



GIS e Geo WEB: piattaforme e architetture

***Docente: Cristoforo Abbattista
eMail: abbattista@planetek.it***



IV lezione

Linguaggi e piattaforme di sviluppo

GIS e Geo WEB: piattaforme e architetture



Un pezzo per volta...

UML - Unified Modelling Language



Cosa il cliente dichiara' di volere



Cosa capi' il capo progetto



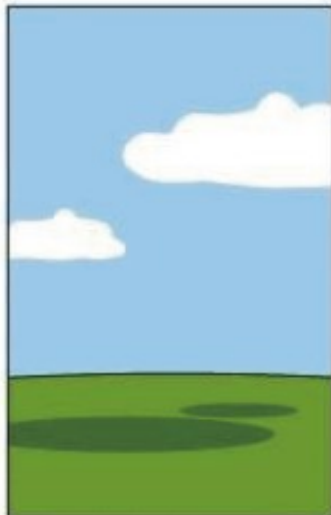
Come lo progetto' l'analista



Cosa programmo' il programmatore



Cosa defini' il consulente



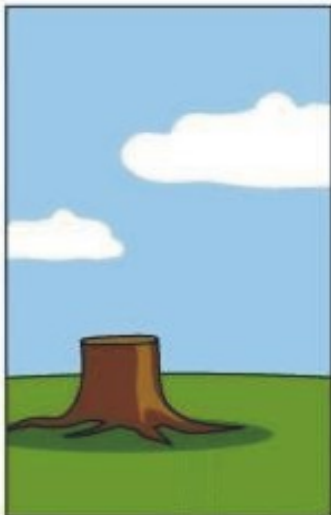
Come il progetto fu documentato



Cosa fu installato



Cosa fu messo in conto al cliente



Come fu eseguita la manutenzione



Quello di cui il cliente avrebbe avuto bisogno

Cosa è l'UML

Unified Modelling Language

- L' UML è un linguaggio standard di modellazione per realizzare sistemi software.
- UML è un linguaggio per documentare
 - le specifiche funzionali
 - l'architettura generale
 - la scomposizione strutturale del sistema
 - le interazioni tra i componenti
 - la distribuzione e il deployment
- UML è un linguaggio per stendere specifiche
 - costruire modelli che siano precisi, non ambigui e completi

Unified Modelling Language

- UML è uno standard
 - uniformità nei concetti e nelle notazioni utilizzate, interoperabilità tra strumenti di sviluppo
 - indipendenza dai produttori, dalle tecnologie, dai metodi
- UML è articolato
 - qualunque sistema software, a diversi livelli di astrazione
- UML è complesso
 - va adattato in base alle specifiche esigenze utilizzando solo ciò che serve

Il Meta-modello

- UML è basato su un meta-modello integrato, composto da numerosi elementi collegati tra loro
- molti elementi hanno una icona che li rappresenta graficamente
- Ogni diagramma coglie un aspetto diverso del sistema

Due livelli

- livello “logico”:
 - diagramma dei casi d’uso (use case)
 - diagramma delle classi (class)
 - diagramma di sequenza (sequence)
 - diagramma di collaborazione (collaboration)
 - diagramma di transizione di stato (state)
 - diagramma delle attività (activity)
- livello “fisico”:
 - diagramma dei componenti (component)
 - diagramma di distribuzione dei componenti (deployment)

Diagrammi dei casi d'uso

Gli Scenari

- I casi d'uso rappresentano le modalità di utilizzo del sistema da parte di uno o più utilizzatori (attori)
- descrivono l'interazione tra **attori** e sistema, non la “logica interna” della funzione
- sono espressi in forma testuale, comprensibile anche per i non “addetti ai lavori”
- ragionare sui casi d'uso aiuta a scoprire i requisiti funzionali
- I casi d'uso sono rappresentati dalla seguente icona:



Caso d'uso

I protagonisti

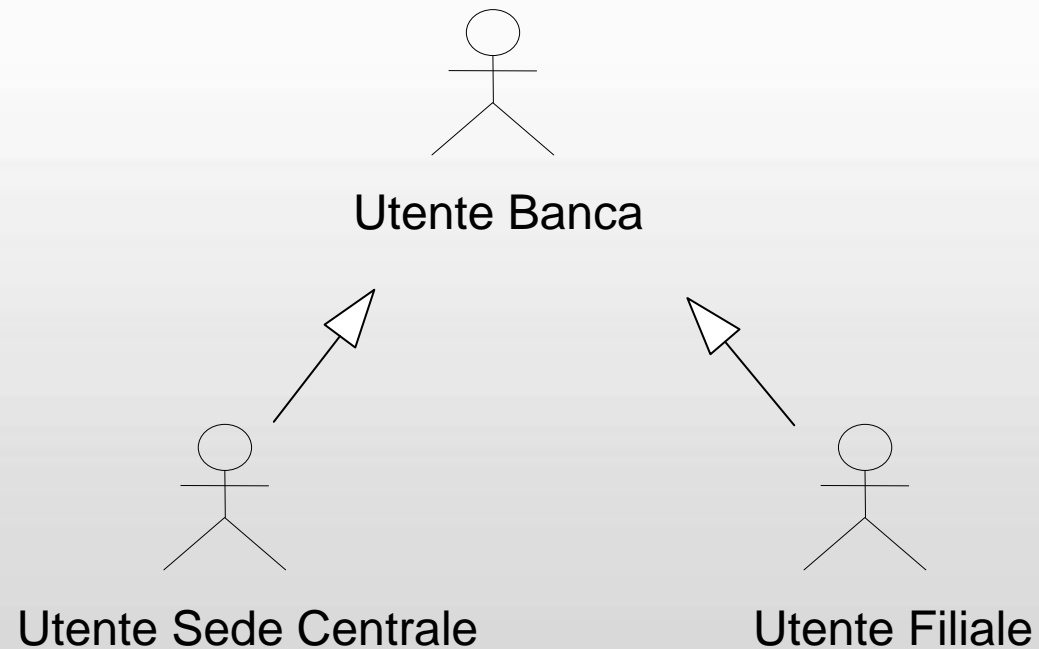
- Gli Attori rappresentano qualunque cosa che deve scambiare informazioni con il sistema
- Gli attori sono normalmente indicati tramite l'icona:



Relazioni tra attori

Relazioni

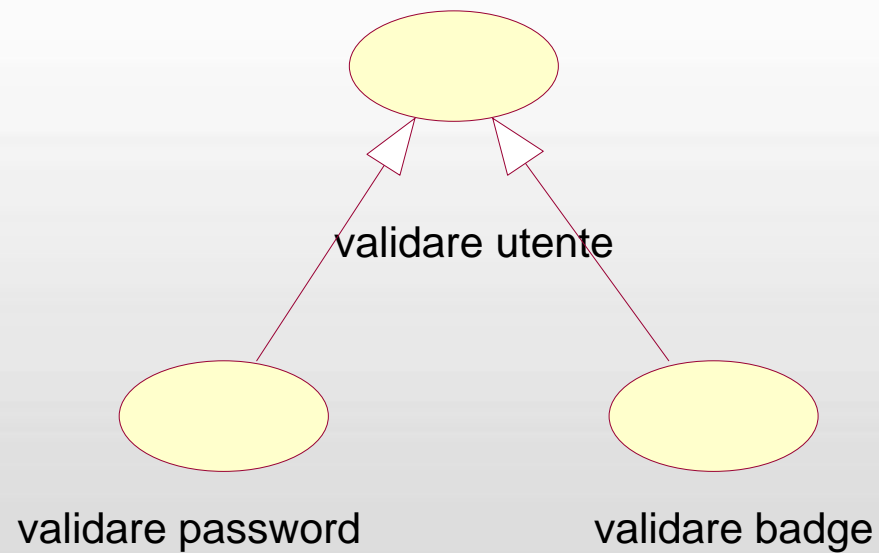
- E' possibile definire le relazioni tra gli attori



Relazioni tra casi d'uso

Generalizzazione e Specializzazione

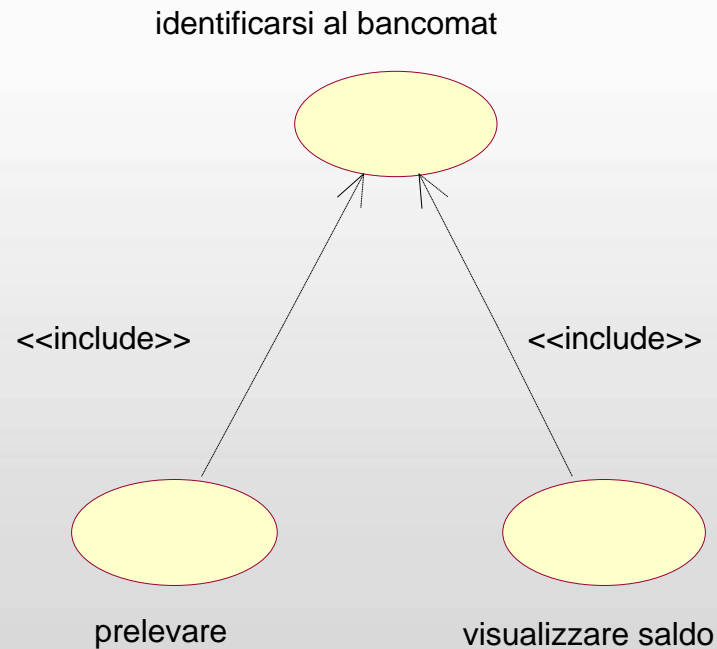
- Associare un caso d'uso ad uno o più casi d'uso specializzati



Relazione tra casi d'uso

Inclusione

- `<<include>>` : mostra il comportamento comune a uno o più casi d'uso

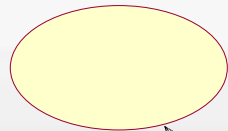


Relazione tra casi d'uso

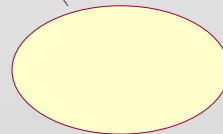
Estensione

- <<extend>>: mostra il comportamento opzionale (alternativo o relativo al trattamento di condizioni anomale)

aprire conto corrente



<<extend>>



trattare condizioni particolari

Diagramma dei casi d'uso

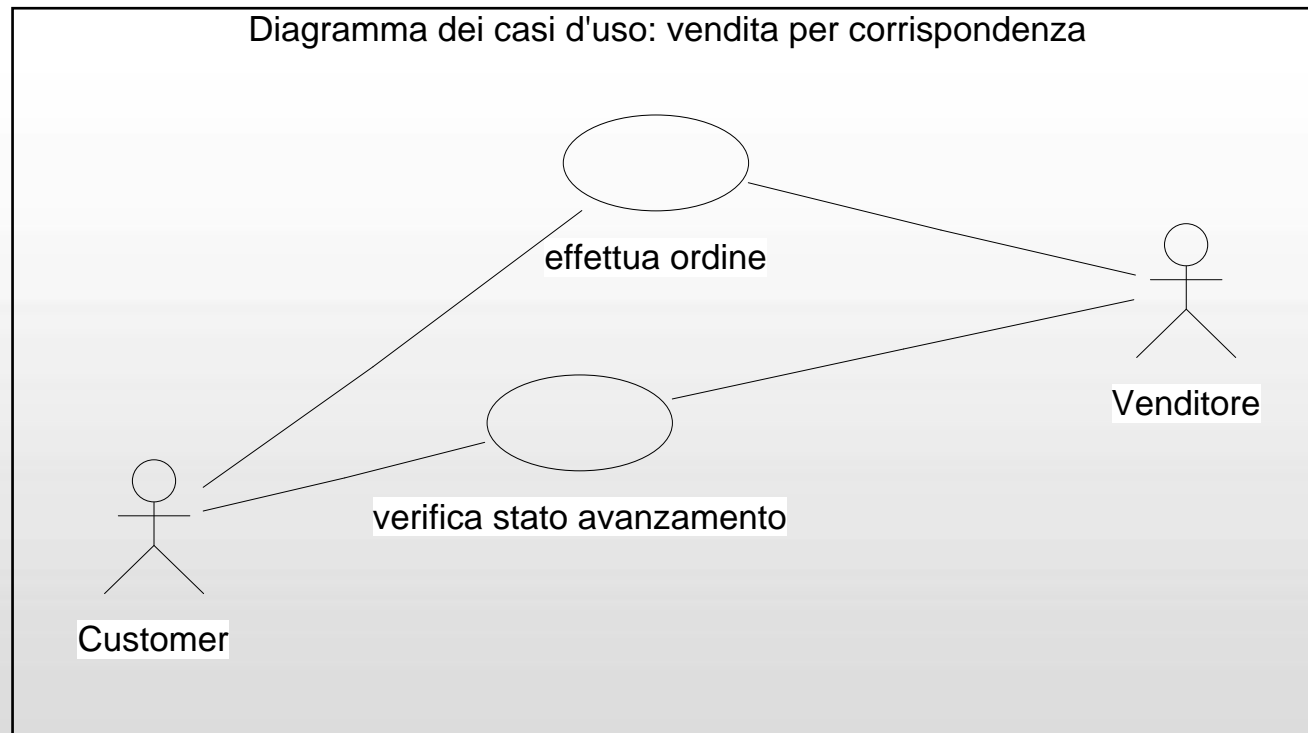


Diagramma delle classi

Descrivere il nostro sistema

- rappresenta le classi e gli oggetti che compongono il sistema, ed i relativi attributi ed operazioni
- Specifica, mediante associazioni, le relazioni fra le classi
- è un modello statico del sistema utilizzato
 - per ottenere un dizionario del sistema
 - avere un modello di collaborazione fra gli oggetti (p.e. per specificare quali oggetti partecipano a una transazione)
 - avere un modello logico dello schema del database

Classe

Elementi descrittivi

- **Nome**
- **Attributi:** proprietà i cui valori identificano un oggetto istanza della classe e ne costituiscono lo stato
- **Operazioni/Metodi:** insieme di funzionalità che esprimono il comportamento di un oggetto

