6. Modellare la specifica

Come descrivere cosa

Andrea Polini

Ingegneria del Software
Corso di Laurea in Informatica

Sommario

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.

Il problema della specifica

Specifica dei requisiti in linguaggio naturale presenta diverse problematiche e difficoltà di verifica.

- linguaggio tecnico
- ambiguità
- sinonimi
- completezza
- inconsistenza
- comprensibilità
- ...

Il problema del significato

Pensate alle differenze con specifiche matematiche (e.g. addizione)

Formalismi

Formalismo: notazione con precisa sintassi e sematica

Specifiche formali vengono definite attraverso l'uso di notazioni formali

Vantaggi principali

- comprensione ?
- ambiguità ?
- completezza ?
- inconsistenza ?

Verifica formale: dato un formalismo ed una specifica formale è possibile verificare esattamente se alcune proprietà valgono oppure no direttamente "ragionando" sul modello.

Specifiche

formali, informali, semiformali

Specifiche formali sono tipicamente costose da definire. In alcuni casi può comunque essere utile poter avere specifiche più semplici e comprensibili ad un più ampio “pubblico”

Si distingue dunque tra:

- Formali
- semiformali
- informali

Specifiche

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

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

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

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

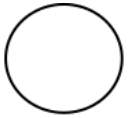
Caratteristiche

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:

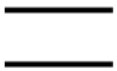
- I dati possono essere immagazzinati in repository
- I dati possono fluire all'interno del sistema
- I dati possono entrare o uscire dal sistema

Sintassi

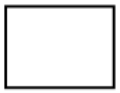


Simbolo utilizzato per rappresentare le funzioni

Simbolo utilizzato per rappresentare il flusso del dato



Simbolo utilizzato per rappresentare un repository



Simbolo utilizzato per rappresentare un dispositivo di input

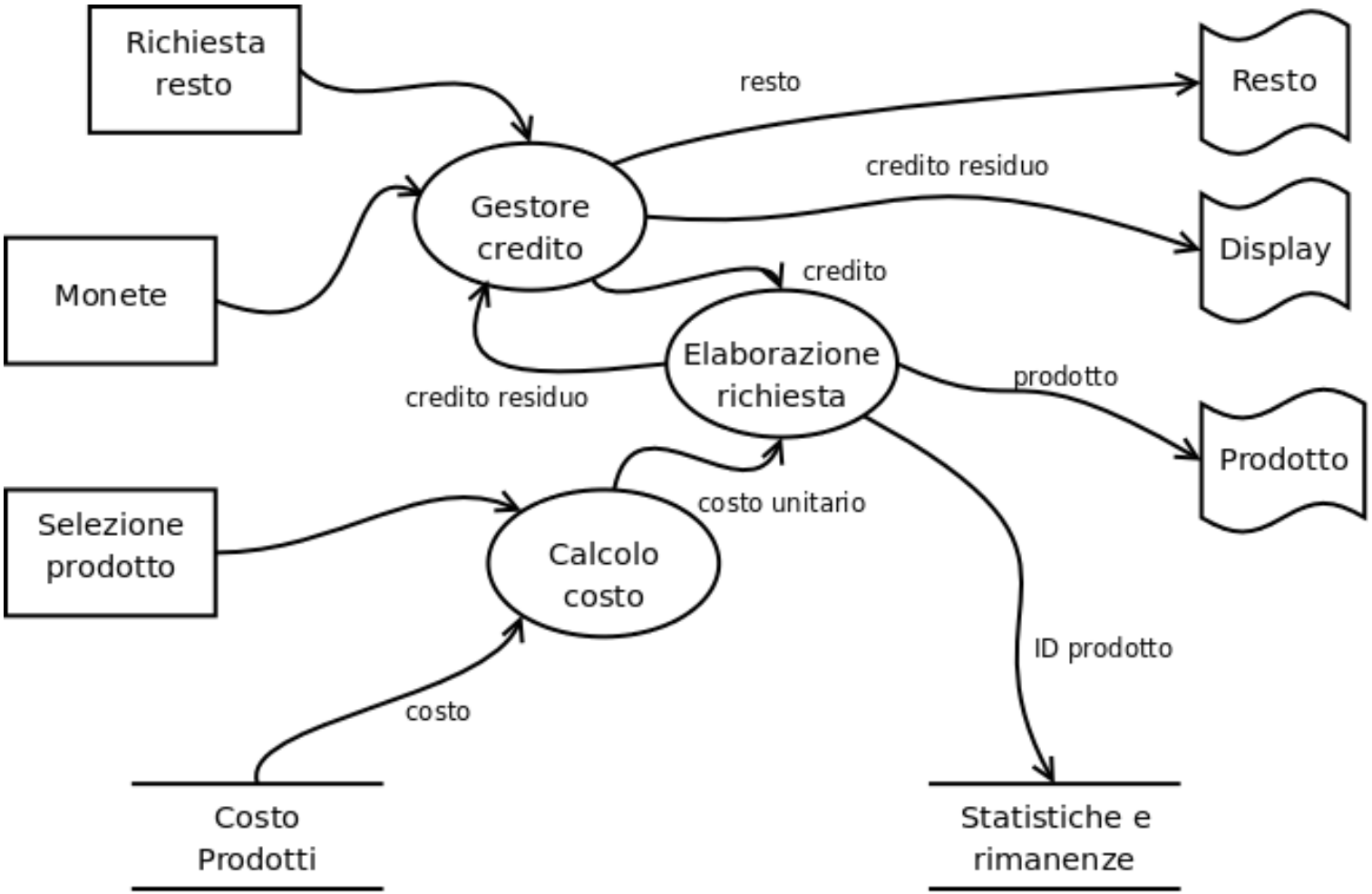
Simbolo utilizzato per rappresentare un dispositivo di output

