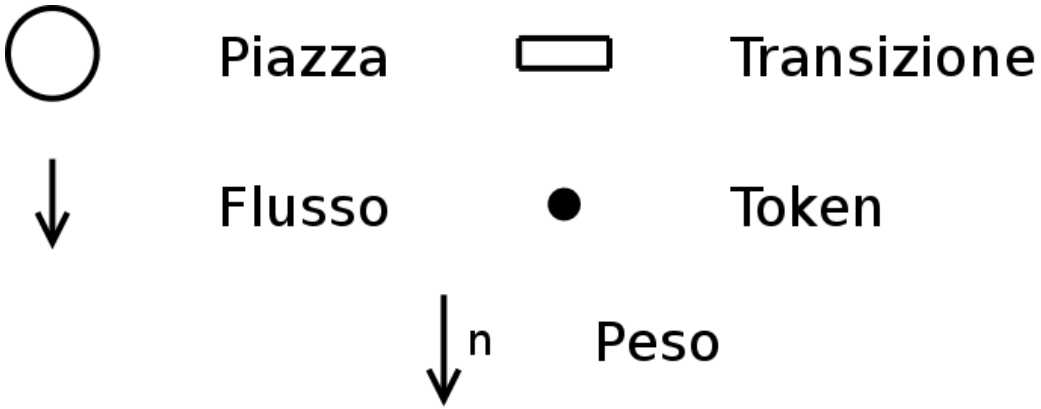
- $\mathcal{P}$ è un insieme finito di piazze
- $\mathcal{T}$ è un insieme finito di transizioni
- $\mathcal{F} \subseteq \{\mathcal{P} \times \mathcal{T}\} \cup \{\mathcal{T} \times \mathcal{P}\}$ è detta relazione di flusso della rete di Petri
- $\mathcal{W} : \mathcal{F} \to \mathbb{N}^+$ è la funzione peso che associa valore non nullo agli elementi di $\mathcal{F}$
- $\mathcal{M}_0 : \mathcal{P} \to \mathbb{N}$ è la marcatura iniziale ed indica lo stato iniziale della rete di Petri

Deve poi valere che: $\mathcal{P} \cup \mathcal{T} \neq \oslash$ e $\mathcal{P} \cap \mathcal{T} = \oslash$

In ogni momento lo stato di una rete di petri è indicato dalla funzione di marcatura che associa ad ogni piazza un numero naturale indicante il numero di "token" (gettoni) presenti nella piazza:
$\mathcal{M} : \mathcal{P} \to \mathbb{N}$

Reti di Petri

notazione grafica



Evoluzione di una rete di Petri:

- Data una transizione chiamiamo piazza di input (o di output), una piazza collegata ad un flusso entrante (uscente) dalla transizione stessa.
- Una transizione è abilitata, e può dunque “scattare”, se e solo se **TUTTE** le piazze collegate ad un qualsiasi flusso di input alla transizione, contengono un numero di token maggiore o uguale al peso dell’elemento flusso che collega ognuna di loro alla transizione.
- Quando una transizione “scatta” ogni piazza collegata ad un flusso uscente dalla transizione riceverà un numero di token pari al peso del corrispondente flusso.
- Le transizioni con nessun flusso entrante sono sempre abilitate.

Petri Net

Tipicamente le piazze possono rappresentare una risorsa e l'accesso alla risorsa può essere modellato dalla presenza di Token. Allo stesso modo piazze e transizioni possono essere utilizzate per rappresentare stati di un processo e l'evoluzione dello stesso

Politiche di risoluzione dei conflitti non sono intrinseche nel formalismo ma devono invece essere implementate. Ad esempio il formalismo non è *fair*. Esempio....

Una rete di Petri si dice in deadlock se non ci sono transizioni abilitate. Nel caso in cui la rete di Petri si trovi in deadlock la sua marcatura rappresenterà lo stato finale delle rete. Esempio...

