Linguistica Computazionale – Lezione 3

Strumenti Informatici

8 Marzo 2007

Cristiano Chesi, chesi@media.unisi.it

Strumenti informatici

- Indice
  - Fondamenti
    - Macchine di Turing (universali)
    - Concetto di computazione e computabilità
    - Dati, programmi, input e output
  - Basi Dati
    - Corpora
    - database
    - strumenti di interrogazione basi dati (esp. regolari, SQL)
  - Algoritmi
    - Cicli ed oggetti
    - Ideazione, descrizione, formalizzazione ed implementazione di un algoritmo

Letture, approfondimenti

- **Bibliografia essenziale**
  - Hutchins & Somers (1992) cap. 3
- **Approfondimenti**
  - Lenci, Montemagni & Pirrelli (2005) *Testo e Computer: Elementi di Linguistica Computazionale*. Carocci, Roma
  - Hopcroft, Motwani & Ullman (2001) *Introduction to the automata theory, languages and computation*. Addison-Wesley. Boston
  - Frixione & Palladino (2004) *Funzioni, Macchine, Algoritmi*. Carocci. Roma

Macchine di Turing (universali)

- **Concetto di computabilità**

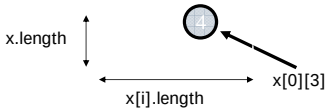
  (informalmente) per **computazione** si intende la specificazione della **relazione tra un input ed un output**. Questa relazione può essere definita a vari livelli, ma fondamentalmente consiste nella descrizione di una **serie di stadi** intermedi in cui l'informazione di input può venire trasformata prima di raggiungere la forma dell'output e nella definizione delle specifiche di trasformazione. Un problema computazionale tenta quindi di ridurre ogni input ad output seguendo una serie di passi consentiti dal modello computazionale.

- **La tesi di Turing-Church**

  ogni compito computazionale che può essere realizzato da un qualsiasi dispositivo fisico può essere realizzato anche da una macchina di Turing. Inoltre se il dispositivo fisico riesce a completare il compito in F(n) passi, con *n* uguale alla dimensione dell'input, la macchina di Turing ci riuscirà in G(n) passi, con F che differisce da G di al massimo un polinomiale.

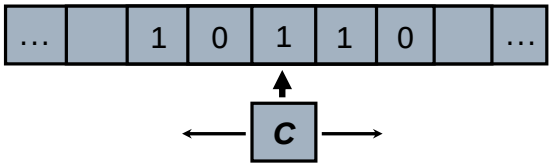
Input e Output

- Affinché un programma possa accettare un **input**, questo input deve essere conforme a certe specifiche dichiarate a priori:
  formato (codifica, es. ASCII, UTF-8, binary)
  struttura (impacchettamento, es. header, TCP-IP... DTD)
- Un **output** deve o sottostare agli stessi standard oppure essere human readable (di solito si preferisce avere output in formato standard, es. XML e poi gestire la visualizzazione, cioè l'interfaccia, con moduli specifici).
- Una funzione (es. f(x) = y) è definita nel suo dominio di applicazione (input: x ∈ X) e di proiezione (output (range) : y ∈ Y)
  - String x = "scuola"
  - Int x = 14
  - Int[][] x = {{1, 2, 3, 4, 5 }, {6, 7, 8, 9, 10} }



Macchine di Turing

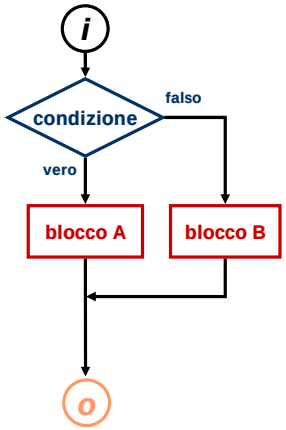
- nastro infinito diviso in celle
- alfabeto ***A*** (di almeno 2 elementi, ad esempio ***A*** ={0,1,∅})