Regole per combinare i simboli...

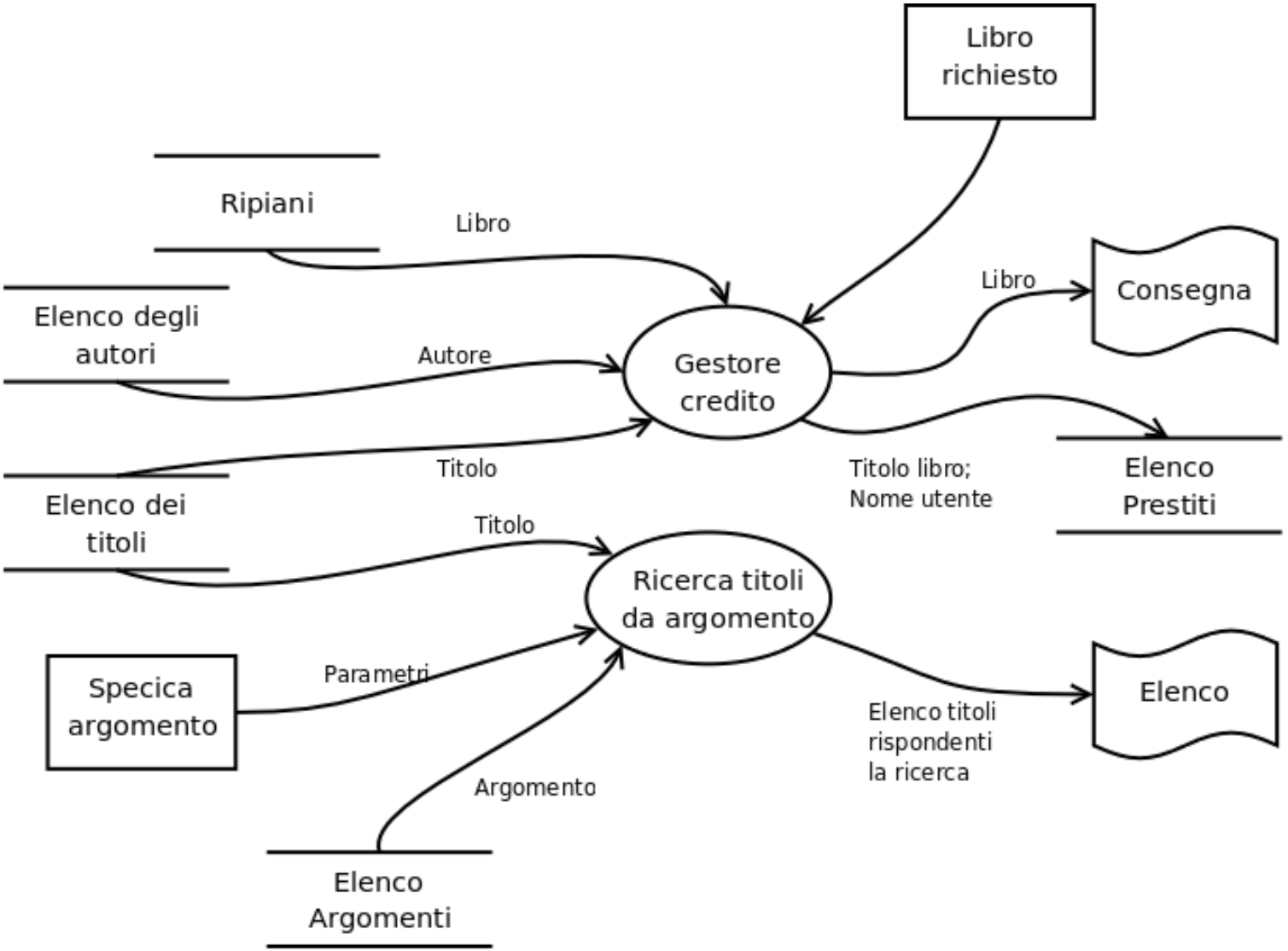
Qualche esempio

- 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

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 ($\mathcal{Q} \times \mathcal{I} \rightarrow \mathcal{Q}$)
- $q_0 \in \mathcal{Q}$ è lo stato iniziale

Notazione grafica . . .

FSM sono particolarmente utili a modellare il controllo di un sistema software.

Qualche esempio

- interruttore
- buffer di lettura scrittura - *un processo lettore, un processo scrittore ed un buffer di due posizioni*
- il distributore di bevande calde - *il distributore accetta monete da 5, 10, 20, 50 c€ ed 1€. La macchinetta eroga caffè al prezzo di 40c€, tè al prezzo di 30c€, cioccolato 45c€. La macchinetta non è capace di memorizzare un credito superiore ai 3€. La macchinetta da resto. Utilizzare un I/O FSM per modellare il distributore.*

Questioni

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. Allo stesso tempo il sistema è un sistema sincrono che non ha attività concorrenti.

Esplosione degli stati (e.g. somma di due interi) ed estensione del formalismo di base. Memoria limitata dal numero degli stati.

Possibili estensioni. I/O FSM. Transizioni con guardia ed azioni.

Sommario

Reti di Petri

Le reti di Petri sono un formalismo ideato negli anni 60 per modellare sistemi concorrenti. 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