Esercizi 1/4

Realizzazione e discussione di un modello di Rete di Petri per i seguenti problemi:

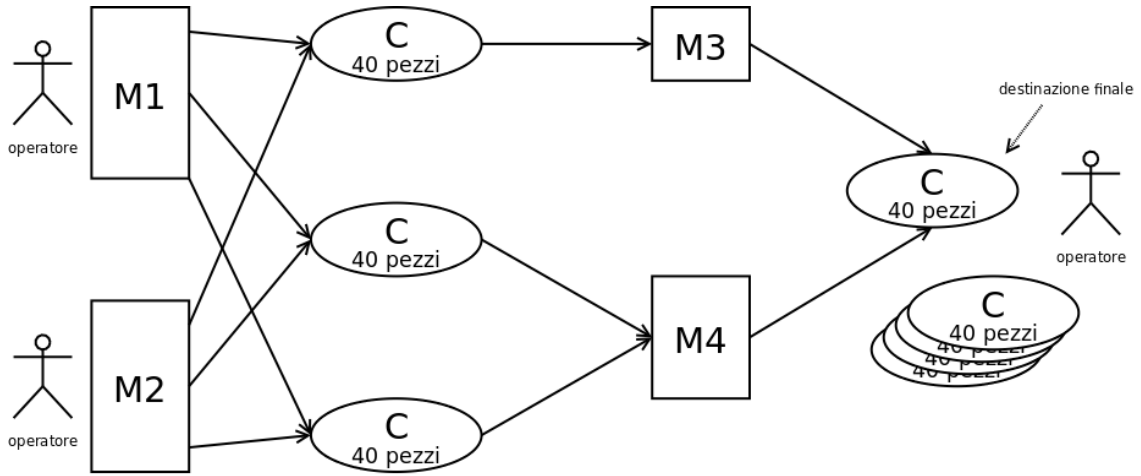
- Scambio di messaggi tra due processi (lettore/scrittore) con un buffer di due posizioni
- Il problema dei processi Lettori/Scrittori
- Gestione accesso incrocio stradale
- Filosofi a cena

Esercizi 2/4

Sistema di Interconnessione - 24 Settembre 2012
L'azienda di produzione di sistemi di interconnessione "Nettoyer" ci richiede di realizzare un semplice meccanismo di instradamendo guidato per nodi con quattro flussi entranti e tre connessioni uscenti. Il sistema si deve comportare nel seguente modo. Numerati da 1 a 4 i flussi entranti il sistema accetterà in modo alternato messaggi provenienti da flussi pari e dispari (inizialmente pari). I flussi entranti vengono instradati sui flussi uscenti in modo progressivo, vale a dire il primo messaggio sul flusso uscente 1, il secondo sul 2 e così via fino a indirizzare il flusso 4 su 1 e dunque ripartire. Il sistema non deve accettare messaggi in ingresso finché il messaggio in uscita non sia stato instradato. Per ogni connessione in entrata deve essere memorizzato il numero di messaggi instradati.

Esercizi 3/4

Catena di montaggio - 29 Settembre 2008
Si consideri il progetto di un sistema di controllo di una catena di montaggio in cui diversi macchinari disposti in serie e parallelo svolgono diverse fasi della lavorazione. In particolare ci si riferisca alla rappresentazione riportata nella figura sottostante. Sono istallate due macchine di produzione M1 ed M2 ognuna controllata da un operatore e perfettamente identiche.
Tali macchine passano da uno stato in cui lavorano il prodotto sotto il diretto controllo dell'operatore, ad uno stato in cui tentano di rilasciare il pezzo all'interno di un cassone (da scegliere tra tre possibili) di dimensioni standard e che possono contenere 40 pezzi ognuno. Ovviamente il rilascio può avvenire soltanto se c'è ancora spazio in uno dei tre cassoni collegati.
Il prelievo dei manufatti dai vari cassoni avviene da parte di due macchina M3 ed M4 aventi velocità di produzione differenti. Tali macchine prelevano il pezzo da lavorare da uno dei possibili cassoni collegati (uno soltanto nel caso della macchina M3 e 2 nel caso della M4) e dopo una fase di ulteriore lavorazione del manufatto lo inseriscono all'interno di un cassone di destinazione finale. Anche in questo caso il rilascio nel cassone può avvenire soltanto se c'è ancora dello spazio disponibile. Al fine di non causare il blocco della catena è compito dell'operatore sostituire il cassone di destinazione finale, dopo essere stato allertato dall'accensione di una luce di controllo.