- cursore ***C*** (che scorre a sinistra e a destra, che può leggere, cancellare e scrivere un carattere)
- insieme finito ***Q*** di stati (q_0, $q_{1 \ldots} q_n$)
- input finito ***I*** costituito da caratteri in A
- insieme finito ***S*** di stati della macchina descritte da quintuple del tipo $<q_iabvq_j>$ dove $q_i, q_j \in \mathbf{Q}$; $a,b \in \mathbf{A}$; $\boldsymbol{v}$ = {destra, sinistra}

Flow charts

- **Grafo orientato etichettato** costituito da:
  - un **ingresso** (dove viene introdotto l'input *i*)
  - una o più **uscite** (da cui viene, eventualmente, recuperato un output *o*)
  - un insieme finito di **blocchi di istruzioni** tali che ogni istruzione è del tipo X = Y, X = X+1, X = X-1
  - un insieme finito (possibilmente nullo) di blocchi speciali, detti **condizioni** tali da presentare interrogazioni booleane del tipo X = Y?
  - un insieme finito di **connettori** che collegano i blocchi, tali che da ogni blocco esce 1 ed 1 sola freccia, mentre dal blocco condizionale escano 2 frecce



Modularità

(inizio ...)

- **Macchine di Turing** e **flow charts** sono equivalenti: esprimono la stessa classe di funzioni: le funzioni computabili.
- Entrambi i formalismi godono della proprietà della **composizionalità** ($M_1 \bullet M_2$). Questo ci garantisce che un algoritmo può in realtà essere il risultato di una composizione di più macchine di Turing (o di più flow charts).
- Si dice "**divide et impera**" il paradigma di programmazione che suggerisce di scomporre un problema nelle sue sottoparti prima di iniziare a pensare agli algoritmi che lo risolvono.

Modularità

(… fine)

- Una prima distinzione può essere fatta tra **dati** e **programmi** (**decoupling**) dove per **dato** si intende un oggetto "inerte" che può essere oggetto (input) o risultato (output) della computazione. La parte più strettamente algoritmica viene descritta dal **programma**.
- In genere modularizzare un problema conviene per i seguenti motivi:
  - **divisione del lavoro** (capire cosa può essere svolto in parallelo)
  - **riutilizzo** (capire quali saranno le funzioni più generali e riutilizzarle)
  - **analisi modulare** (cicli di sviluppo più rapidi: progetto > implementazione > testing > debugging …)
  - **modifiche locali** (maggiore scalabilità)

La struttura dei dati

- **Linguaggi naturali e linguaggi "taggati"**
  - parentesi [[A] [B C]]
  - HTML 123 Mario Rossi
  - XML
    ```
    <studente> <id> 123 </id>
    <nome> Mario Rossi </nome>
    </studente>
    ```
  - voiceXML
    ```
    <vxml version="2.0">
    <form>
    <block>123, Mario Rossi</block>
    </form>
    </vxml>
    ```


Un esempio di linguaggio taggato

(inizio …)

- Il **VoiceXML** permette di standardizzare la creazione di dialoghi (uomo-servizio informatizzato) basati sulla modalità vocale in particolare (Interactive Voice Response, IVR):
  - sintesi del parlato (o utilizzo di audio digitalizzato)
  - toni della tastiera (Dual Tone Multi Frequency, DTMF)
  - riconoscimento di comandi vocali
  - specificazione di grammatiche

Un esempio di linguaggio taggato

(… continua …)

- **Un documento VoiceXML** (applicazione):

```
<vxml version="2.0">
    <form>
        <field name="prezzo" type="boolean">
        <prompt> Vuoi sapere il prezzo del biglietto?  </prompt>
        <filled> Ok!
        <if cond="prezzo"> Ecco qua:
                <goto next="prezzo.vxml" />
        <else /> Allora torniamo alla lista…
                <goto next="lista.vxml" />
        </if>
        </filled>
        </field>
    </form>
</vxml>
```

Un esempio di linguaggio taggato

(… continua …)