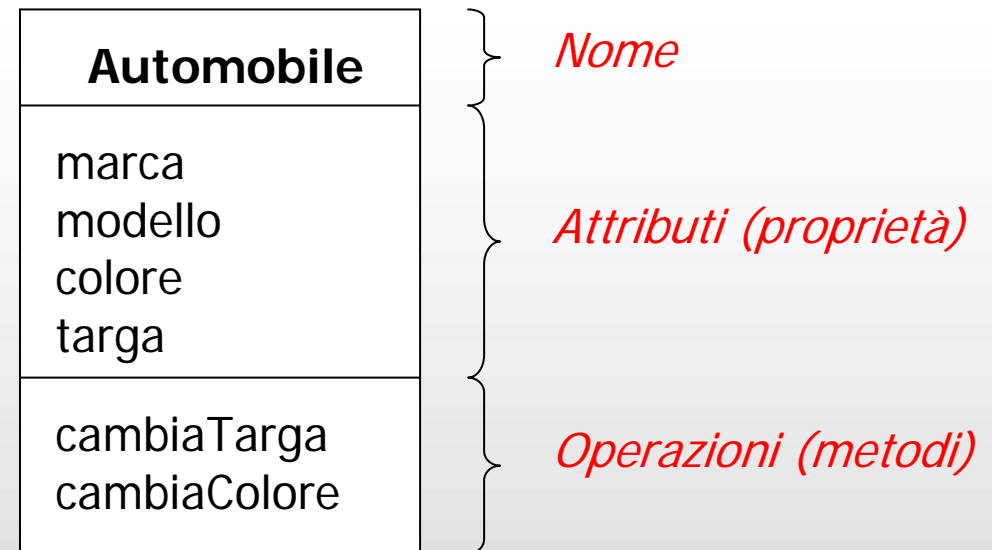


Diagramma delle classi: Associazioni

Relazioni tra classi

- **Associazione:** correlazione fra classi; nel diagramma è una linea continua fra due classi, con esplicita semantica nei due sensi
 - Aggregazione
 - Composizione
- **Molteplicità:** numero di oggetti che partecipano all'associazione. Esempi di molteplicità sono:

1	Esattamente una istanza
0..*	Nessun limite al numero di istanze
1..*	Almeno una istanza
n..m	Da n a m istanze

Diagramma delle classi: esempio associazione

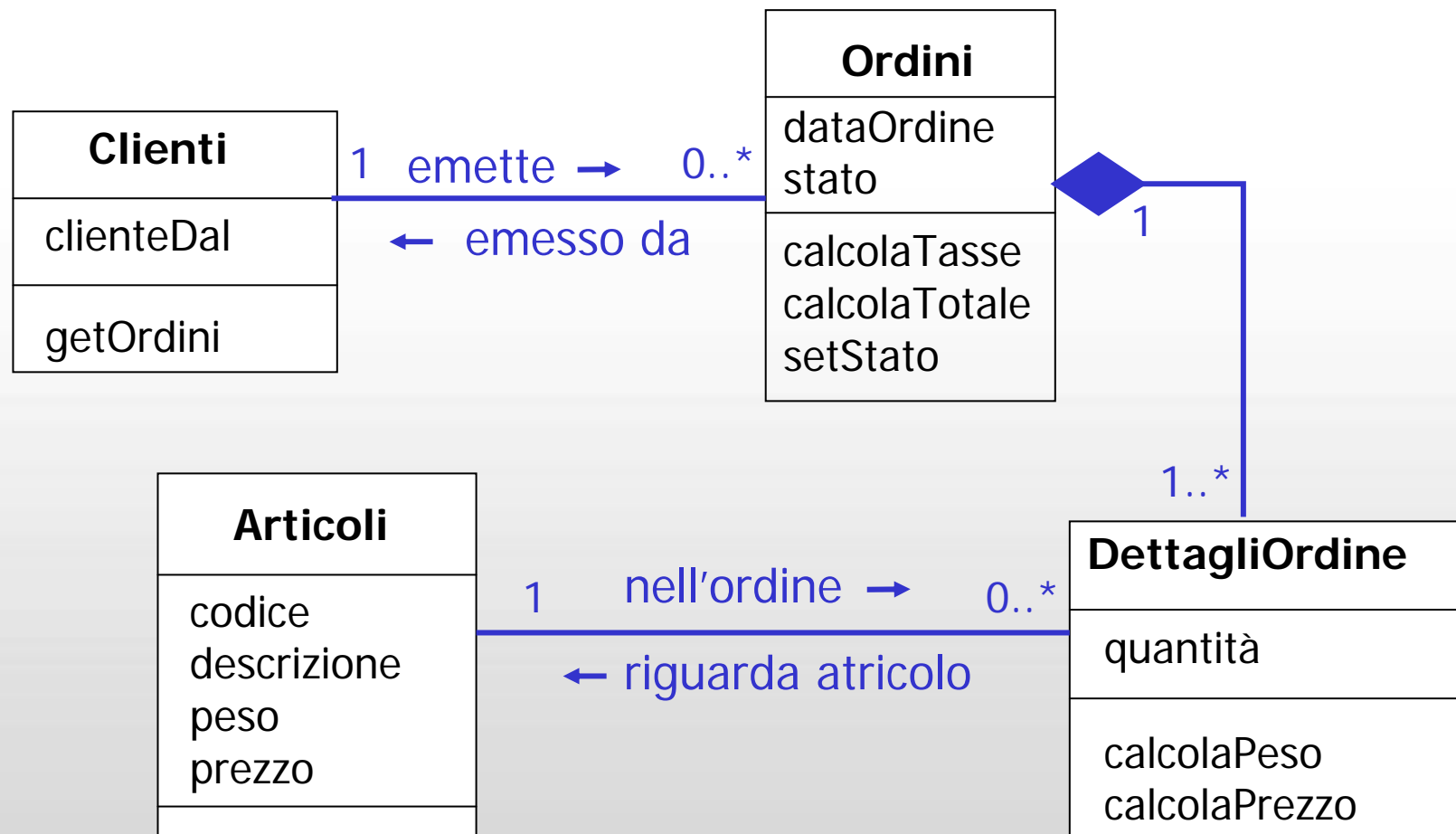


Diagramma delle classi: aggregazione/composizione

Relazioni tra classi

▪ **Aggregazione**

- esprime il concetto “è *parte di*”

▪ **Composizione**

- la parte (componente) non può esistere da sola, cioè senza la classe composto
- una componente appartiene ad un solo composto

Diagramma delle classi: Esempio di aggregazione/composizione

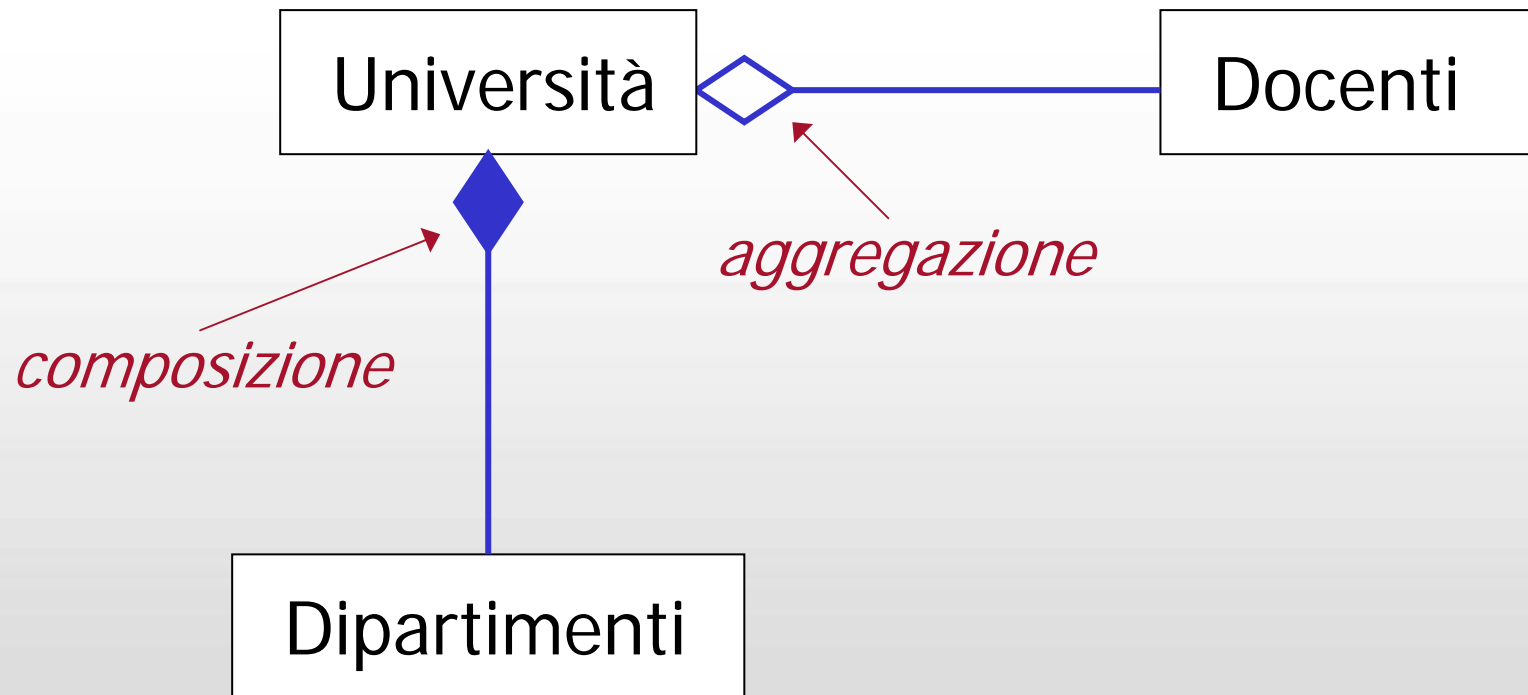
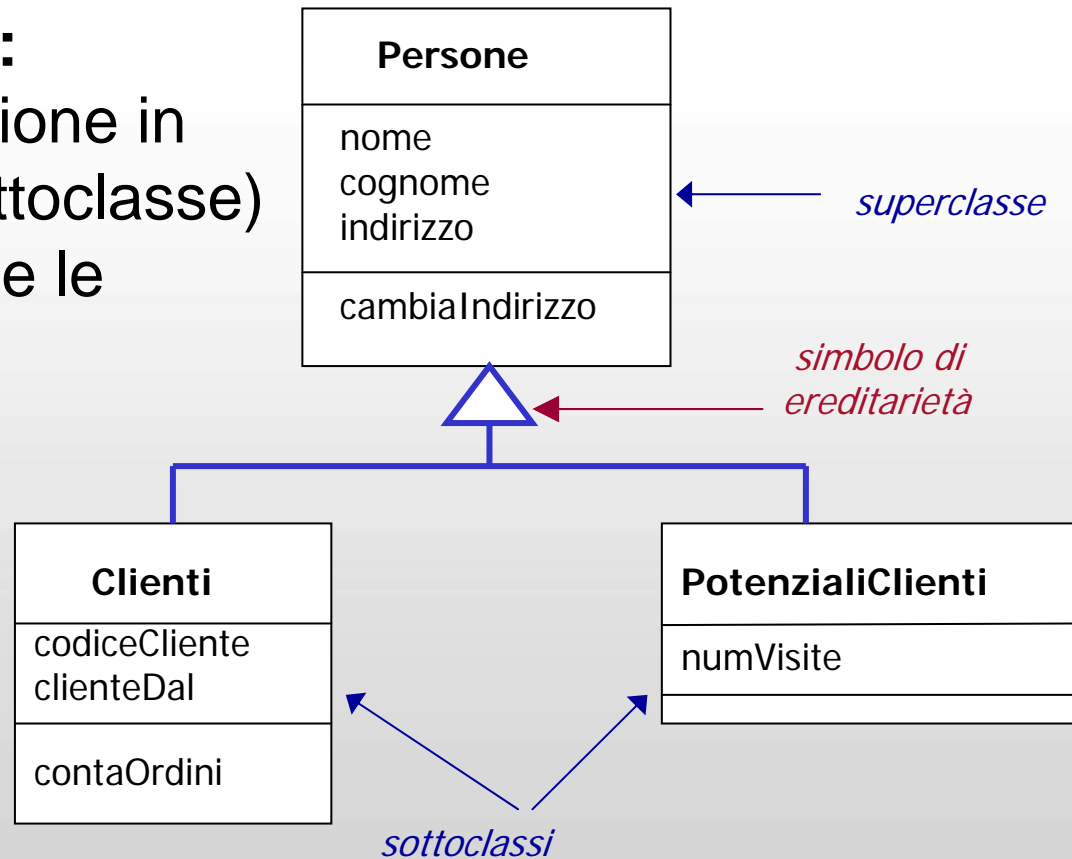


Diagramma delle classi: generalizzazione

Relazioni tra classi

- **Generalizzazione:**
(ereditarietà) relazione in cui una classe (sottoclasse) eredita gli attributi e le operazioni di una superclasse

- multipla
- semplice



Diagrammi dinamici

- Interaction Diagrams
 - come interagiscono gli Attori e gli oggetti per realizzare uno Use Case

- Due modalità
 - Diagrammi di sequenza
 - Diagrammi di collaborazione

Diagramma di sequenza

Visione temporale

- Specifica come gli oggetti interagiscono
 - sequenza temporale dei messaggi scambiati
- Non si vedono le associazioni tra oggetti
- Possono corrispondere a uno scenario specifico o a un intero caso d'uso
- Il diagramma ha due dimensioni
 - sull'asse orizzontale sono rappresentati gli oggetti che interagiscono
 - sull'asse verticale la sequenza temporale dei messaggi

Oggetti del diagramma di sequenza

Elementi

▪ **Oggetto**

- un box in alto con un nome sottolineato e una linea tratteggiata verticale detta linea di vita dell'oggetto
- Il tempo scorre dall'alto verso il basso