Esercizi 4/4

Ufficio Spedizioni - 24 Febbraio 2010
Il sistema software da sviluppare intende fornire uno strumento di gestione e controllo delle attività di un ufficio spedizioni. In particolare i clienti di un tale ufficio hanno la possibilità di scegliere all'entrata il tipo di servizio di cui hanno bisogno ed il sistema comunicherà all'utente il numero d'ordine nella coda assegnatagli. L'ufficio fornisce tre differenti tipologie di servizio di spedizione:

- Il primo tipo di spedizione prevede delle pratiche particolarmente lunghe ed il sistema non accetta che siano inseriti in coda più di 10 utenti contemporaneamente. Dunque se un cliente sceglie tale servizio quando i posti in coda sono già esauriti il sistema non permetterà l'accodamento.
- Il secondo tipo di spedizione è di tipo speciale e prevede che vengano restituite all'utente delle ricevute di spedizione e che il documento presentato per la spedizione sia riposto in apposite cartelle che saranno poi gestite con priorità
- Il terzo tipo di servizio non presenta particolari requisiti e non prevede la gestione di documentazione aggiuntiva. L'addetto accetterà il documento/pacco da spedire e non dovrà mettere in atto particolari attività di gestione della spedizione.

L'ufficio da considerare ha tre sportelli che seguono procedure di servizio differenti. In particolare:

- Il primo sportello è preposto in prima istanza all'accettazione del solo primo tipo di spedizione. Soltanto nel caso non vi sia nella coda corrispondente alcun cliente richiedente spedizioni l'addetto potrà scegliere di servire un utente che necessita di una spedizione del terzo tipo.
- Il secondo sportello è invece preposto al servizio di clienti richiedenti spedizioni del secondo e del terzo tipo.
- il terzo sportello è infine preposto solo ed esclusivamente a servire clienti richiedenti spedizioni del secondo tipo.

Si noti che con riferimento alle spedizioni del secondo tipo l'addetto allo sportello dovrà produrre la stampa della ricevuta all'utente. Tale stampa può avvenire tramite una stampante condivisa tra gli sportelli 2 e 3 che non ha buffer. Successivamente il documento da spedire dovrà essere posto dall'addetto in uno specifico contenitore che può contenere al più 10 documenti alla volta. Ognuno degli sportelli 2 e 3 ha a disposizione un contenitore. Tali contenitori vengono poi gestiti e sostituiti da un addetto dell'ufficio il quale provvede a riporli in un'apposita area di smistamento dell'ufficio che possiamo considerare di dimensione non limitata. È altresì necessario che il sistema fornisca informazioni relative sia al numero di spedizioni del primo tipo che sono state accettate, sia al numero di contenitori di spedizioni (non singole spedizioni) del secondo tipo che siano stati riempiti e posti nell'apposita area per lo smistamento. Non è invece necessario fornire infomazioni relative a spedizioni del terzo tipo.

Problemi e possibili estensioni al modello

Formalismo che si focalizza sul controllo. Comunque presenta alcune lacune in particolare per la gestione dei dati.

- In particolare non è possibile modificare il flusso in dipendenza dei dati.
- Non è possibile selezionare una transizione tra più transizioni attive.
- Possibilità di specificare tempo e deadline.

Le reti di Petri sono state estese per poter modellare sistemi più complessi:

- Token con associati valori. Token tipati e dunque gestione del contenuto. (Coloured Petri Nets - CPN)
- Politiche di priorità. pri:T $\rightarrow \mathbb{N}$.
- Reti di Petri temporizzate. Alle transizioni sono associati valori di tempo minimo e massimo entro i quali la transizione, se abilitata, potrà scattare.
- Reti temporizzate con associate funzioni di distribuzione (Stochastic Petri Nets - SPN)

Sommario

Modelli Descrittivi

Questo tipo di modelli forniscono una formalizzazione del sistema in termini delle proprietà che questo soddisfa. Non viene descritto direttamente il suo comportamento.