- **Specificazione di una grammatica in VoiceXML** (Speech Recognition Grammar Specification, SRGS):

```
<grammar xml:lang="IT" type="application/srgs+xml" version="1.0"
    mode="voice">
    <rule id="yes_no_cancel" scope="public">
        <one-of>
            <item tag="no">no</item>
            <item tag="yes">sì</item>
            <item tag="yes">va bene</item>
            <item tag="cancel">annulla</item>
        </one-of>
    </rule>
</grammar>
```


Un esempio di linguaggio taggato

(… fine)


```
<grammar version="1.0" mode="voice" root="basicCmd">

<rule id="basicCmd" scope="public">
    <example> please move the window </example><example> open a file </example>
    <ruleref uri="#command"/>
</rule>
<rule id="command">
    <ruleref uri="#action"/> <ruleref uri="#object"/>
</rule>

<rule id="action"> <one-of>
    <item weight="10"> open <tag>TAG-CONTENT-1</tag> </item>
    <item weight="2"> close <tag>TAG-CONTENT-2</tag> </item>
    <item weight="1"> delete <tag>TAG-CONTENT-3</tag> </item>
    <item weight="1"> move <tag>TAG-CONTENT-4</tag> </item>
    </one-of> </rule>
<rule id="object">
    <item repeat="0-1">
        <one-of> <item> the </item> <item> a </item> </one-of>
    </item>
        <one-of> <item> window </item> <item> file </item> <item> menu </item>
        </one-of>
</rule>
</grammar>
```


Cosa sono, che struttura hanno

- collezioni **finite** di informazioni, **omogenee** e **rappresentative** rispetto ad un dominio, raccolte in un modo **sistematico**, in **condizioni controllate** in modo da riflettere la **reale distribuzione** (quantitativa e qualitativa) dei fenomeni linguistici che si intendono studiare
  - **non strutturate** (unica informazione presente è l'informazione linguistica del testo)
    es. files di testo con formattazione non significativa (colonne, giustificazione…)
  - **strutturate** (convenzioni precise indicano la natura dei dati linguistici)
    es. database, testo taggato
  - **semistrutturate** (convenzioni implicite forniscono implicitamente informazioni (extra)linguistiche)
    es. pagine html, testi formattati (titoli, paragrafi, sottoparagrafi, grassetti, corsivi ecc.)

Corpora linguistici

(inizio …)

- **Brown corpus** (Francis and Kucera, 1964)
  - circa un milione di parole rappresentativo dell'inglese americano scritto (500 testi del 1961)
  - i testi sono raccolti in 15 categorie:
    - A. stampa: reportage (44 texts)
    - B. stampa: editoriali (27 texts)
    - C. stampa: periodici (17 texts)
    - D. religione (17 texts)
    - E. arti e mestieri (36 texts)
    - F. tradizioni popolari (48 texts)
    - …
  - Esempio:
    A01 0010 The Fulton County Grand Jury said Friday an investigation
    A01 0020 of Atlanta's recent primary election produced "no evidence" that
    A01 0030 any irregularities took place. The jury further said in term-end
    A01 0040 presentments that the City Executive Committee, which had over-all
    A01 0050 charge of the election, "deserves the praise and thanks of the
    A01 0060 City of Atlanta" for the manner in which the election was conducted.

Esempio di corpus linguisticamente strutturato

(… continua …)

- **Penn Treebank** (Marcus & al. , 1989-1992)
  - 1 milione di parole provenienti da articoli del 1989 del Wall Street Journal
  - Un campione di materiale proveniente da ATIS-3 (Automatic Terminal Information Service)
  - Etichettatura secondo lo "standard" Treebank II style
  - esempio:

    (S (PP (IN Of) (NP (NN course))) (, ,) (S (S (NP (DT some) (PP (IN of) (NP (PRP$ my) (NN color) (NNS values)))) (AUX (VBP do)) (NEG (RB not)) (VP (VB match) (NP (NP (DT the) (JJ old) (NN Master)) (POS 's)))) (CC and) (S (NP (DT the) (NN perspective)) (VP (VBZ is) (ADJP (JJ faulty)))) (CC but) (S (NP (PRP I)) (VP (VBP believe) (S (NP (PRP it)) (AUX (TO to)) (VP (VB be) (NP (DT a) (JJ fair) (NN copy))))))))