▪ **Messaggio**

- freccia con un nome
- Sincroni o asincroni

▪ **Chiamata interna (self-call):**

- per chiamare una delle sue funzioni

Diagramma di sequenza

Elementi

- **Condizione:** è associata ad un messaggio: solo se è soddisfatta il messaggio viene generato [ha disponib]
- **Iterazione:** indica che un messaggio viene inoltrato più volte ad oggetti diversi di uno stesso gruppo
- **Ritorno:** indica il valore restituito all'oggetto chiamante e non un nuovo messaggio; può essere omesso ← - - - - .
- **Distruzione:** deallocazione di un oggetto come metodo interno o tramite messaggio da un altro oggetto X

Diagramma di sequenza

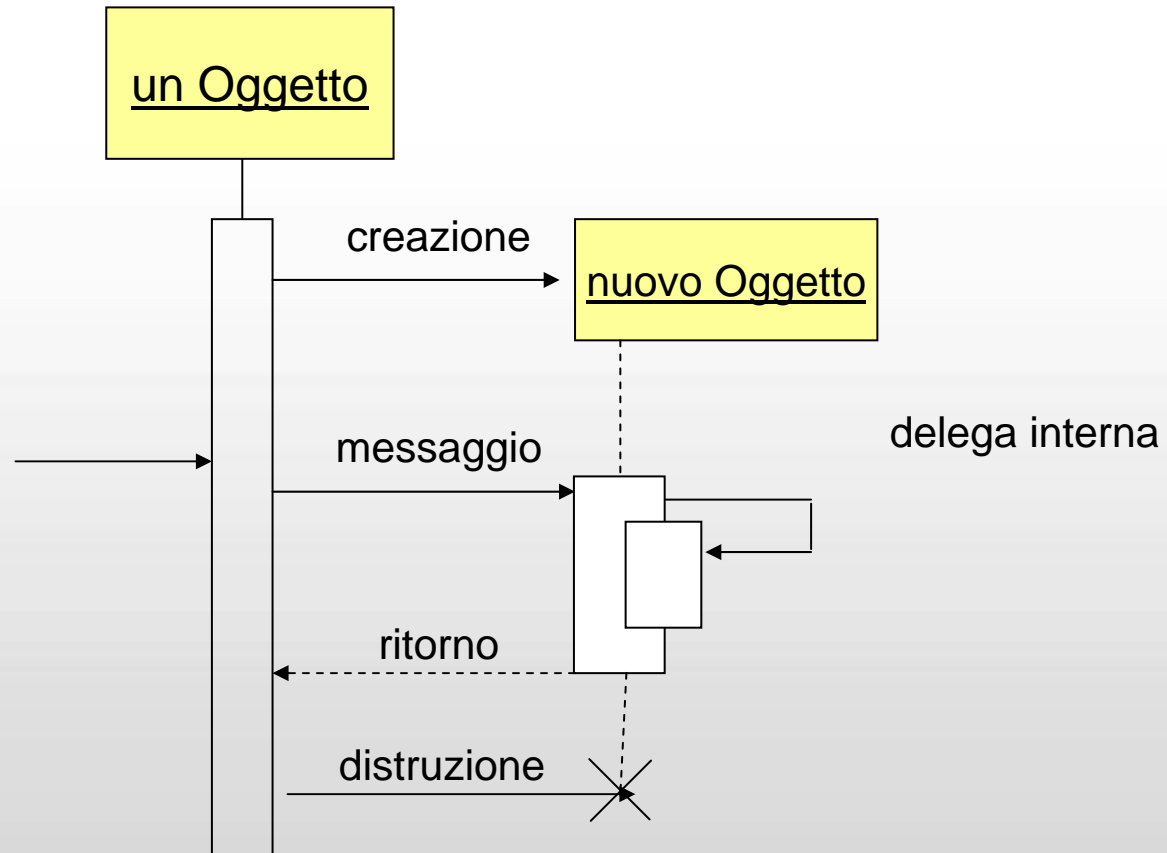


Diagramma di collaborazione

I collegamenti dinamici tra gli oggetti

- Evidenzia la collaborazione fra gli oggetti
 - sono evidenti i legami tra gli oggetti.
- I componenti del diagramma sono sempre oggetti e messaggi
 - non è modellato lo scorrere del tempo
- I messaggi sono numerati sequenzialmente
- I messaggi sono accompagnati da frecce

Diagramma di collaborazione

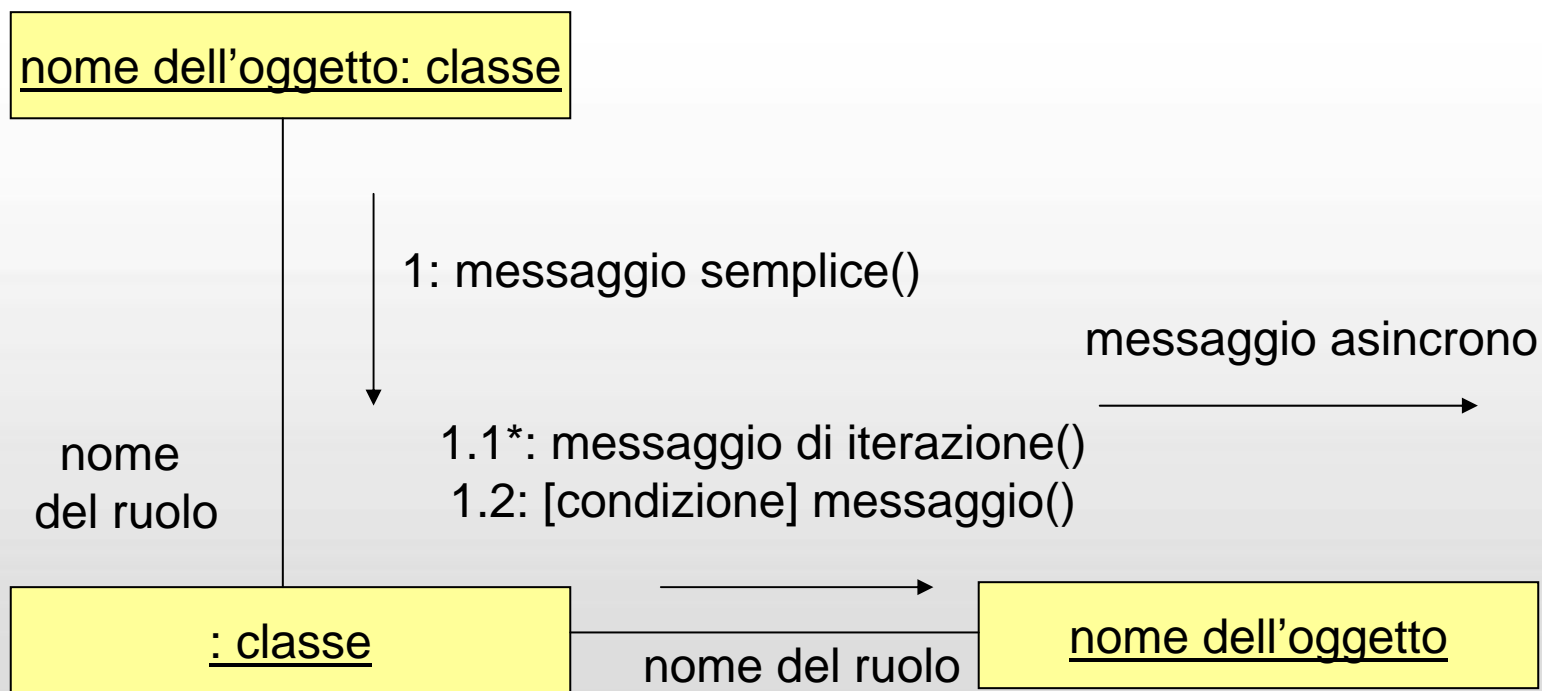


Diagramma delle transazioni di stato

La vita di un oggetto

- specifica il ciclo di vita degli oggetti di una classe, definendo le regole che lo governano
- quando un oggetto si trova in un certo stato può essere interessato da determinati eventi
- come risultato di un evento l'oggetto può passare ad un nuovo stato (transizione)
- Uno **stato** è una situazione durante la vita di un oggetto nella quale
 - l'oggetto soddisfa qualche condizione
 - esegue qualche attività
 - aspetta per qualche evento

Diagramma di transazioni di stato

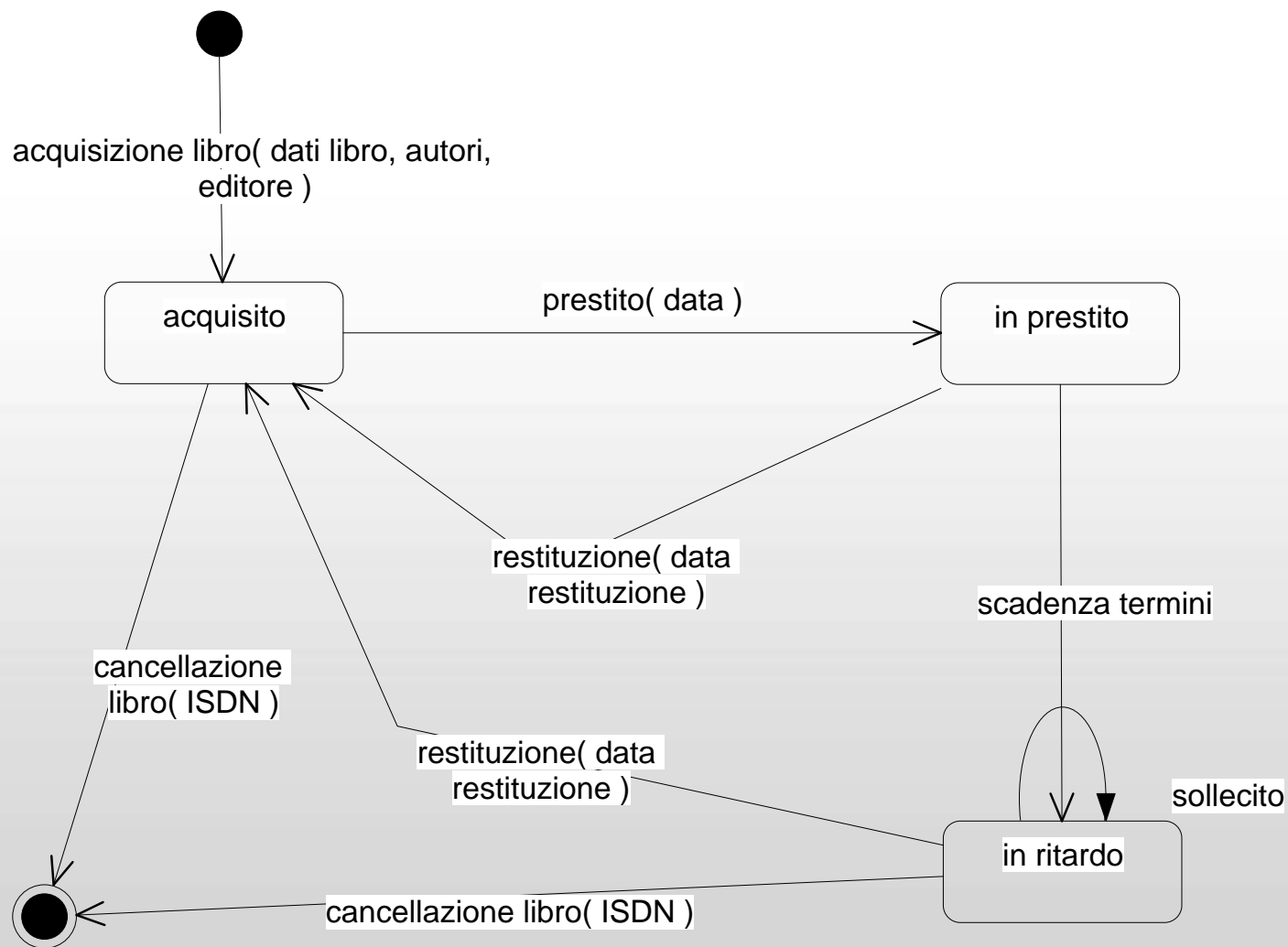


Diagramma delle attività

Descrizione dei Workflow

- Servono per la modellazione di workflow
- I componenti principali sono:
 - **Attività:** un processo del mondo reale (es. compilare un ordine) o l'esecuzione di una procedura software (es. metodo di una classe)
 - **Transazioni:** sono innescate semplicemente dal termine dell'attività precedente
- Il diagramma consiste in una sequenza di attività e supporta l'esecuzione di cicli, l'esecuzione parallela e quella condizionale

Diagramma delle attività: Branch & Merge

Diramazioni e giunzioni

- **Branch:** diramazione con una transizione entrante e più di una uscente, con condizioni mutuamente esclusive
- **Merge:** giunzione con più transizioni entranti ed una sola uscente
- **Join:** unione con più transizioni entranti ed una sola uscente che può scattare solo dopo che sono terminate le attività degli stati corrispondenti alle transizioni entranti
- **Fork:** divisione con una transizione entrante e più di una uscente che si eseguono in parallelo