- Diagrammi Entità Relazione (ER)
- Specifiche Logiche
- Specifiche Algebriche

Sommario

Diagrammi Entità-Relazione

Descrizione concettuale della struttura dei dati e delle loro relazioni.
Rispetto ai DFD:

- Diversamente DFD si focalizzavano sul flusso del dato ma non specificavano la loro struttura
- Possono essere utilizzati insieme per modellare diversi aspetti

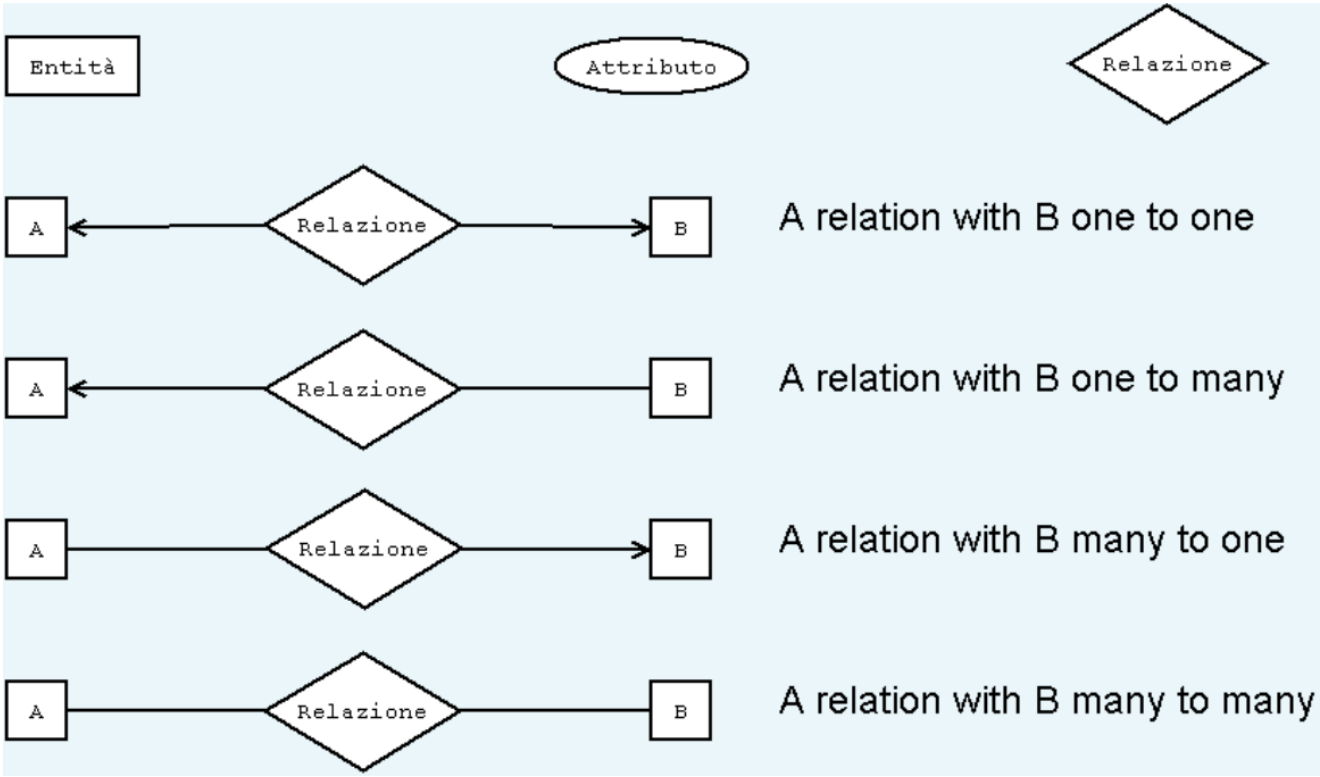
Vengono modellati tre concetti fondamentali che permettono di rappresentare la struttura logica dei dati:

- Entità
- Attributi
- Relazioni

Diagrammi Entità-Relazione

notazione grafica

ER non sono stati standardizzati, dunque molte varianti



In alcune varianti è permesso definire relazioni parziali

Limitazioni

Potere espressivo è piuttosto limitato. Concetti complessi di relazione non possono essere rappresentati (anche se alcune varianti permettono di rappresentare relazioni di ereditarietà tramite relazioni “is-a”)

Alcune varianti permettono di definire relazioni n-arie che comunque risultano concettualmente e tecnicamente complesse da interpretare ed implementare

Non è possibile specificare che un’entità esiste solo se il numero di istanze in una relazione sarà maggiore di un certo numero.

Come al solito poi per i problemi più “spinosi” sono proposte estensioni. In generale il diagramma viene complementato con informazioni aggiuntive. Formali e non.

Diagramma delle classi ed ER

Sommario

Specifiche logiche

generalità

Utilizzo di formule FOT (First Order Theory) per descrivere proprietà di un programma

Una proprietà sarà soddisfatta se il corrispondente predicato assumerà valore positivo.

FOT è un espressione che può comprendere:

- variabili (a, b, ...)
- costanti numeriche (1, 3.4, 4.233e10, ...)
- funzioni $f(x, y, \ldots)$, ...
- predicati $p()$, ...
- parentesi ( ), [ ], { }, ...
- connettivi logici ($\wedge$, $\vee$, $\neg$, $\Rightarrow$, ...)
- quantificatori ($\forall$, $\exists$)

Specifiche logiche

- Variabili libere e legate e dipendenza del valore di verità
- Valore di verità dipende dal dominio scelto per le variabili della formula

Dato un programma P con input $<i_1, i_2, \ldots, i_n>$ ed output $<o_1, o_2, \ldots, o_m>$ una specifica logica per P è definita come:

{**Pre**($i_1, i_2, \ldots, i_n$)}
P
{**Post**($o_1, o_2, \ldots, o_m, i_1, i_2, \ldots, i_n$)}

Esercizi:

1. programma che calcola somma dei primi n numeri
2. programma che genera permutazione di un vettore di ingresso
3. programma che verifica che il primo elemento di un vettore sia duplicato all'interno dello stesso
4. ...

Specifiche di parti

Specifiche logiche possono essere utilizzate anche per specificare parti di programma utilizzando le variabili stesse del programma all'interno della specifica logica

Il caso di una classe nei linguaggi OO e le invarianti.

- Stato di un oggetto
- Esecuzione dei metodi e pre e post condizioni
  - $\{\text{INV} \wedge \text{Pre}(i_1, i_2, \ldots, i_n)\}$
    $\text{m}(i_1, i_2, \ldots, i_n)$
    $\{\text{INV} \wedge \text{Post}(o_1, o_2, \ldots, o_m, i_1, i_2, \ldots, i_n)\}$

Esempio: vettore estendibile di elementi ordinati

Specifiche logiche

considerazioni

Anche semplici programmi generano specifiche piuttosto complesse.

Definizione e riuso di formule permettono di semplificare la specifica

Spesso associate ad altri formalismi per aumentarne la potenza espressiva.

Specifiche logiche e verifica

Utilizzando FOT viene specificato che per qualsiasi implementazione di un programma o di un frammento di programma i predicati devono essere valutati a vero

Espressioni logiche possono essere utilizzate nella fase di analisi applicando deduzione logica. La proprietà espressa in forma FOT deve poter essere derivata dalla specifica. Questo fornisce una prova!

Questo non era possibile per le specifiche operazionali con le quali è possibile scoprire comportamento (simulazioni) ma non fornire prove. Ma . . . theorem proving è indecidibile in FOT

Specifiche logiche e verifica

Utilizzando FOT viene specificato che per qualsiasi implementazione di un programma o di un frammento di programma i predicati devono essere valutati a vero

Espressioni logiche possono essere utilizzate nella fase di analisi applicando deduzione logica. La proprietà espressa in forma FOT deve poter essere derivata dalla specifica. Questo fornisce una prova!