Esempio di corpus linguisticamente semi-strutturato

(… fine)

- **Childes** (MacWhinney & Snow, 1985)
  - (**Chi**ld **L**anguage **D**ata **E**xchange **S**ystem) è un archivio di trascrizioni spontanee di bambini (solitamente dai 14 mesi ai quattro anni di età) che interagiscono con adulti in varie situazioni. Generalmente ogni trascrizione si riferisce ad una conversazione di durata variabile dai 20 ai 60 minuti.
  - Le trascrizioni sono codificate secondo il formato standardizzato, detto **CHAT**

```
@UTF8
@Begin
@Participants:   CHI Cam Target_Child, DON Mother
@Age of CHI:     3;4.9
@Sex of CHI:     female
@Birth of CHI:   3-MAY-1988
@Date:           12-SEP-1991
*DON:            quale volevi ?
*CHI:            io volevo questo .
*DON:            si ma cosa, che canzoni ci sono, sopra .
*CHI:            non lo so .
*DON:            come non lo sai ?
                 […]
@End
```


A cosa servono i corpora

- Informazioni quantitativamente significative per implementare **sistemi esperti** o per l'**estrazione di grammatiche**
  - registrazioni telefoniche (call center)
  - corpora taggati

  …
- **Analisi linguistiche** specifiche
  - acquisizione prima lingua
  - acquisizione seconda lingua
  - soggetti con disturbi linguistici (Specific Language Impairment, sordi, afasici …)

  …

Database relazionali

- **tabelle**

studenti

ID	nome	cognome	mail	…
1	aldo	rossi	aldo@…	
2	giovanni	bianchi	gio@…	
3	giacomo	verdi	gia@…	
…				

- **relazioni (o link relazionali o join)**

studenti

ID	nome	cognome	stato
1	aldo	rossi	1
2	giovanni	bianchi	2

iscrizione

ID	descrizione
1	presente
2	assente

Database relazionali - normalizzazione

- **procedura di normalizzazione**

  far in modo che ogni informazione **occorra una sola volta** nella struttura dati.

  Si preferisce quindi far riferimento ad un'unica istanza di questa informazione (via **link relazionale**) piuttosto che copiare la stessa informazione in più posizioni (risolve il problema dell' "**update anomaly**": alcune istanze dell'informazione sono aggiornate, mentre altre sfuggono al controllo e diventano obsolete)

Interrogazioni su corpus testuali

- **Espressioni Regolari**
  - notazione algebrica per definire insiemi di stringhe di testo (linguaggi di **tipo 3**).
  - Il cuore dell'espressione regolare è il **pattern di identificazione** composto da caratteri alfanumerici (compresi segni di spaziatura e di interpunzione) e da segni speciali volti a stabilire le relazioni tra i caratteri del pattern.

Espressione Regolare	Corrispondenza	Es. pattern identificato
[Dd]uomo	Duomo oppure duomo	Il duomo è nella piazza
[^a-z]	tutto fuorché lettere minuscole	Il duomo è …
sali?ta	salita oppure salta	Marco deve saltare
sal.ta	accetta ogni carattere tra le i e la t	Marco saluta
bu*	b seguito da un numero imprecisato (anche nullo) di u	buuuuuu! oppure b!
^L Vs. a$	^ = inizio stringa; $ = fine stringa	La casa
cas(a|e)	| è equivalente alla disgiunzione logica	Marco vive in un casale
*	il backslash è il simbolo di escape	A*