Diagramma delle attività

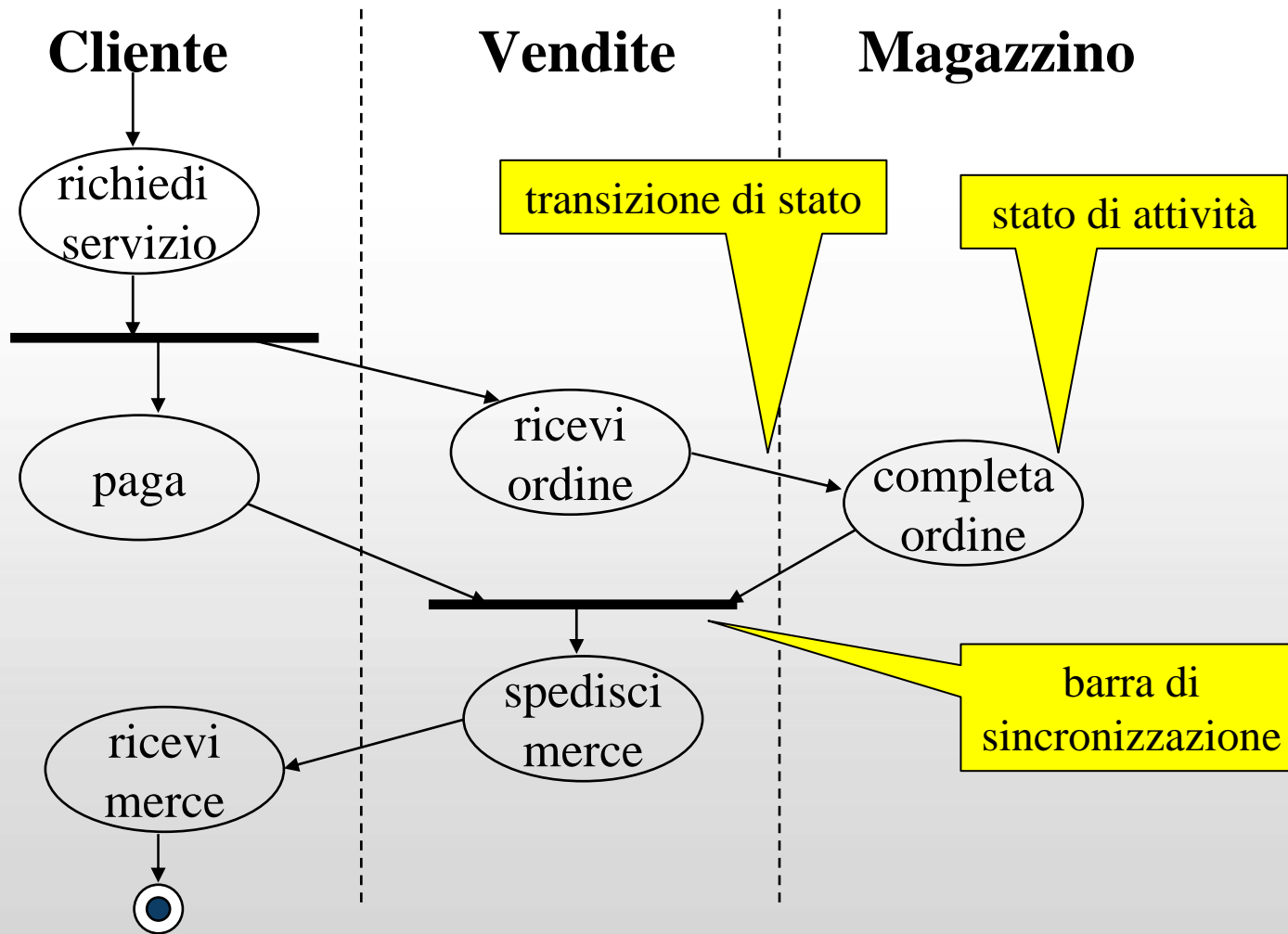


Diagramma delle componenti

Suddivisione del sistema

- Illustra i componenti di un sistema e le relative dipendenze.
 - i componenti sono moduli software eseguibili
 - i componenti possono essere raggruppati in package
- **Package:** Raggruppamento concettuale di elementi del modello
- **Componente:** Raggruppamento di elementi fisici del sistema.
- Mostra come i cambiamenti apportati ad un componente si ripercuotono sugli altri. Esistono dipendenze:
 - di comunicazione
 - di compilazione

Diagramma delle componenti

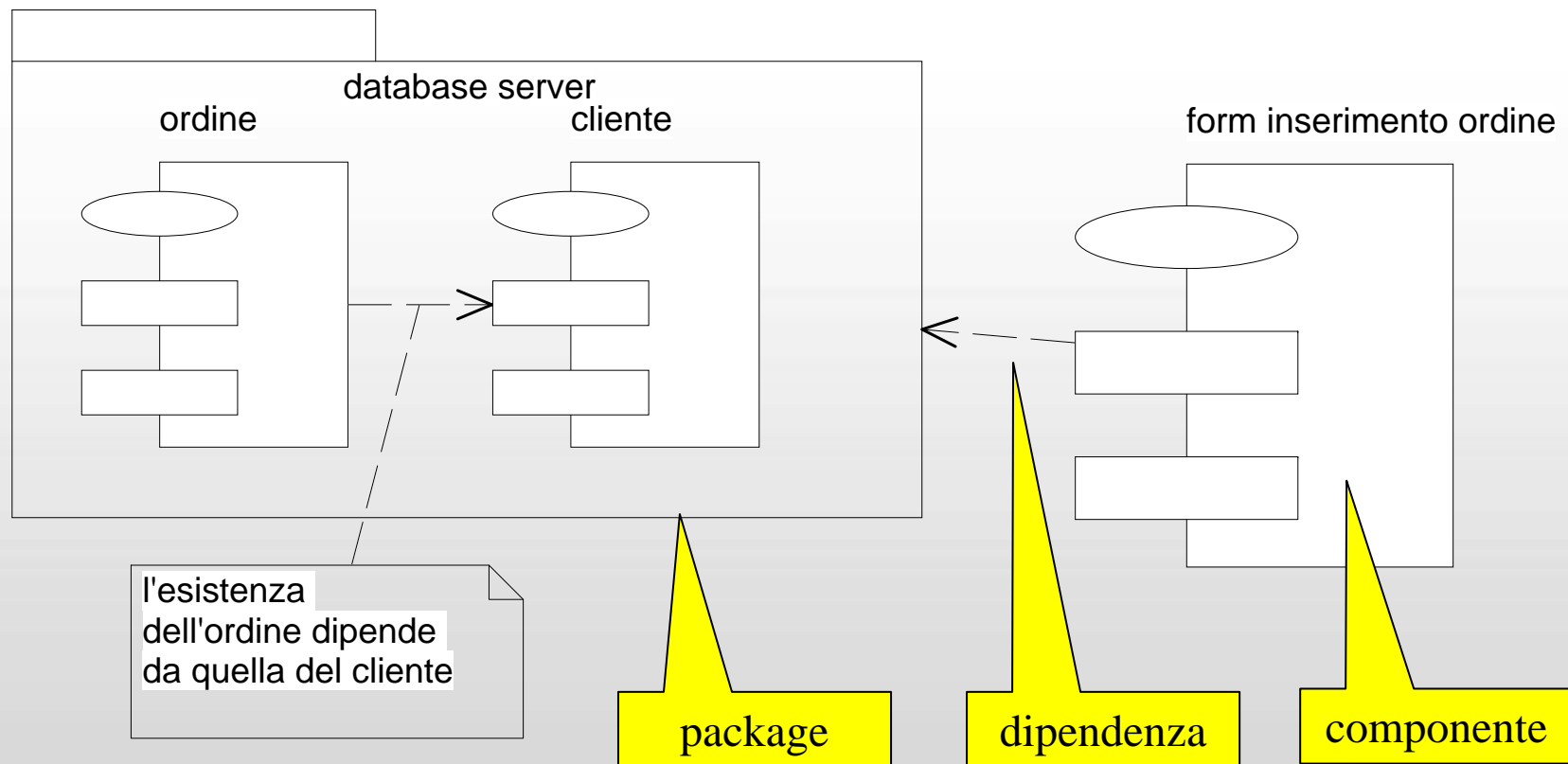
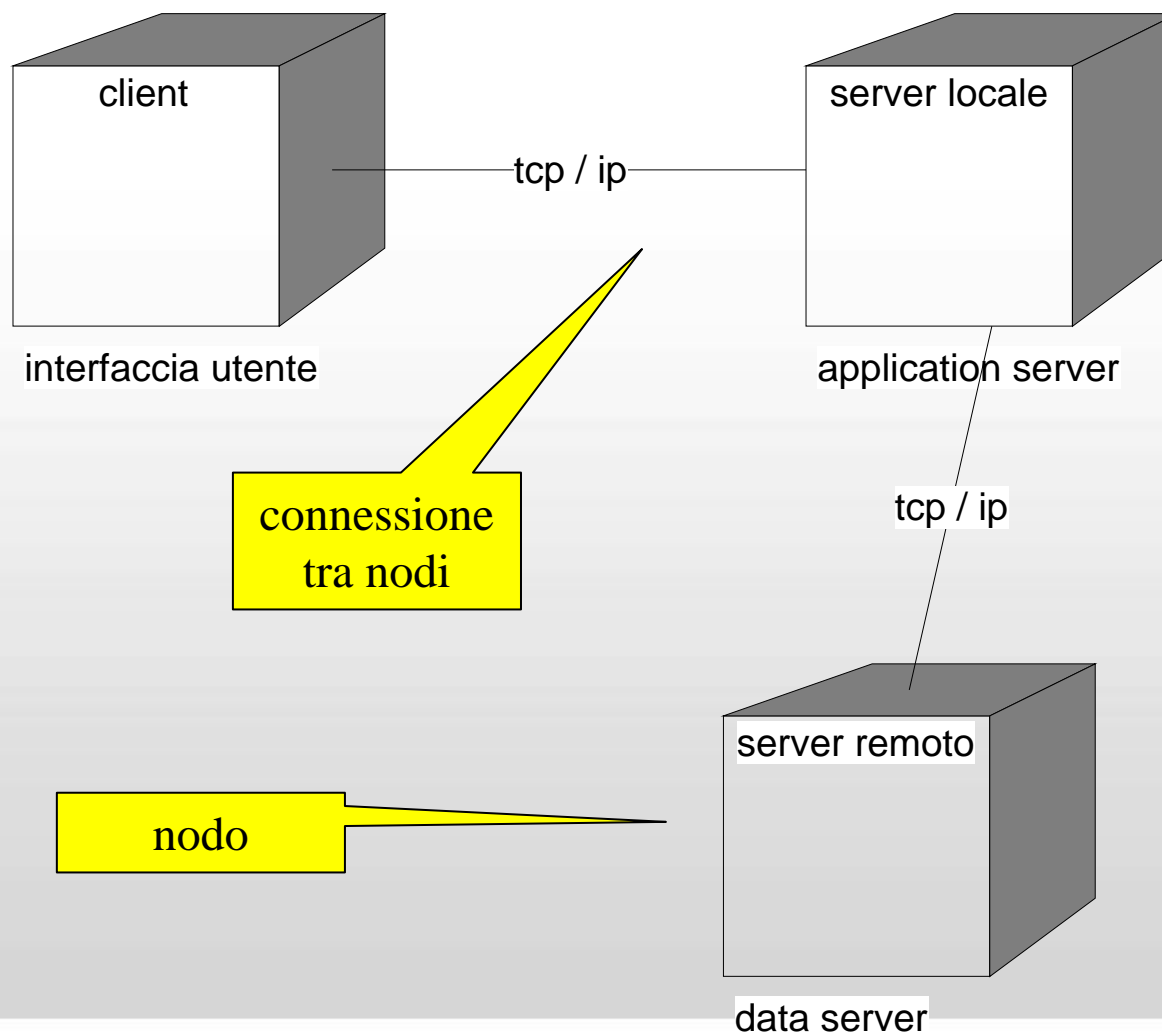


Diagramma di deployment

Dispiegamento fisico del sistema

- permette di rappresentare l'architettura fisica del sistema
- Mostra le relazioni fisiche tra i componenti software ed hardware del sistema finito.
- Le unità computazionali sono rappresentate come nodi
- Le associazioni tra nodi rappresentano le connessioni fisiche usate dai componenti del sistema per interagire

Diagramma di deployment





Ora comandiamo noi!

I linguaggi di programmazione

Linguaggi di programmazione

Cos'è

- Un linguaggio di programmazione è costituito:
 - Da un alfabeto
 - Da parole chiave (il vocabolario)
 - Da regole sintattiche (la grammatica)

A che serve

- Un programma descrive al computer, in estremo dettaglio, la sequenza di passi necessari per un particolare compito.
- Un programma implementa un algoritmo
 - Senza algoritmo non ci può essere un programma

Linguaggi di programmazione

Linguaggio macchina (Assembler)

- I microprocessori riconoscono ed eseguono un insieme ristretto di istruzioni
 - Codificate
 - Poco intuitive
- Codifica complicata
 - Sequenze di bit ottenute dall'assembler
 - Programma molto efficiente
 - Programma molto lungo
 - Linguaggio poco potente
- Serve una conoscenza del calcolatore
 - Uno per ogni calcolatore
 - Programmi non portabili (da riscrivere)

Linguaggi di programmazione

Linguaggi di alto livello

- Si slegano dal funzionamento della macchina
 - Pascal, Fortran, Cobol, C, Perl, Java, Javascript, ASP, php, python
 - Da regole sintattiche (la grammatica)
- Linguaggio compilato
 - Tradurre in linguaggio macchina il codice di alto livello prima della sua esecuzione
 - Non serve il compilatore durante l'esecuzione
- Linguaggio interpretato
 - Tradurre in linguaggio macchina il codice di alto livello durante la sua esecuzione
- Compilatori ed interpreti valgono per ogni diverso tipo di macchina
 - La portabilità vale solo per il codice sorgente (non per l'eseguibile)
- La virtual machine di Java va oltre

Errori di programmazione

- Sintattici
 - System. **aut** .println("Hello, World!");
 - System.out.println("Hello, World !) ;
 - Ce lo segnala il compilatore 😊
- Errori logici
 - System.out.println("He llo, World !) ;
 - Ce lo segnala il nostro cliente ☹

Linguaggi di programmazione

Elementi di un programma

- Due componenti:
 - Logica di business: processamento dei dati secondo le diverse necessità
 - Interfaccia utente
- Nella *programmazione web* i due elementi sono fisicamente divisi:
 - L'interfaccia utente risiede sul computer client;
 - La logica applicativa risiede sul computer server;
 - Possono essere scritti in linguaggi completamente differenti
 - linguaggi client-side per l'interfaccia utente
 - HTML, CSS, JavaScript e non solo
 - linguaggi server-side per la logica applicativa
 - ASP, JSP, PHP



Generalizziamo

XML

Markup language

`Markup`

- È un linguaggio basato su testo (interoperabilità)
- Contiene anche i metadati che danno istruzioni su se stesso