Questo non era possibile per le specifiche operazionali con le quali è possibile scoprire comportamento (simulazioni) ma non fornire prove. Ma . . . theorem proving è indecidibile in FOT

Esercizi

23 Giugno 2008

Si supponga di voler specificare tramite un formalismo logico una funzione che data una matrice di interi di dimensioni non nulle ed un vettore di lunghezza non nulla ed ordinato in maniera crescente sia capace di restituire il vettore prodotto. Per la definizione della funzione si consideri la seguente sintassi:

```
int[] MatrixProduct(int M[][], int V[])
```

Nella definzione della specifica è possibile utilizzare il meccanismo di accesso agli elementi delle strutture dati attraverso la notazione `M[i][j]` che indica l'elemento della matrice alla riga `i` ed alla colonna `j` (si operi similmente per accedere agli elementi del vettore). Nella definizione dei predicati logici è poi possibile utilizzare le funzioni di sommatoria e produttoria. Infine possono essere utilizzare le funzioni:
`int length(int V[])` – che restituisce la lunghezza di un vettore
`int dimRow(int M[][]),int dimColumn(int M[][])` – che restituiscono rispettivamente il numero di righe e di colonne di una matrice.

Esercizi

12 Dicembre 2008

Si utilizzi il formalismo logico per definire il comportamento delle seguenti funzioni:

a) Si fornisca una specifica per la seguente funzione che data in ingresso una matrice `MI` di interi non nulli di dimensioni $n \times m$ e per cui somma di tutti gli elementi sia 0 restituisce la corrispondente matrice trasposta:

```
int[][] TraspostaNull(int[][] MI)
```

b) Si fornisca una specifica per la seguente funzione che presa in ingresso una matrice quadrata `MI` di dimensione n restituisce in output una struttura dati complessa di tipo `Result` così definita:
`public class Record {int[][] MR; int[] V;}`
La matrice MR conterrà la trasposta della matrice di input mentre il vettore V conterrà le posizioni di riga e colonna in posti adiacenti (con riferimento al triangolo superiore della matrice) per gli elementi invarianti nella trasposizione che non si trovino sulla diagonale.

```
Result TraspostaInv(int[][] MI)
```

Ad esempio data in ingresso la seguente matrice:

12	20	9	10
20	9	10	11
12	12	13	11
2	11	5	1

la funzione restituirà una struttura dati di tipo `Result` contenente la seguente matrice e vettore:

12	20	12	2
20	9	12	11
9	10	13	5
10	11	11	1

1	2	2	4

Nella definizione della specifica si possono utilizzare le tipiche funzioni definite sulle strutture dati matrice (`dimRow`, `dimColumn`) e vettore (`length`) che restituiscono le corrispondenti dimensioni.

Esercizi

09 Settembre 2013

Utilizzando un formalismo per la logica del primo ordine si fornisca la specifica delle pre e post condizioni della seguente procedura le cui caratteristiche vengono descritte nel seguito:

```
int[] EULERO(int first, int second)
```

La funzione prende in ingresso due interi positivi e restituisce in uscita un vettore contenente gli elementi intermedi ottenuti applicando l'algoritmo di Eulero per il calcolo del MCD. In particolare il vettore contiene in celle adiacenti le coppie successive ottenute dal metodo di Eulero. Le ultime due celle del vettore dovranno dunque contenere il valore del MCD tra il `first` e `second`. La figura seguente riporta un esempio corretto e scorretto di possibili output.

`[30,24,6,24,6,18,6,12,6,6] = EULERO(30,24)` – OK
`[15,18,15,3,12,3,9,3,6,3,3,3] = EULERO(15,18)` – OK
`[30,26,4,26,4,22,4,18,4,14,4,10,4,6] = EULERO(30,26)` – NOK

Nella specifica è possibile utilizzare le funzioni `max(int a,int b)` e `min(int a, int b)` che hanno l'ovvio comportamento di restituire rispettivamente il massimo o il minimo tra i dati di input.