Note su Eliza

- **Espressioni Regolari e l'operazione di Sostituzione**
  - La **sostituzione** è un'operazione che permette di sostituire l'occorrenza di un'espressione regolare con un'altra espressione regolare e può essere definita come segue:
  - **s/espressione_regolare1/espressione_regolare2/**
  - **s/ www\.[a-z]*\.com / www\.pe{2}\.com/**
- **Registri**: se si usano più blocchi di operatori (ogni parentesi tonda delimita un blocco), si può riutilizzare l'espressione trovata da un determinato blocco nell'espressione da sostituire, facendo riferimento all'ordine dei blocchi nel pattern di ricerca:
  - **s/ la (casa|macchina) è stata comprata da (Maria|Gianni)/ \2 ha comprato la \1 /**

    permette di costruire la forma attiva (Gianni ha comprato la casa) della frase passiva (la casa è stata comprata da Gianni).
- operazioni di sostituzione in ELIZA:
  - s/ sono [.* |](depress[o|a]|triste)/sono spiacente di sapere che sei \1/
  - s/ sono tutt[i|e] (.*) /in che senso sono \1?/
  - s/ sempre / potresti far riferimento ad un esempio specifico?

Database relazionali - Interrogazione

- **Structured Query Language (SQL)**

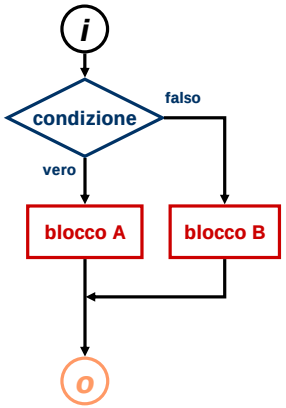
  Linguaggio di interrogazione e manipolazione di database

  - **Data Manipulation Language (DML)**
    - **select**
      select * from studenti where id > 1
    - **insert**
      insert into studenti (nome, cognome) values ("mario", "rossi")
    - **delete**
      delete from studenti where id=10
    - **update**
      update studenti set nome="gianni" where id=11
  - **Data Definition Language (DDL)**
    - **create database**
    - **create table**
    - **drop database**

Controllo di flusso condizionale: If ... else ...


```
if (condizione-booleana) {
  blocco A
}
else {
  blocco B
}
```

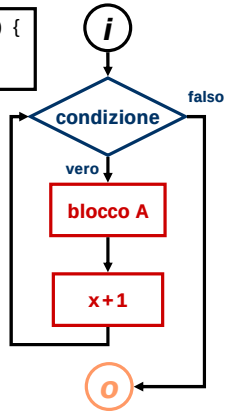
```
if (i < 10) {
  i + 1;
  o = i;
}
else {
  o = 10;
}
```



Cicli determinati

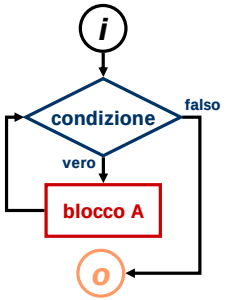

```
for (input; condizione-booleana; incremento) {
  blocco
}
```

```
for (int i = 0; i < 10; i++) {
  System.out.println(i);
}
```



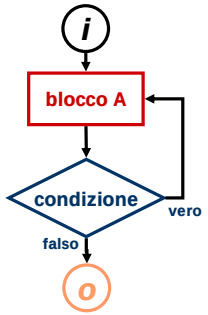
Cicli indeterminati

- Cicli While




```
while (boolean-cond) {
  blocco
}
```

- Cicli Do ... While




```
do {
  blocco
} while (boolean-cond)
```


Interruzioni dei cicli

- **Break** interrompe l'esecuzione di un ciclo ed esce definitivamente dal ciclo (while, do while, for)

- **Continue** interrompe la corrente iterazione e ritorna all'inizio del ciclo, iniziandone un'altra.

```
for (int i = 0, i < 20; i++) {
  if (i = 0)
    continue;
  if (i % 2 == 0)
    break;
  System.out.println(i);
}
```

Strutture dati ed OGGETTI

- **Oggetto**
  entità costituita da **proprietà** (il **valore** assegnato a tale proprietà in un determinato istante di tempo determina lo **stato** dell'oggetto) e **comportamenti** (metodi o procedure di modificazione dei dati proprie dell'oggetto).
  Un oggetto complesso è un oggetto costituito da altri oggetti (funzione di **estensione**).