SGML

- Standard Generalized Markup Language
 - Generale e complesso
- HTML: HyperText Markup Language
 - Presentazione visualizzazione di contenuti
 - collegamento tra pezzi di informazione (ipertesto con ancore)
 - Un linguaggio di successo
 - Il TAG è intuitivo e accettato tra gli sviluppatori
 - La semplicità come punto di forza

eXtensible Markup Language

- Sviluppato dal World Wide Web Consortium (<http://www.w3c.org>)
 - Versione semplificata del SGML
 - pensato per descrivere qualsiasi tipo di dati basato sul markup
 - Ambiti diversi dalla presentazione di ipertesti
 - Per descrivere dati generici
 - È un formato di interscambio di dati tra applicazioni diverse

Da HTML ad XML a XHTML

XML come metalinguaggio

- I tag XML non sono predefiniti
- XML fornisce le regole sintattiche per costruire particolari linguaggi (applicazioni XML)
 - Definire il 'linguaggio' e/o il 'vocabolario' per descrivere potenzialmente ogni cosa
 - ad esempio i **nomi di persona** ... in XML:

```
<name>  
  <first>Giulio</first>  
  <last>Cesare</last>  
</name>
```
 - XHTML come evoluzione dell' HTML basato sul XML
 - Vocabolario: <table>, <body>, <p>, etc

Flessibilità, interpretabilità e trasformabilità del XML

Flessibilità

- Diverse rappresentazioni degli stessi dati

```
<name>Giulio Cesare</name>
.. oppure ...
<name>
  <first>Giulio</first>
  <last>Cesare</last>
</name>
```

Interpretabilità

- Il PARSER può interpretare qualsiasi documento XML
 - Interpreta la sintassi XML
 - Estrae le informazioni per l'applicazione

Flessibilità, interpretabilità e trasformabilità del XML

Trasformabilità

- Per favorire l'interoperabilità tra applicazioni servono
 - Vocabolari
 - regole standard (Schemi)
 - SVG, MathXML, XHMTL
- Come si passa da uno schema all'altro?
 - **XSLT** è il linguaggio per definire una mappatura tra schemi diversi

XML ben formato e valido

Well formed

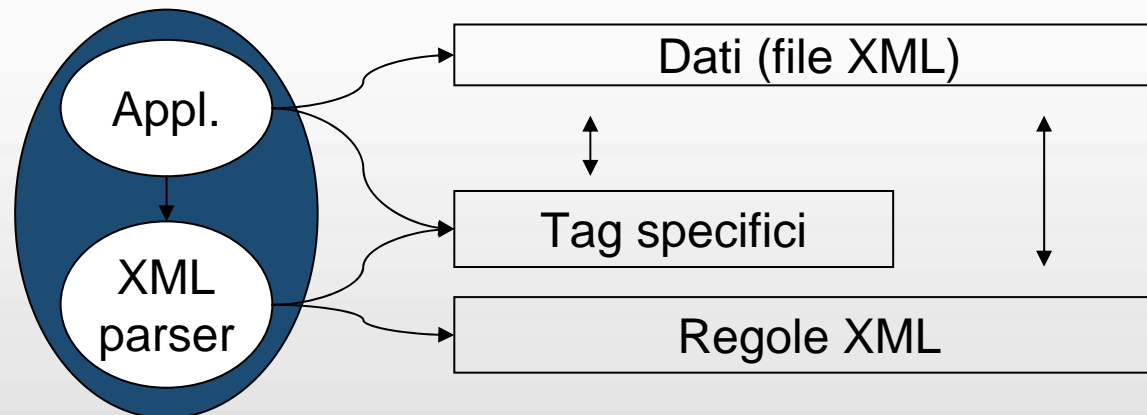
- Ogni documento XML deve essere ben formato
 - Rispettare la sintassi XML
- Un documento XML ben formato non richiede la presenza di un DTD

Valido

- Un documento XML è valido se
 - si riferisce a una DTD esplicita mediante una [Doctype declaration](#)
 - ne soddisfa i vincoli strutturali (nome, sequenza occorrenze ed attributi degli elementi)
- Il controllo di validità lo effettua il [parser](#)

XML, ovvero?

Il mondo XML



XML, ovvero?

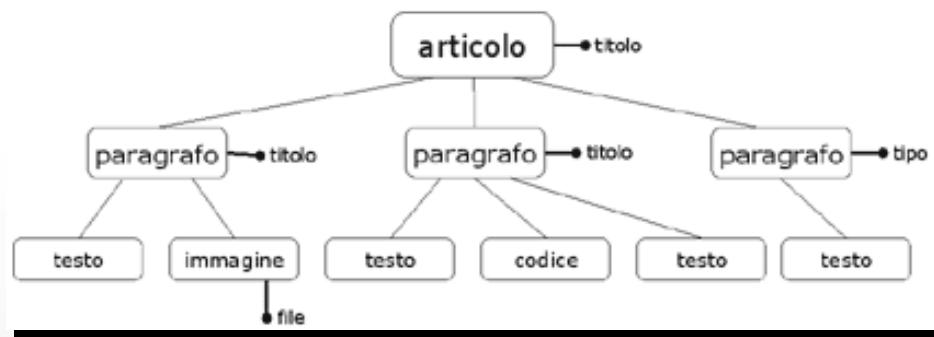
Struttura gerarchica ad Albero

- Tipi di relazioni:
 - padre-figlio
 - fratello-fratello

```
<?xml version="1.0" ?>
<articolo titolo="Titolo dell'articolo">
  <paragrafo titolo="Titolo del primo paragrafo">
    <testo>
      Blocco di testo del primo paragrafo
    </testo>
    <immagine file="immagine1.jpg">
    </immagine>
  </paragrafo>
  <paragrafo titolo="Titolo del secondo paragrafo">
    <testo>
      Blocco di testo del secondo paragrafo
    </testo>
    <codice>
      Esempio di codice
    </codice>
    <testo>
      Altro blocco di testo
    </testo>
  </paragrafo>
  <paragrafo tipo="bibliografia">
    <testo>
      Riferimento ad un articolo
    </testo>
  </paragrafo>
</articolo>
```

XML, ovvero?

Struttura gerarchica ad Albero



```
<?xml version="1.0" ?>
<articolo titolo="Titolo dell'articolo">
  <paragrafo titolo="Titolo del primo paragrafo">
    <testo>
      Blocco di testo del primo paragrafo
    </testo>
    <immagine file="immagine1.jpg">
    </immagine>
  </paragrafo>
  <paragrafo titolo="Titolo del secondo paragrafo">
    <testo>
      Blocco di testo del secondo paragrafo
    </testo>
    <codice>
      Esempio di codice
    </codice>
    <testo>
      Altro blocco di testo
    </testo>
  </paragrafo>
  <paragrafo tipo="bibliografia">
    <testo>
      Riferimento ad un articolo
    </testo>
  </paragrafo>
</articolo>
```

Declaration

```
<? xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

- **Version** –versione delle specifiche XML a cui il documento è conforme
- **Encoding** –tipo di codifica utilizzata per i caratteri
- **Standalone** – indica se il documento si basa su altri documenti (ed esempio un DTD)

Body

```
<note id="12345">  
  <to>Luca</to>  
  <from>Carlo</from>  
  <title>Appuntamento</title>  
  <message>Ricordati la riunione di oggi</message>  
</note>
```

The diagram illustrates the structure of the XML body with colored boxes and arrows pointing to specific parts of the code:

- attribute** (blue box) points to the `id="12345"` attribute in the opening `<note>` tag.
- element** (green box) points to the `<to>` opening tag.
- PCDATA** (orange box) points to the text `Appuntamento` inside the `<title>` tag.
- tag** (purple box) points to the closing `</note>` tag.

Elementi

- Tutto ciò che è compreso tra il tag di apertura (incluso) ed il corrispettivo tag di chiusura (incluso)
- Il contenuto dell'elemento può essere:
 - Element content: altri elementi
 - Simple content: semplice testo
 - Mixed content: testo e altri elementi
 - Empty content: vuoto
- Entità radice

Attributi vs Elementi

Uguali o diversi

<u>Sottoelementi</u>	<u>Attributi</u>
<pre><note> <to>Luca</to> <from>Carlo</from> <title>Appuntamento</title> <message>...</message> </note></pre>	<pre><note title="Appuntamento"> <to>Luca</to> <from>Carlo</from> <message>...</message> </note></pre>

- La scelta è soggettiva, ma non sempre equivalente
- Infatti gli attributi:
 - Non possono contenere valori multipli
`<parent name="Luca"><child>Marco</child> <child>Mario</child></parent>`
 - Sono difficilmente espandibili (aggiunta di sottoelementi)
 - Non possono descrivere strutture
`<book><author><name>..</name><surname>..</surname></author></book>`

Un po' di sintassi

Regole

- Tutti i tag devono essere chiusi
- I tag devono essere correttamente innestati
- Gli attributi devono sempre essere inclusi tra apici singoli o doppi
- XML è case sensitive
- In XML gli spazi vengono preservati all'interno dei PCDATA
- I commenti sono inseriti tra i segni “<!--” e “-->”
 - Possono contenere sintassi XML che non verrà considerata da un parser
- Caratteri non validi:
 - Es: `<comparison> 6 is < 7 & 7 is >6 </comparison>`

&	→	&
<	→	<
>	→	>
'	→	'
&quite;	→	”

Un po' di sintassi

<![CDATA [.....]]>

- Usato per evitare errori di parsing quando il “contenuto” potrebbe essere interpretato come codice XML
- Anche mettendo del codice XML non viene elaborato dal parser XML, ma restituito all'utente

Istruzioni allo strato applicativo

- Non sono processate dal parser XML, ma sono per un'applicazione specifica
 - <?My_Application My_Application_data>
- Esempio
 - <?xml-stylesheet type="text/xsl" href="bpg4-0.xsl"?>
 - Applicazione target: xml-stylesheet
 - Dati per l'applicazione: type="text/xsl" href="bpg4-0.xsl"

XSD e DTD

- Linguaggi per definire le entità e le loro relazioni
 - DTD (Document Type Definition)
 - XSD (XML Schema Definition)
- Controllare la struttura dei dati
 - quanti figli può avere un padre?
 - che tipo di figli può avere un padre?
 - quanti figli sono ammessi per ogni tipo?
- Definire un vocabolario
 - Interoperabilità tra diverse entità:
 - Termini diversi per oggetti uguali
 - Termini uguali per oggetti diversi

Come è fatto

```
<?xml version="1.0"?>
```

```
<!DOCTYPE name SYSTEM "name.dtd">
```

```
<!DOCTYPE name [
```

```
  <!ELEMENT name (first, middle, last)>
```

```
  <!ELEMENT first (#PCDATA)>
```

```
  <!ELEMENT middle (#PCDATA)>
```

```
  <!ELEMENT last (#PCDATA)>
```

```
]>
```

```
<name>
```

```
  <first>John</first>
```

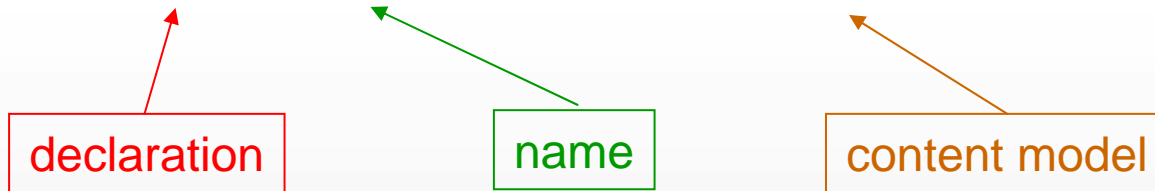
```
  <middle>Fitzgerald Johansen</middle>
```

```
  <last>Doe</last>
```

```
</name>
```

Sintassi ELEMENT

<!ELEMENT dinosaurs (carnivore, herbivore, omnivore)>



Name:

Il nome degli elementi va dichiarato dentro il DTD

Content:

- Empty
- Element
- Mixed
- Any

Sintassi ELEMENT

- Empty content:

<!ELEMENT br EMPTY>

L'elemento dovrà essere sempre vuoto!

- Element content:

<!ELEMENT carnivore (species, length, height, speed, discoverer)>

- un set di elementi permessi ed obbligatori
- Se un elemento *carnivoro* contiene solo il sottoelemento *species* **NON E' VALIDO** a meno che non aggiunga: <!ELEMENT carnivore (species)>
- Sono permesse due modalità:
 - **Sequences**
 - **Choices**

Sintassi ELEMENT

Sequences:

<!ELEMENT name (first, middle, last)>

- Ogni element nel documento XML deve avere **tutti i sottoelementi indicati nel giusto ordine**

Choices:

<!ELEMENT gender (male | female)>

- Ogni elemento nel documento XML può avere **l'uno OPPURE l'altro sottoelemento, ma mai nessuno o più di uno**

Combinate:

<!ELEMENT location (GPS | (country, region))>

Sintassi ELEMENT

Mixed content:

```
<!ELEMENT description (#PCDATA | i | b)*>
```

Any content:

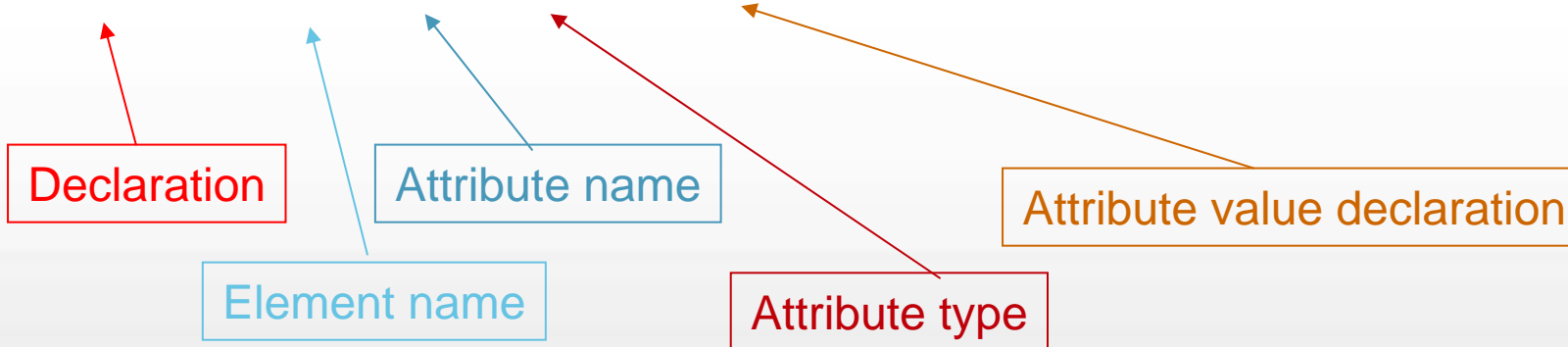
```
<!ELEMENT description ANY>
```

Cardinalità ELEMENT

- [none] l'elemento può apparire una e una sola volta
- ? l'elemento può apparire una volta o non apparire
- + l'elemento può apparire una o più volte
- * l'elemento può apparire zero, una o più volte

Sintassi ATTRIBUTE

`<!ATTLIST dinosaurs source CDATA #IMPLIED>`



`<!ATTLIST carnivore period (Triassic | Jurassic | Cretaceous) "Jurassic">`

`<!ATTLIST dinosaurs version CDATA #FIXED "1.0">`

`<!ATTLIST carnivore period (Triassic | Jurassic | Cretaceous) #REQUIRED>`

`<!ATTLIST weapon image-format (jpg | gif | bmp) #IMPLIED>`

I namespace

Tanti dizionari

- I termini di un vocabolario sono definiti attraverso uno schema
 - XMLSchema
 - DTD
- Ognuno descrive i propri dati (es. Ordini, prodotti) con un proprio vocabolario (product, id, customer ...)
- Come possiamo scrivere un documento XML che unisca dati di diversi domini (due aziende con diversi tipi di prodotti)?
- XML Namespaces
 - specifiche <http://www.w3.org/TR/REC-xml-names/>

```
<?xml version="1.0"?>
<person>
  <name>
    <title>Sir</title>
    <first>John</first>
    <middle>Fitzgerald Johansen</middle>
    <last>Doe</last>
  </name>
  <position>Vice President of Marketing</position>
  <résumé>
    <html>
      <head><title>Resume of John Doe</title></head>
      <body>
        <h1>John Doe</h1>
        <p>John's a great guy, you know?</p>
      </body>
    </html>
  </résumé>
</person>
```

I namespace

URI - Uniform Resource Identifier

- Usare i prefissi per riferire il vocabolario giusto per ogni termine
 - **html:title**,
 - **pers:title**
- Il prefisso deve riferire in maniera univoca il vocabolario
- Uniform Resource Identifier
 - Risorse raggiungibili: URL Uniform Resource Locator - *http://amazon.com*
 - Risorse non raggiungibili: URN Uniform Resource Name - *ISBN:837483748*
- URL: **<xmlns:pers=http://www.iuav.it/pers>**
 - **È un modo semplice**
 - **Non si può avere duplicazione**
 - **Un URL appartiene a chi mantiene il domain name**
 - **L'URL in generale contiene anche lo schema XML (o il DTD)**

I Namespace

Esempi

```
<?xml version="1.0"?>
<pers:person
  xmlns:pers="http://sernaferna.com/pers"
  xmlns:html="http://www.w3.org/1999/xhtml">
  <pers:name>
    <pers:title>Sir</pers:title>
    <pers:first>John</pers:first>
    <pers:middle>Fitzgerald Johansen</pers:middle>
    <pers:last>Doe</pers:last>
  </pers:name>
  <pers:position>
    Vice President of Marketing
  </pers:position>
  <pers:r sum >
    <html:html>
      <html:head>
        <html:title>Resume of John Doe</html:title>
      </html:head>
      <html:body>
        <html:h1>John Doe</html:h1>
        <html:p>John's a great guy, you know?</html:p>
      </html:body>
    </html:html>
  </pers:r sum >
</pers:person>
```

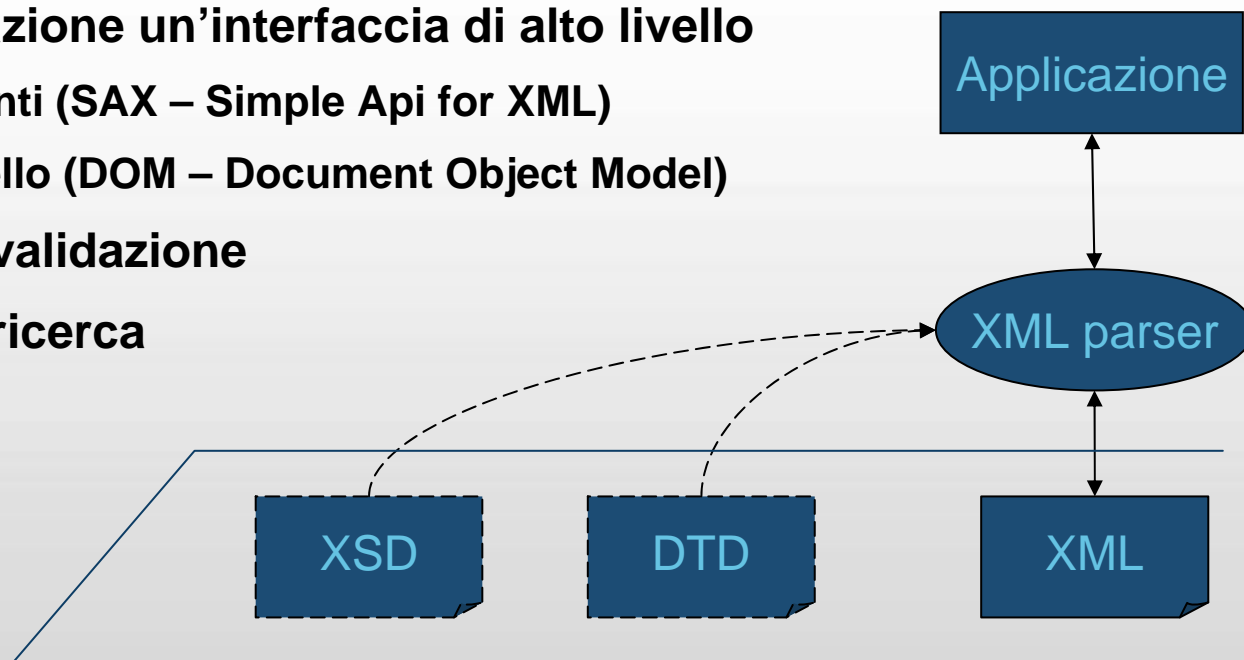
```
<?xml version="1.0"?>
<person xmlns="http://sernaferna.com/pers"
  xmlns:html="http://www.w3.org/1999/xhtml">
  <name>
    <title>Sir</title>
    <first>John</first>
    <middle>Fitzgerald Johansen</middle>
    <last>Doe</last>
  </name>
  <position>Vice President of Marketing</position>
  <r sum >
    <html:html>
      <html:head>
        <html:title>Resume of John Doe</html:title>
      </html:head>
      <html:body>
        <html:h1>John Doe</html:h1>
        <html:p>John's a great guy, you know?</html:p>
      </html:body>
    </html:html>
  </r sum >
</person>
```