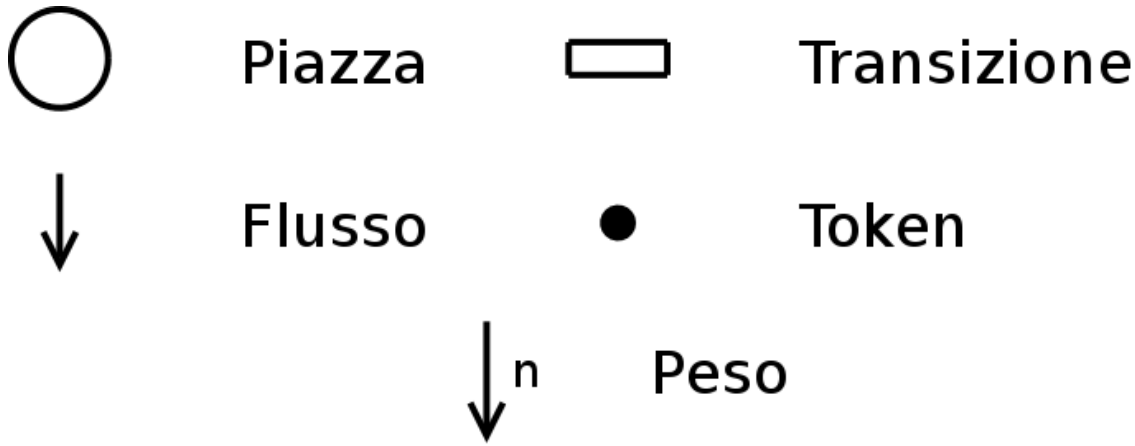
Deve poi valere che: $\mathcal{P} \cup \mathcal{T} \neq \oslash$ e $\mathcal{P} \cap \mathcal{T} = \oslash$

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:

- Piazze di input ed output
- Una transizione è abilitata se tutte le piazze di input contengono un numero di token uguale o maggiore dei pesi degli elementi flusso entranti sulla transizione
- Transizione con nessuna piazza di input sono sempre abilitate

Petri Net

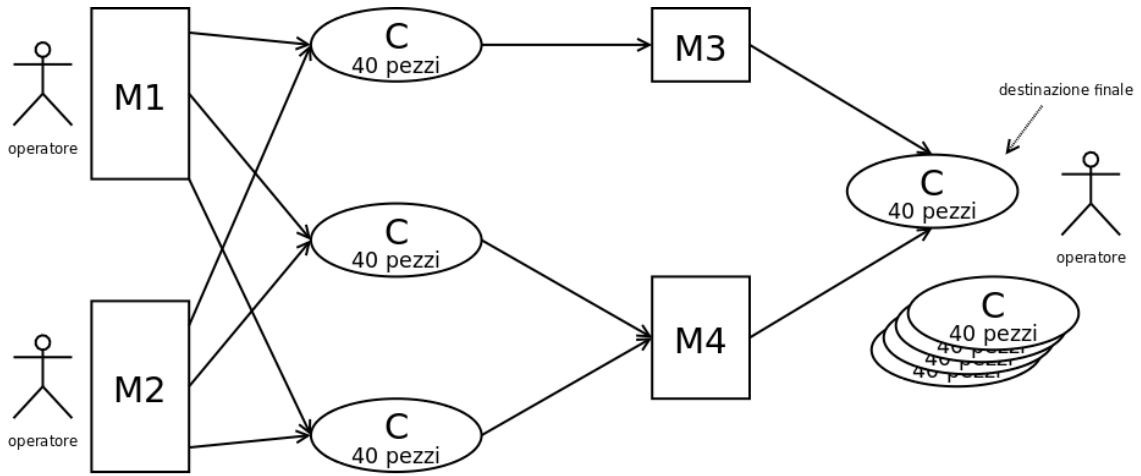
Tipicamente le piazze possono rappresentare una risorsa e l'accesso alla risorsa può essere modellato dalla presenza di Token.

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. Esempio...

Esempi

- Buffer di due posizioni
- Lettori/Scrittori
- filosofi a cena
- Semaforo
- Catena di montaggio



Problemi e possibili estensioni al modello

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

- Non è possibile modificare il flusso in dipendenza dei dati.

- Possibilità di selezionare una transizione tra più transizioni attive. Modello intrinsecamente non-deterministico.

- Possibilità di specificare tempo e deadline.