- **Identità**
  si dice **Object IDentifier** (**OID**) l'identificativo unico dell'oggetto, indipendente dai valori che tale oggetto assume (avere lo stesso identificativo è diverso dall'avere gli stessi valori!)

Strutture dati ed OGGETTI – EREDITARIETA'

- Gli oggetti possono essere definiti gerarchicamente: ogni oggetto gerarchicamente inferiore eredita proprietà e metodi dagli oggetti padri

- Nuovi metodi/proprietà possono essere inclusi nei figli, gli stessi metodi ereditati possono essere ridefiniti (**overriding**)

- si può parlare di ereditarietà semplice (classi>sottoclassi) oppure ereditarietà multipla (reticolo aciclico)

Formalizzazione di un algoritmo

- **UML (Unified Modeling Language)**

http://www.uml.org/

http://www.sparxsystems.com.au/resources/uml2_tutorial/index.html

Use case

- **Commenti generali** che descrivono il caso d'uso;
- **Requisiti**: cosa deve permettere di fare il sistema (modifica, aggiornamento...);
- **Vincoli**: cosa (non) si può fare: stato del sistema prima dell'uso, rapporti di propedeuticità tra funzioni, condizioni di output, condizioni sempre presenti;
- **Scenari**: descrizioni testuali delle modalità d'uso;
- **Diagrammi di scenario**: descrizioni diagrammatiche degli scenari sopra descritte;
- Altri attributi quali fase di implementazione, versione, grado di complessità...

Formalizzazione di un algoritmo

Elementi di un use case

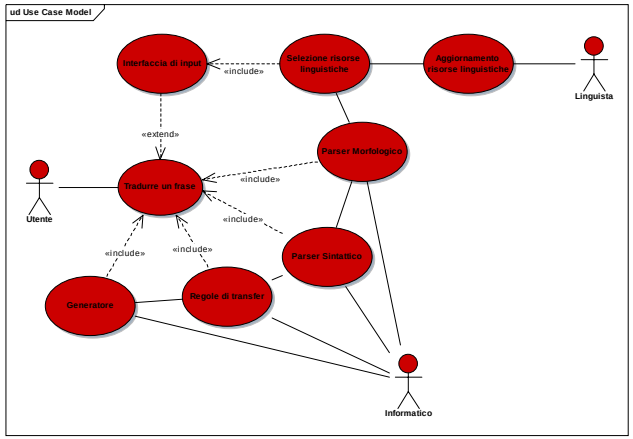
- **Attori**: entità attive che interagiscono dall'esterno con il sistema

- **Casi d'uso**: un modello di comportamento implementato nel sistema

- **Relazioni**, sono link tra attori e casi d'uso:
  - **Associazione:** collega un attore al caso d'uso con il quale interagisce
  - **Estensione:** estende il caso d'uso A al caso d'uso B (il secondo può includere, in determinate condizioni, il comportamento del primo)
  - **Generalizzazione:** collega un caso d'uso più generale ad uno più specifico che eredita dal primo alcune funzionalità aggiungendone, possibilmente, altre
  - **Inclusione:** collega un caso d'uso di base ad uno più generale che ne contiene le funzionalità

- **Contesto**: rappresenta un sottoinsieme dei casi d'uso dell'intero sistema

Esempio di Use Case Model



Vari tipi di diagrammi in UML 2

1. Structural Modeling Diagrams

Definiscono la struttura statica del modello. Servono per modellare classi, oggetti e componenti fisici con relative relazioni e dipendenze.

- **Package diagrams** are used to divide the model into logical containers or 'packages' and describe the interactions between them at a high level
- **Class or Structural diagrams** define the basic building blocks of a model: the types, classes and general materials that are used to construct a full model
- **Object diagrams** show how instances of structural elements are related and used at run-time.
- **Composite Structure** diagrams provide a means of layering an element's structure and focusing on inner detail, construction and relationships
- **Component diagrams** are used to model higher level or more complex structures, usually built up from one or more classes, and providing a well defined interface
- **Deployment diagrams** show the physical disposition of significant artefacts within a real-world setting.