```
<?xml version="1.0"?>
<person xmlns="http://sernaferna.com/pers">
  <name>
    <title>Sir</title>
    <first>John</first>
    <middle>Fitzgerald Johansen</middle>
    <last>Doe</last>
  </name>
  <position>Vice President of Marketing</position>
  <r sum >
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head><title>Resume of John Doe</title></head>
      <body>
        <h1>John Doe</h1>
        <p>John's a great guy, you know?</p>
      </body>
    </html>
  </r sum >
</person>
```

Il Parser

A cosa serve

- **Caricare il documento XML**
- **Memorizzare i dati in memoria**
- **Fornire all'applicazione un'interfaccia di alto livello**
 - Approccio ad Eventi (SAX – Simple Api for XML)
 - Approccio a Modello (DOM – Document Object Model)
- **Fornire servizi di validazione**
- **Fornire servizi di ricerca**

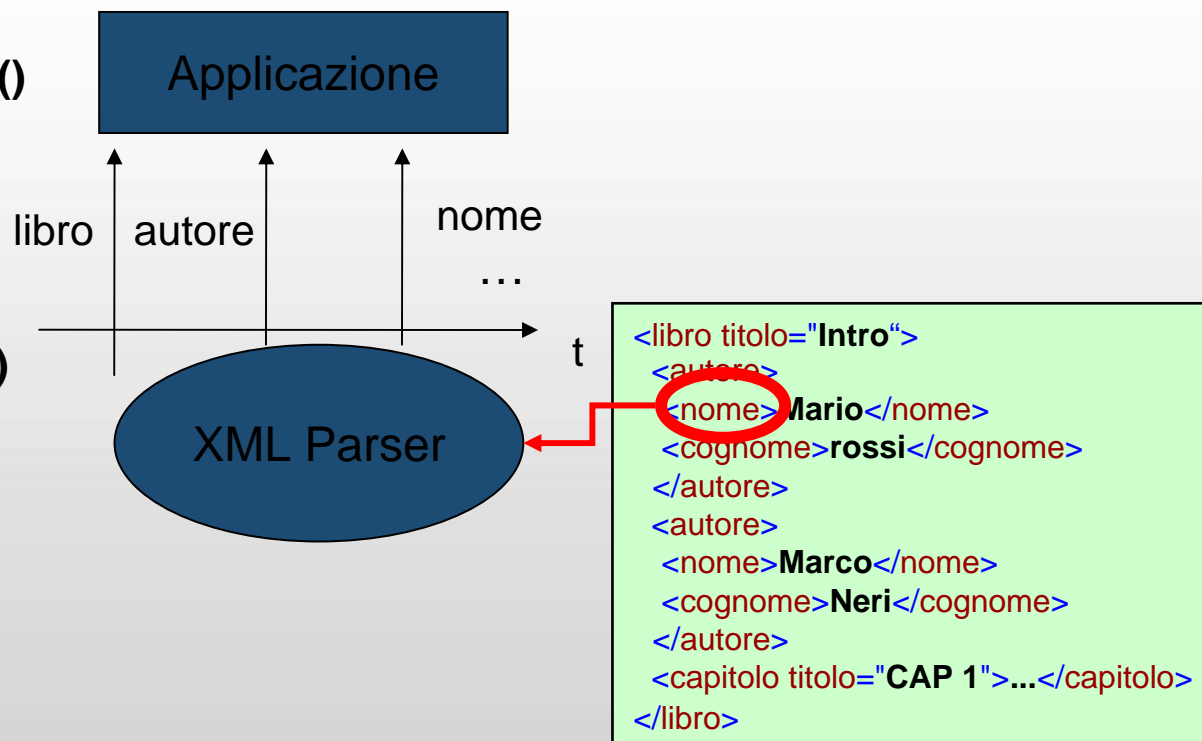


Il Parser – approccio ad eventi

Funzionamento

- Il parser scandisce l'intero file
- Per ogni evento gestito informa l'applicazione tramite la tecnica del Callback

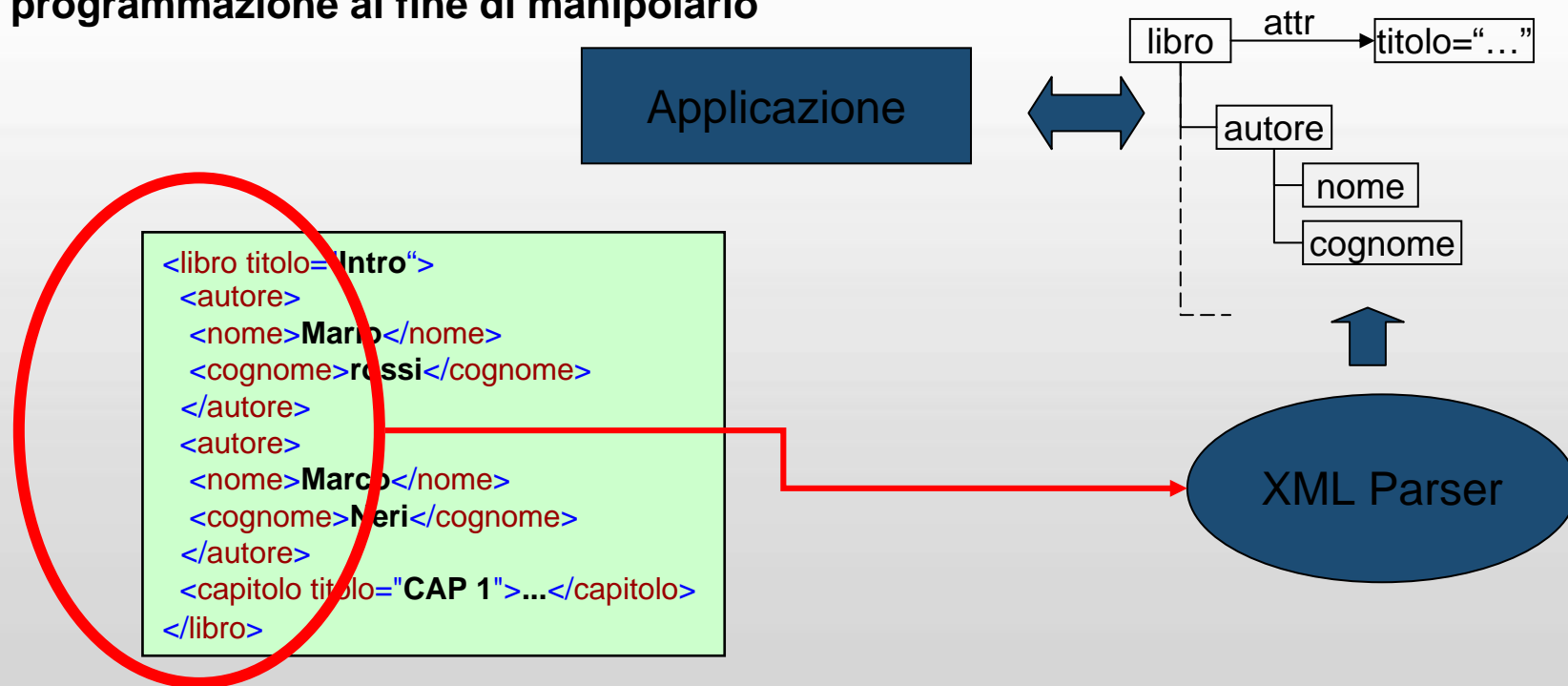
- startDocument()
- startElement()
- characters()
- endElement()
- endDocument()



Il Parser – approccio sul Modello

Funzionamento

- Il parser costruisce la struttura ad albero che rappresenta il documento
- Fornisce all'applicazione delle API per navigare l'albero e ritrovare i dati
- Attraverso il DOM il documento XML diventa accessibile per un linguaggio di programmazione al fine di manipolarlo



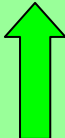

Il Parser – approccio sul Modello

I NODE

- **Il DOM presenta i documenti come una struttura gerarchica di oggetti di tipo Node.**
 - I Node possono avere zero o più nodi figli.
- **Le funzioni di Node per la manipolazione dei nodi figli sono**
 - `appendChild`,
 - `removeChild`,
 - `replaceChild`,
 - `insertBefore`.
- **La legalità di ciascuno di questi metodi dipende dal tipo effettivo del nodo.**
- **Nel caso l'operazione non sia disponibile (ad esempio, `appendChild` su un nodo `Text`), viene generata un'eccezione di tipo `DOMException`.**

Il Parser – i due approcci

Confronto

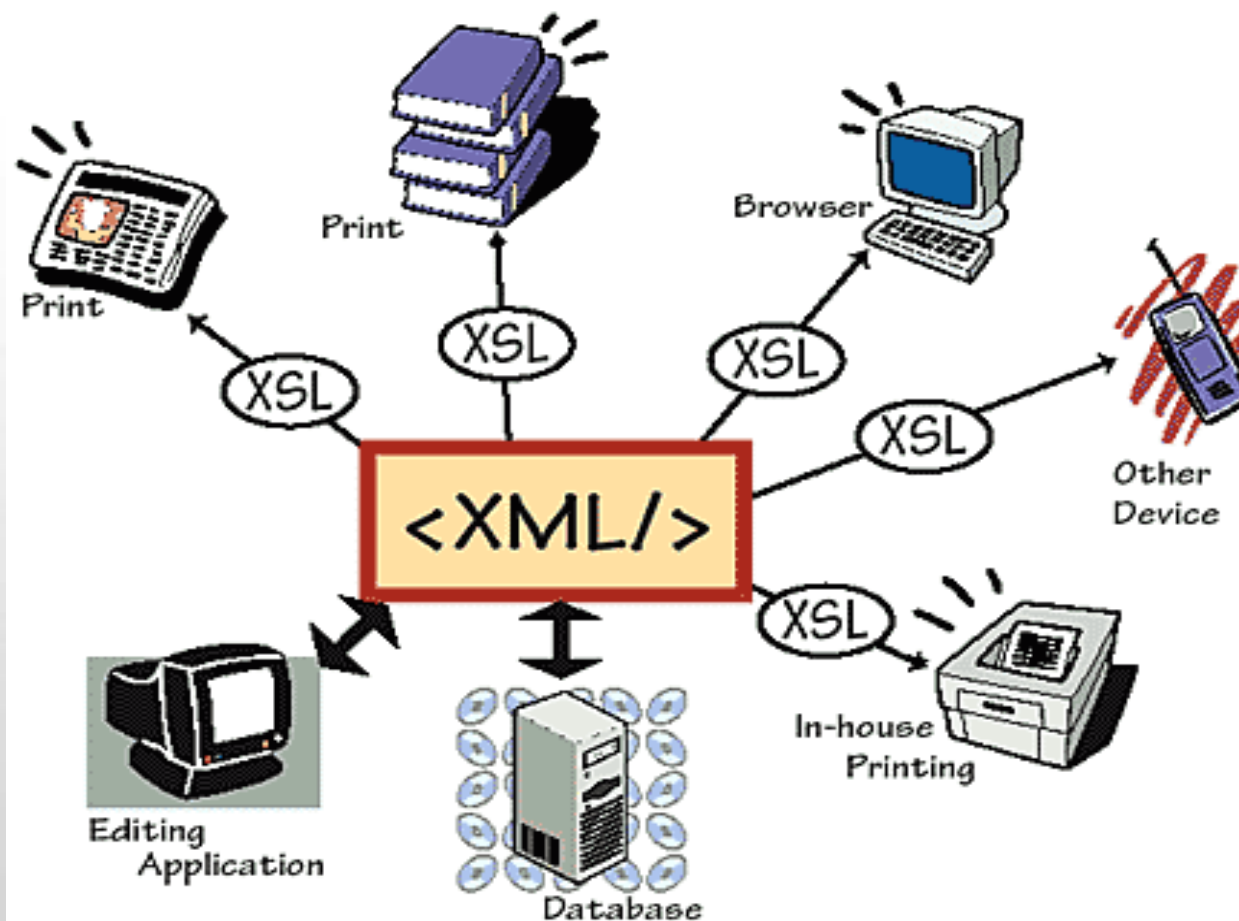
	Approccio ad eventi	Approccio del modello
Pro 	<ul style="list-style-type: none">▪ È “leggero”▪ È veloce▪ La ricerca nel documento si ferma dopo aver trovato l'informazione▪ Ideale per documenti grandi▪ Il programmatore implementa solo le funzionalità necessarie	<ul style="list-style-type: none">▪ fornisce all'applicazione un modello ricco del documento▪ mantiene una rappresentazione completa in memoria▪ Consente di modificare il modello
Contro 	<ul style="list-style-type: none">▪ Interfaccia troppo semplice richiede più codice nell'applicazione▪ Non si può modificare il documento▪ Nessun modo (API) per operare sul documento▪ Non c'è una versione per Browser, ma solo in linguaggi server-side	<ul style="list-style-type: none">▪ Richiede una occupazione di memoria per tutto il documento

Necessità

- Grazie a XML è possibile trasformare i documenti di dati XML in altri formati e strutture, come i file HyperText Markup Language (HTML).
- Lo strumento è l'Extensible Stylesheet Language Transformations (XSLT).
- Un insieme di regole che permettono di trasformare un documento in un altro documento
- XSL è una applicazione XML

XSLT

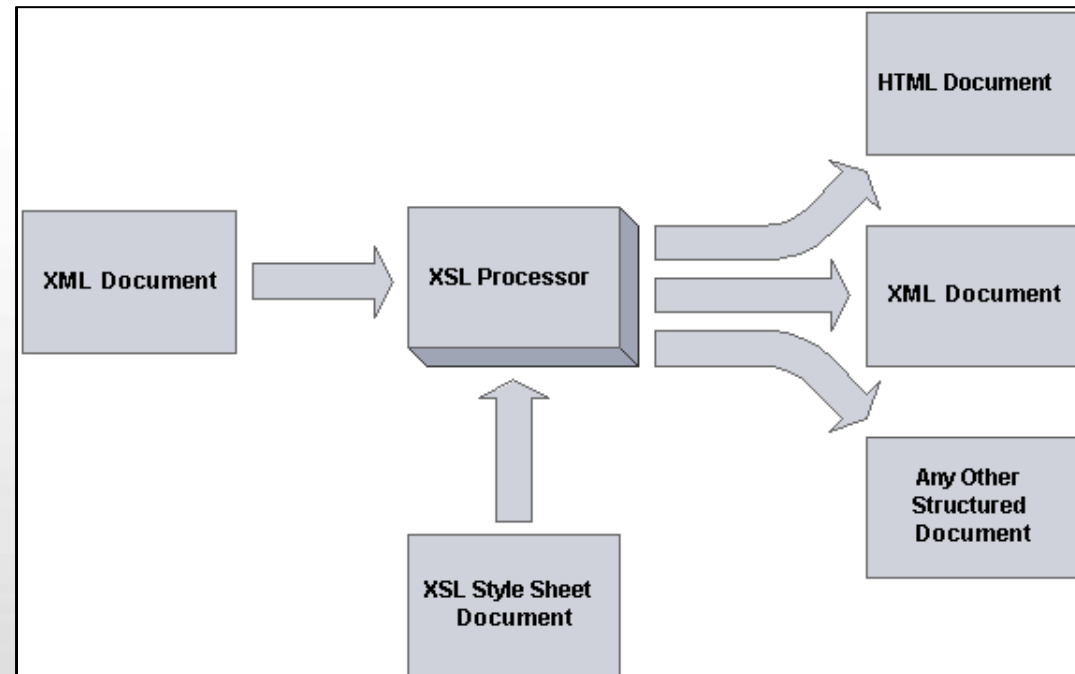
Necessità



Trasformazione XSLT

Processo

- Tre modi differenti
 - a livello server utilizzando script Java, ASP, PHP ecc. ;
 - a livello client: sul Browser che supporta questa tecnologia;
 - con un programma adeguato come XML Spy.



Namespace XSL

Essendo XML....

- XSL è una applicazione XML
- XSL definisce un proprio DTD e un Namespace
- L'URL da usare per il namespace è
`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
- Il namespace viene generalmente associato al prefisso 'xsl'
- L'elemento radice è `xsl:stylesheet`
 - All'interno dell'elemento `xsl:stylesheet` si mettono le regole di trasformazione
 - Il più semplice foglio di stile XSL non ha regole di trasformazione
 - Si preleva tutto il testo interno agli elementi del documento di input
 - Viene quindi riprodotto
 - Il risultato non è un documento XML

Esempio origine XML

```
<?xml version="1.0" encoding="UTF-8"?> <!-- Prologo XML -->
<?xml-stylesheet
  type="text/xsl" href="listacd_es1.xslt"?> <!-- Istruzione che
  indica il documento XSLT da associare -->
<listacd> <!-- Nodo Principale o Elemento Radice -->
  <artista>
    <nome>Stanley Jordan</nome>
    <albums>
      <album>
        <titolo>Magic Touch</titolo>
        <anno>1985</anno>
        <etichetta>Blue Note</etichetta>
      </album>
      <album>
        <titolo>Stolen Moments</titolo>
        <anno>1991</anno>
        <etichetta>Blue Note</etichetta>
      </album>
    </albums>
  </artista>
  <artista>
    <nome>Nick Drake</nome>
    <albums>
      <album>
        <titolo>Pink Moon</titolo>
        <anno>1972</anno>
        <etichetta>Island</etichetta>
      </album>
      <album>
        <titolo>Bryter Layter</titolo>
        <anno>1970</anno>
        <etichetta>Island</etichetta>
      </album>
      <album>
        <titolo>Five leaves left</titolo>
        <anno>1970</anno>
        <etichetta>Island</etichetta>
      </album>
    </albums>
  </artista>
  .....
</listacd>
```

Regole

■ Trasformazione vuota

```
<?xml version="1.0" encoding="UTF-8"?>  <!-- Prologo XML -->  
<xsl:stylesheet version="1.0"  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
<!-- Dichiarazione del documento XSLT -->  
</xsl:stylesheet>
```

■ Risultato trasformazione vuota

Stanley Jordan Magic Touch 1985 Blue Note
Stolen Moments 1991 Blue Note
Nick Drake Pink Moon 1972
Island
Bryter Layter 1970
Island

Regole

- Attraverso il modello (elemento `xsl:template`) e l'attributo `match` si possono effettuare le trasformazioni
 - `xsl:template` definisce una regola di trasformazione
 - `match` identifica il tipo di input che attiva la regola

```
<xsl:template match="/">  <!-- Applica questo template al nodo radice indicato dal carattere / -->
  <html>
    <xsl:apply-templates>  <!-- Richiama e applica gli altri templates -->
  </xsl:apply-templates>
</html>
</xsl:template>
<xsl:template match="artista">  <!-- Quando trova un nodo artista applica questa regola -->
  <xsl:value-of select="nome">
  </xsl:value-of>
  <br></br>
</xsl:template>
```

- Output

```
<html>
  Stanley Jordan
  <br>
  Nick Drake
  <br>
</html>
```

Regole

- L'attributo `select` dell'elemento `xsl:value-of` può assumere i seguenti valori:
 - il nome del TAG dell'elemento
 - il carattere `.` (punto) che prende il valore dell'elemento corrente
 - `@nome_attributo` per prendere il valore dell'attributo dell'elemento puntato
 - `text()` prende il testo contenuto nell'elemento corrente
 - `comment()` per prendere il commento dell'elemento corrente
- In generale il valore di `select` può essere una espressione XPath
 - `.` valore del nodo corrente
 - `..` risale al genitore dell'elemento corrente
 - `*` tutti e solo gli elementi figli del nodo corrente
 - `//x` tutti gli elementi `<x>` indipendentemente dalla loro profondità

Regole

- Altre informazioni che si possono ottenere sono
 - position() valore della posizione del nodo corrente
 - name() nome del nodo
 - count() numero dei nodi all'interno di un elemento specificato
- Operatori logici
 - !=, =, <, >, >=, <=, and, or,
- Operatori matematici
 - +, *, div, mod, round(), ceiling(), floor(), sum()

Regole

- Esempio

```
<td align="right">  
  <xsl:value-of select="sum(albums//durata)"/>  
  <!-- Somma della durata Totale dei CD per artista -->  
</td>  
<td align="right">  
  <xsl:value-of select="count(albums//album)"/>  
  <!-- Numero Totale di CD per artista -->  
</td>
```

Iterazioni, filtri

- Ciclo for

```
<xsl:for-each select="albums/album">  
  <!-- Regole da applicare -->  
</xsl:for-each>
```

- Condizionale

```
<xsl:template match="//artista//album">  
  <xsl:if test="durata > 60">  
    <tr bgcolor="#f3f3f3">  
      <td width="150">  
        <xsl:value-of select="../../@nome"/>  
      </td>  
      <td>  
        <xsl:value-of select="durata"/>  
      </td>  
    </tr>  
  </xsl:if>  
</xsl:template>
```

Ma perché a noi interessa l'XML?

Estensioni spaziali XML

- **Gestire i dati vettoriali su Internet con l'XML**
 - alleggerire la renderizzazione sul server
 - consentire query sul client
 - Cambiare le tematizzazioni sul client
- **SVG (Scalable Vector Graphic)**
 - http://it.wikipedia.org/wiki/Scalable_Vector_Graphics
- **GML (Geography Markup Language)**
 - http://it.wikipedia.org/wiki/Geography_Markup_Language
- **GML Application Schemas**
 - http://en.wikipedia.org/wiki/GML_Application_Schemas
- **KML (Keyhole Markup Language)**
 - http://it.wikipedia.org/wiki/Keyhole_Markup_Language

Riferimenti XML

Risorse

XML@W3C: <http://www.w3.org/XML/>

XML Tutorial: <http://www.w3schools.com/xml/default.asp>

SAX specifications: <http://www.saxproject.org>

Specifiche RSS: <http://blogs.law.harvard.edu/tech/rss>

Esempio RSS: <http://www.repubblica.it/rss/homepage/rss2.0.xml>

Parser web RSS: <http://www.aggreg8.net/RSS/RSSParser.php>

DTD tutorial: <http://www.w3schools.com/dtd/default.asp>

XMLSchema tutorial: <http://www.w3schools.com/schema/default.asp>

DOM in Java: <http://www.jdom.org/>



Punti di vista

ASP o PHP

ASP o PHP?