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

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. In relazione a DFD:

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

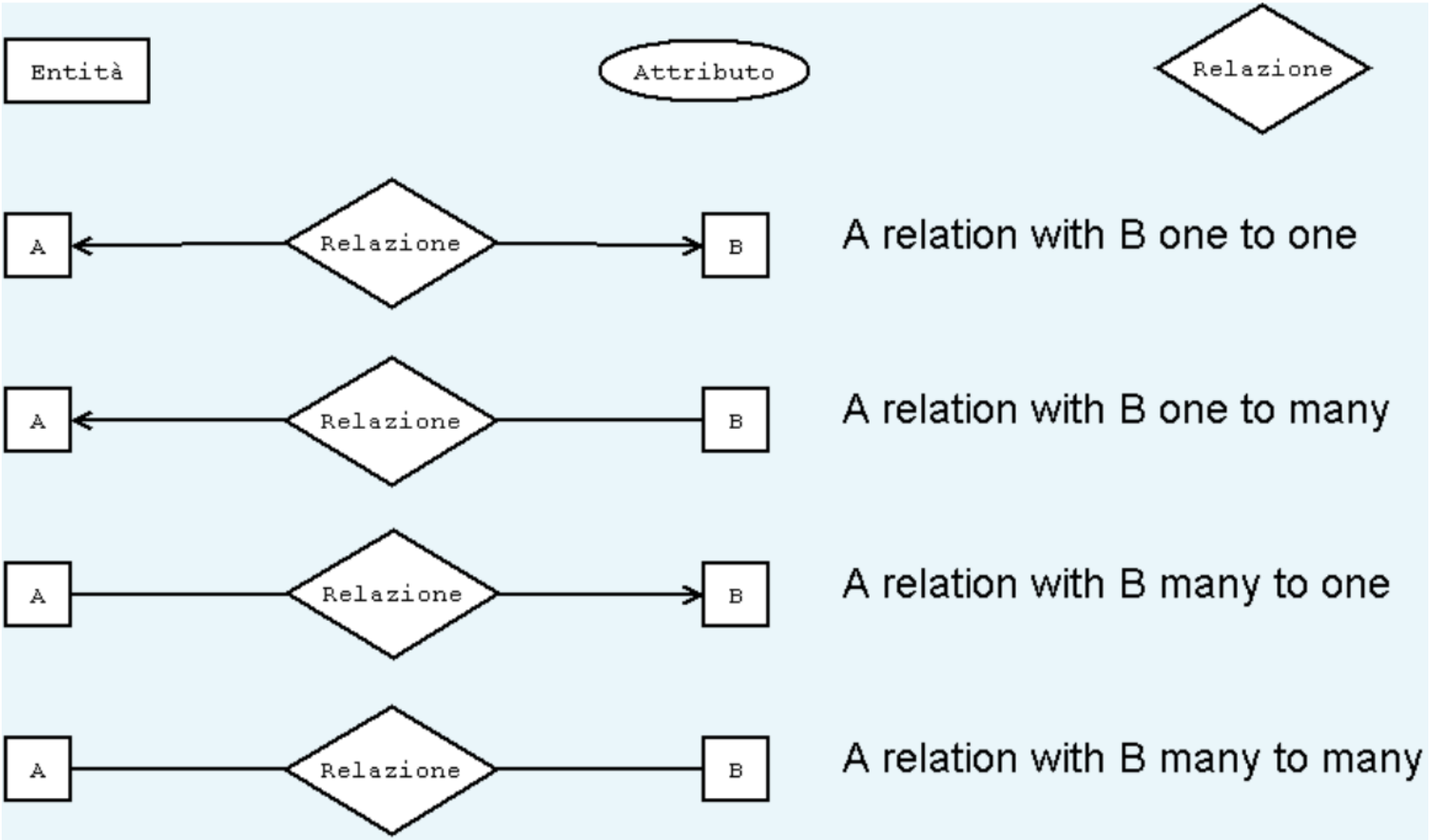
Vengono modellati tre concetti fondamentali:

- Entità
- Attributi
- Relazioni

Diagrammi Entità-Relazione

notazione grafica

ER non sono stati standardizzati, dunque molte varianti



Limitazioni

Potere espressivo è piuttosto limitato. Concetti complessi di relazione non possono essere rappresentati. (e.g. Ereditarietà).