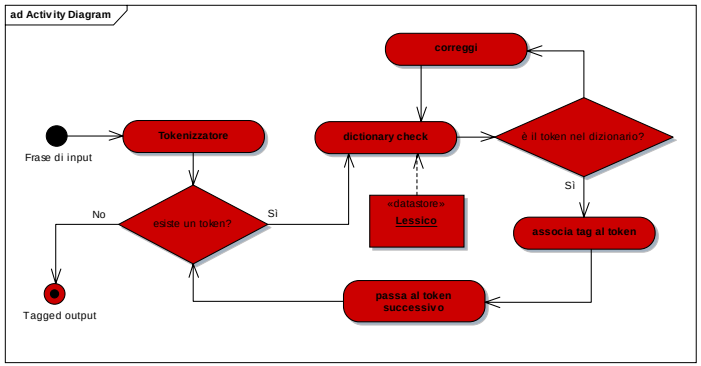
Vari tipi di diagrammi in UML 2

2. Behavioral Modeling Diagrams

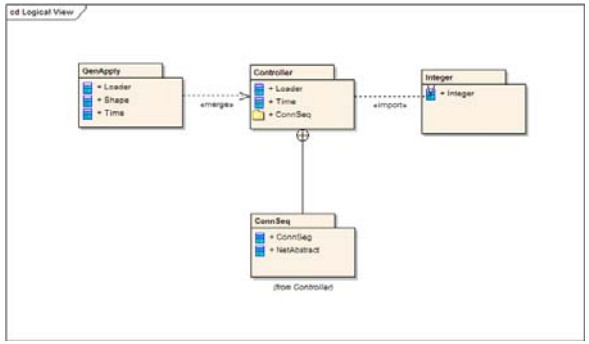
Questi diagrammi descrivono le varie relazioni e gli stati che il modello raggiunge durante il processamento

- **Use Case diagrams** are used to model user/system interactions. They define behavior, requirements and constraints in the form of scripts or scenarios
- **Activity diagrams** have a wide number of uses, from defining basic program flow, to capturing the decision points and actions within any generalized process
- **State Machine diagrams** are essential to understanding the instant to intant condition or "run state" of a model when it executes
- **Communication diagrams** show the network and sequence of messages or communications between objects at run-time during a collaboration instance
- **Sequence diagrams** are closely related to Communication diagrams and show the sequence of messages passed between objects using a vertical timeline
- **Timing diagrams** fuse Sequence and State diagrams to provide a view of an object's state over time and messages which modify that state
- **Interaction Overview diagrams** fuse Activity and Sequence diagrams to provide allow interaction fragments to be easily combined with decision points and flows

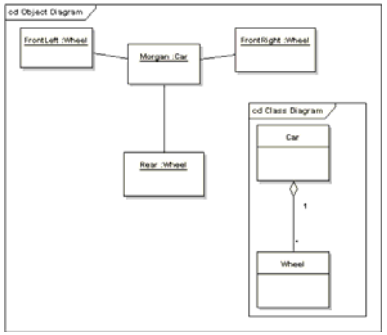
Esempio di Activity Diagram



Logical View



Diagrammi degli oggetti



Dati e programmi

- Difficile progettare sistemi **top-down** (prime suddivisioni funzionali troppo “lontane” dalla reale implementazione del sistema: difficile prevedere modularizzazioni efficienti)
- (di solito) è meglio **partire dai dati** e organizzare i moduli allo stesso livello di astrazione, cercando di individuare le dipendenze ed eventualmente raffinando parallelamente tutte le strutture.

Prossima lezione

(Martedì 13 Marzo, ore 16-18, Aula 456, Palazzo S. Niccolò)

- Analisi Morfologica
  - Lassico generativo
  - Morfologia a due livelli
  - Analisi morfologica con FSA
  - Stemming
- Normalizzazione dell’input
  - Classificazione errori
  - Correzione ortografica
  - T9