Esercizi

7 Luglio 2014

Si consideri il seguente metodo della classe `Scheduler` ed utilizzando un formalismo logico se ne definiscano le pre e le post-condizioni:

```
public void schedulePSJF(Job job)
```

Dove la classe `Job` è la struttura dati utilizzata per rappresentare i vari "job" che devono essere eseguiti sul sistema. La struttura dati è così composta:

```
public class Job {
  int id;          // rappresenta l'identificativo del processo
  int duration;    /* rappresenta la durata necessaria al Job per poter completare */
  int priority;    /* rappresenta la priorità che il processo ha nell'esecuzione */    }
```

mentre la classe `Scheduler` è così definita:

```
public class Scheduler {
  Job[] queue;     /* rappresenta la coda dei Job che viene utilizzata per scegliere i Job da
                      mandare in esecuzione */
  Job running;     /* contiene il riferimento al Job che è al momento in esecuzione */
  private void preempt(Job job) { ... }
  public void schedulePSJF(Job job) { ... }  }
```

Esercizi

7 Luglio 2014

Il metodo `schedulePSJF` (da “Prioritized Shortest Job First”) permette di applicare una politica di *sheduling* che ordina i *job* in ordine di priorità e per quelli con la medesima priorità applica la politica *shortest job first*. Dunque il metodo provvede ad inserire il “job” passato come parametro correttamente nella coda. Inoltre nel caso in cui il `job` in esecuzione abbia una priorità inferiore provvede a mettere in esecuzione il “job” passato come parametro attraverso l’esecuzione del metodo `preempt`.

Il vettore `queue` della classe `Scheduler` deve risultare ordinato per priorità e per durata dei job sia prima che dopo l’invocazione del metodo. Dunque nella prima posizione si troverà il *job* con priorità massima e che tra questi abbia durata minima. L’invocazione del metodo causerà l’inserimento del *job* nella coda (la cui lunghezza sarà maggiore di uno) e nel caso in cui la sua priorità sia maggiore del *job* in esecuzione provocherà il rilascio del processore al processo in esecuzione che sarà messo nella coda, nonché l’inserimento del nuovo processo in stato di esecuzione. Al fine di potersi riferire allo stato della coda precedentemente all’invocazione del metodo, nella definizione della post-condizione del metodo stesso, si utilizzi la notazione `@queue`. Si consideri poi che i `job` devono avere una priorità inclusa tra 1 e 10 e che la loro durata deve essere maggiore di 0.

Esercizi

22 Giugno 2009

Si consideri la seguente funzione e se ne definiscano le pre e le post-condizioni in accordo al comportamento descritto nel seguito:

```
int[] LangCheck(Char[][] Automa, int[] Final, String word)
```

La funzione `LangCheck` prende in input:

- una matrice di Char (denominata `Automa`) che rappresenta la funzione di transizione di un automa a stati finiti deterministico. Gli stati dell'automa sono enumerati da 1 a n, dove n è la dimesione della matrice, e lo stato iniziale è rappresentato dall'etichetta 1. Attenzione il carattere `-` è utilizzato a rappresentare la mancanza della transizione. Tale carattere non può dunque far parte dell'alfabeto dell'automa.
- un vettore di interi `Final` la cui lunghezza è pari alla dimensione della matrice `Automa`. L'elemento i-esimo del vettore conterrà il valore 1 se il corrispondente stato è uno stato finale. Altrimenti il valore dell'elemento sarà 0
- una stringa `word` che non potrà contenere il simbolo `-`

Come risultato della computazione la funzione restituirà un vettore di interi che rappresenta la sequenza di stati attraversati nel caso la stringa sia accettata dall'automa. In caso contrario il vettore sarà costituito dalla sequenza di stati attraversati dalla computazione fino al riconoscimento di non accettazione. Il vettore conterrà in tal caso come ultimo elemento il valore 0.
Nella definizione delle pre- e post-condizioni si possono utilizzare le seguenti funzioni:

- `int dimRow(type[][] M)`, `int dimColum(type[][] M)` che data in ingresso una matrice di tipo qualsiasi restituiscono rispettivamente il numero di righe e colonne della matrice
- `int length(type[] V)` che dato in ingresso un vettore di tipo qualsiasi ne restituisce la lunghezza. Inoltre l'elemento i-simo del vettore V può essere ottenuto utilizzando la funzione di proiezione `[ ]`, scrivendo `V[i]`
- `int stringLength(String s)` che restituisce il numero di caratteri in una stringa
- `Char charAt(String s, int i)` che data una stringa ed un intero restituisce lo i-esimo carattere della stringa

Specifica di sistemi reattivi

Specifiche logiche e sistemi reattivi?

- Predicati che descrivono eventi
- Predicati che descrivono stati
- Regole che mettono in relazione stati ed eventi

e.g.: Specifica della macchinetta del caffè

- **Insiemi**: c:coin, p:product, t:time, n:int
- **Funzioni**: prices(products p):int, credit(time t):int
- **Eventi**: insertCoins(c,t), returnChange(t,c), selectProduct(p,t), productReady(p,t) ...
- **Stati**: servingProduct(p,t_1,t_2), creditAvailable(c,t_1,t_2), availableProduct(p,t_1,n)

Esempio: possiamo specificare che la richiesta di una bevanda venga soddisfatta in un tempo definito.

Sommario

Specifiche algebriche (cenni)

generalità

Formalismo descrittivo basato su definizione ed uso di algebre

- algebre omogenee
- algebre eterogenee

Molti sistemi software possono essere specificati “facilmente” utilizzando algebre eterogenee

Specifiche algebriche (cenni)

esempio

Si vuole specificare un sistema che gestisce le stringhe. Con le seguenti operazioni:

- creazione di una stringa vuota
- concatenazione di stringhe
- aggiunta di un carattere ad una stringa
- lunghezza di una stringa
- verifica se una stringa è vuota o meno
- verifica di uguaglianza di stringhe

Insiemi?

Specifiche algebriche (cenni)

esempio

Definizione di algebra (sorts, segnatura, operazioni, vincoli)

La semantica delle operazioni viene data tramite equazioni che definiscono le proprietà fondamentali che devono valere quando le operazioni sono applicate (assiomi)

Descrivere algebra con il linguaggio Larch:

```
algebra [name]
imports [algebra list]
introduces
  sorts [sort list]
  operations
    ...
constraints [operations list] so that
   [name] generated by [ operations list ]
    [constraints list]
```

Specifiche algebriche (cenni)

vincoli

constraints isEmpty, append, new, add, equal, length so that
for all [s,s1,s2:String;c:Char]

- isEmpty(new()) = ?
- isEmpty(add(s,c)) = ?
- length(new()) = ?
- length(add(s,c)) = ?
- append(s,new()) = ?
- append(s1,add(s2,c)) = ?
- equal(new(),new()) = ?
- equal(new(),add(s,c)) = ?
- equal(add(s,c),new()) = ?
- equal(add(s1,c),add(s2,c)) = ?

Specifiche algebriche (cenni)

derivazione di proprietà

Una proprietà è vera se può essere derivata dagli assiomi:

append(new(),add(new(),c)) = add(new(),c)

append(new(),s) = s

Specifiche algebriche (cenni)

proprietà non derivabili dagli assiomi

Capita spesso di poter incontrare proprietà che non possono essere derivate degli assiomi. Si consideri ad esempio la proprietà:

equal(add(s,'a'),add(s,'b')) = false

La specifica è incompleta e possono essere attuati tentativi di rendere la specifica "più" completa aggiungendo altri assiomi.

Specifiche algebriche (cenni)

specifiche modulari

Sistemi complessi possono richiedere di definire molte algebre allo stesso tempo.

Sono previsti operatori di “import” che permettono di includere un’algebra nella definizione di un’altra