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 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. somma di n numeri
2. programma che genera permutazione
3. programma che verifica che il primo elemento di un vettore sia duplicato

Specifiche di parti

Specifiche logiche possono essere utilizzate anche per specificare parti di programma . . . richiede di poter utilizzare le variabili di programma all'interno della specifica logica

Esempi...

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

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

Specifiche logiche

considerazioni

Anche semplici programmi generano specifiche piuttosto complesse.

Definizione e riuso di formule

Spesso associate ad altri formalismi per aumentarne la potenza espressiva. E.g. FSM e formalismi logici il caso delle code.

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

Si supponga di voler specificare tramite formalismo logico la funzione (`findInVect`) che dato un vettore di interi di dimensione non nulla non contenente duplicati, e dato un intero positivo, restituisce un dato astratto di tipo `Result` contenente due campi publici, `found` di tipo booleano e `index` di tipo intero. Tali campi assumeranno rispettivamente il valore `true` e l'indice corrispondente alla posizione nel vettore nel caso in cui l'elemento cercato sia presente. Altrimenti i due campi assumeranno rispettivamente valore `false` e -1 nel caso in cui l'elemento non sia presente nel vettore.
Si definiscano dunque la Pre-condizione e la Post-condizione relativa alla seguente definizione della funzione:

```
findInVect(in:int VI[], in:int val, out:Result res)
```

dove il tipo `Result` risulta essere così definito:
abstract data Result boolean found; int index;
Nella definzione della specifica è possibile utilizzare l'operatore di `.` per accedere ai campi del tipo di dato astratto `Result`. Inoltre il meccanismo di accesso agli elementi del vettore `VI[i]` permette di indicare l'elemento i-esimo del vettore. È infine possibile utilizzare la funzione `length(in:int V[], out:int l)` che dato un vettore in ingresso ne restituisce la dimensione.

Si utilizzi la funzione `findInVect(...)` sopra definita nella specifica della Pre- e Post-Condizione della funzione `checkAllValue` così definita:

```
checkAllValue(in:int VI[], in:int upper, in:int lower, out:boolean check)
```

Tale funzione prende in ingresso un vettore di interi positivi e due interi positivi di cui il primo (`upper`) maggiore o uguale del secondo (`lower`). Come risultato la funzione restituisce il valore `true` nel caso in cui tutti i valori compresi tra il primo ed il secondo valore siano presenti nel vettore. In caso negativo la funzione restituirà `false`.
Nella definizione della funzione si utilizzi la funzione

```
removeDuplicate(in:int VI[], out:int VO[])
```

tale funzione dato un vettore che può contenere duplicati ne restituisce uno in cui i duplicati sono stati rimossi.

Uso di specifiche logiche

Modellare la macchinetta del caffè

- Insiemi: c:coins, d:drinks, t:time, n:int
- Funzioni: prices(d):c,
- Eventi: insertCoins(c,t), giveChange(t), selectDrink(d,t), readyDrink(d,t) ...
- Stati: servingDrink(d,t_1,t_2), credit(c,t_1,t_2), availDrink(d,t_1,n)

Descrivere relazioni e proprietà

- funzionamento
- credito costante finchè non c'è inserimento monete
- selezione non disponibile durante servizio
- ...

Sommario

Specifiche algebriche

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

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

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

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

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

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

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

Esercizio

Confronto tra i diversi tipi di specifica e dei meccanismi di analisi applicabili:

Obiettivo: Salvare capra e cavoli :-)