```
<html>
<head>
  <title>
    Pagina di prova
  </title>
</head>
<body>
  Buona giornata!
</body>
</html>
```

HTML



```
<html>
<head>
  <title>
    <%
      response.write ("Pagina di prova")
    %>
  </title>
</head>
<body>
  <%
    response.write ("Buona giornata!")
  %>
</body>
</html>
```

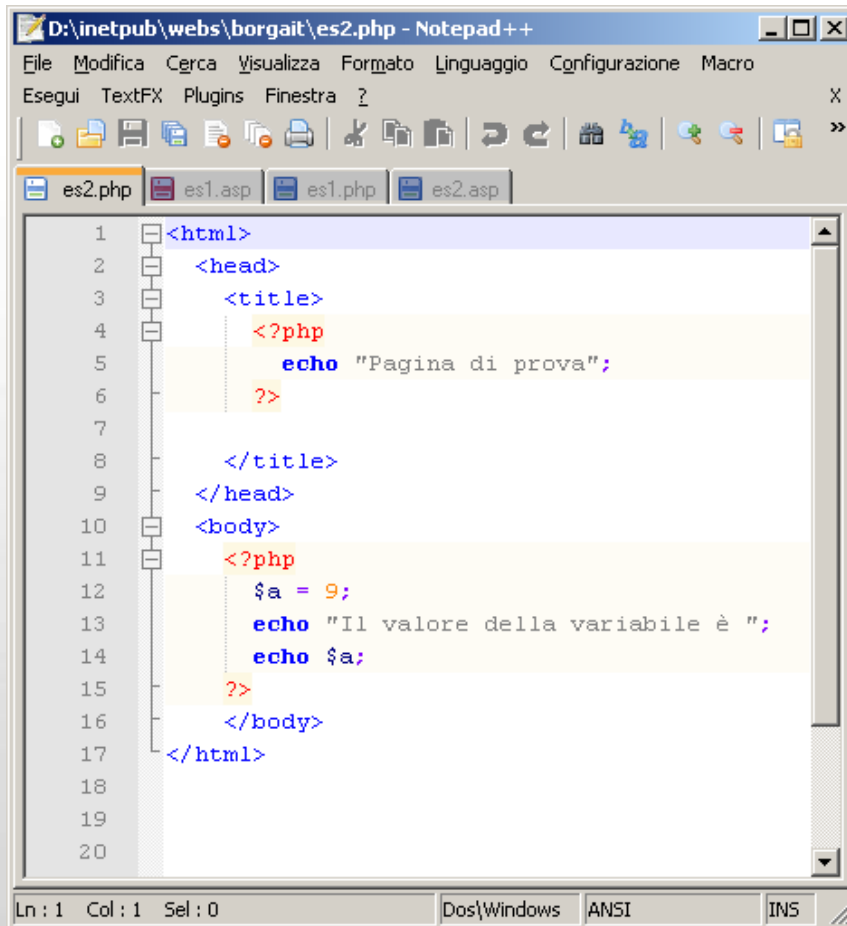
ASP



```
<html>
<head>
  <title>
    <?php
      echo "Pagina di prova";
    ?>
  </title>
</head>
<body>
  <?php
    echo "Buona giornata!";
  ?>
</body>
</html>
```

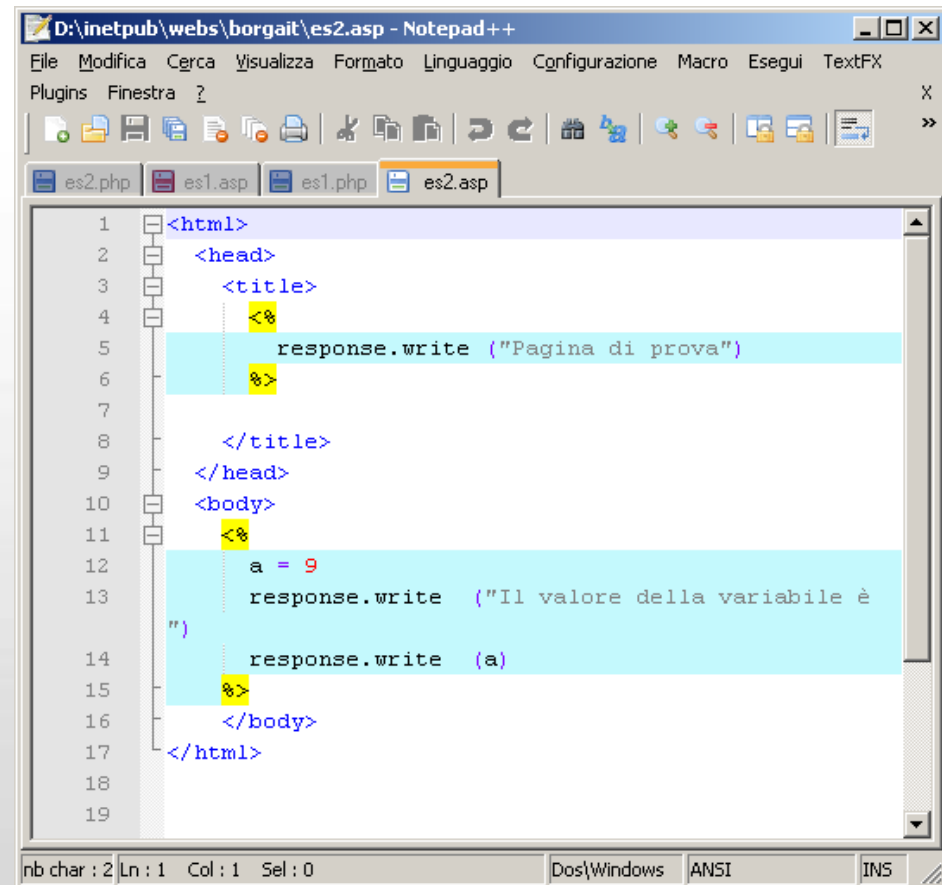
PHP

ASP o PHP?



The screenshot shows a Notepad++ window titled "D:\inetpub\webs\borgait\es2.php - Notepad++". The code is written in PHP within an ASP file. It includes an HTML header with a title and a body with two PHP blocks. The first block echoes "Pagina di prova". The second block sets a variable \$a to 9 and echoes its value.

```
1 <html>
2   <head>
3     <title>
4       <?php
5         echo "Pagina di prova";
6       ?>
7     </title>
8   </head>
9   <body>
10    <?php
11      $a = 9;
12      echo "Il valore della variabile è ";
13      echo $a;
14    ?>
15  </body>
16 </html>
```



The screenshot shows a Notepad++ window titled "D:\inetpub\webs\borgait\es2.asp - Notepad++". The code is written in ASP within an ASP file. It includes an HTML header with a title and a body with two ASP blocks. The first block echoes "Pagina di prova". The second block sets a variable a to 9 and echoes its value.

```
1 <html>
2   <head>
3     <title>
4       <%
5         response.write ("Pagina di prova")
6       %>
7     </title>
8   </head>
9   <body>
10    <%
11      a = 9
12      response.write ("Il valore della variabile è
13      ")
14      response.write (a)
15    %>
16  </body>
17 </html>
```

Come funzionano

- Il browser richiede una pagina .asp o .php.
 - Può inviare parametri all'applicazione attraverso vari metodi i cui due principali sono
 - GET: informazioni passati attraverso il querystring del tipo
`http://host/myfile.asp?firstname=Mario&lastname=Rossi&age=20&users
tatus=new`
 - POST: informazioni passate nel BODY del messaggio HTTP (nei FORM HTML)
- Il web server avvia l'interprete relativo (Application Server) specificando la pagina richiesta.
 - L'interprete è una DLL caricata dal web server
 - È una CGI (Common Gateway Interface)
- Il risultato dell'elaborazione viene restituito al web server.
- Il web server restituisce il codice HTML al browser
 - Al browser non arriva mai il codice di scripting della pagina

Cos'è

- PHP: Hypertext Preprocessor
 - Linguaggio di scripting
- Il codice PHP è immerso nell'HTML ed è delimitato da tag di start "<?" e end "?>"
- PHP esiste su tutti i principali sistemi operativi
- È supportato dalla maggior parte dei web server esistenti come Apache e IIS.

Tipi di variabili

- Tipi scalari:
 - boolean
 - integer
 - float
 - string
- Tipi composti:
 - array
 - object
- Tipi speciali:
 - resource
 - NULL

Funzioni

```
function myfunc($arg_1, $arg_2, ..., $arg_n)
{
    echo "Funzione di esempio.\n";
    return $retval;
}
```

Strutture di controllo PHP

Costrutti

▪ if.. else.. elseif

```
if ($a > $b) {  
    print "a > b";  
} else {  
    print "a <= b";  
}
```

▪ while

```
$i = 1;  
while ($i <= 10) {  
    print $i++;  
}
```

▪ do..while

```
$i = 0;  
do {  
    print $i;  
} while ($i>0);
```

▪ for

```
for ($i = 1; $i <= 10; $i++)  
    { print $i; }
```

▪ for..each

```
$a = array (1, 2, 3, 17);  
foreach ($a as $v) {  
    print "Valore corrente di \$a:  
    $v.\n"; }
```

▪ switch

```
switch ($i) {  
    case 0: print "i è uguale a 0";  
    break;  
    case 1: print "i è uguale a 1";  
    break;  
    case 2: print "i è uguale a 2";  
    break; }
```

Variabili speciali

- **\$_SERVER**: contiene variabili impostate dal web server e relative all'ambiente di esecuzione dello script
- **\$_GET**: contiene variabili ricevute dallo script via HTTP GET
- **\$_POST**: contiene variabili ricevute dallo script via HTTP POST
- **\$_COOKIE**: contiene variabili ricevute dallo script tramite l'invio di cookie
- **\$_ENV**: contiene variabili d'ambiente dello script.
- **\$_SESSION**: contiene variabili che sono registrate nella sessione corrente di esecuzione dello script.

Gestire la sessione

- Memorizzare una variabile con `$_SESSION`.

```
if (!isset($_SESSION['count'])) {  
    $_SESSION['count'] = 0; }  
else {  
    $_SESSION['count']++;  
}
```

- Eliminare una variabile dalla sessione.

```
unset($_SESSION['count']);
```

- Memorizzare una variabile con `session_register()`.

```
$nome = "Mario";  
session_register("nome");
```


Cos'è

- ASP: Active Server Pages

- VBScript

- Microsoft JScript.

```
<SCRIPT LANGUAGE="VBScript" RUNAT="Server" >
```

```
Codice script
```

```
</SCRIPT>
```

- Il codice ASP è immerso nell'HTML ed è delimitato da tag di start "<%>" e end "%>"
- ASP funziona solo Windows e su IIS
 - Può utilizzare tutti gli oggetti COM di Windows
- Esistono anche moduli per Apache, ma non è il caso di fidarsi

Tipi di variabili

- In VBScript, tutte le variabili sono di tipo variant e si possono memorizzare diversi tipi di dati come
 - Numero Intero, Numero in Virgola Mobile, Stringa, Data, Boolean, Valuta
- Tipi scalari:

dim name

name=valore

- Tipi vettoriali:

dim giorni(7)

giorni(0)="domenica"

giorni(1)="lunedì"

giorni(2)="martedì"

dim tabella(4, 6)

Funzioni e procedure

```
<%@LANGUAGE=VBSCRIPT%>
<HTML><BODY>
<%Call Echo%><BR>
<%Call PrintDate()%>
</BODY></HTML>
<%Sub Echo
    Response.Write "<P>Ciao Mondo!"
End Sub%>
<SCRIPT LANGUAGE=Jscript RUNAT=Server>
function PrintDate(){
    var x;
    x=NewDate()
    Response.Write("<P>Oggi è il ");
    Response.Write(x.toString())
}</SCRIPT>
```

Strutture di controllo ASP

Costrutti

- **if..then**

- **if..then..end if**

- **if..then..else..end if**

- **if..then..elseif..end if**

If (condizione) Then istruzione

- **select case**

Select Case (espressione)

Case "valore1"

istruzioni

Case "valore2"

istruzioni

Case Else

codice_di_default

End Select

- **For...Next**

*For cont = valore_iniziale To
valore_finale Step incremento*

codice

*...
Next*

- **For Each...Next**

- **Do while...Loop**

Do While (condizione)

codice

Loop

- **While...Wend**

Variabili speciali

- **Request**
 - per ottenere tutte le informazioni che vengono passate con una richiesta HTTP.
 - contiene due collection
 - **QueryString**: per il metodo GET
 - **Form**: per il metodo POST
- **Response**
 - per gestire le informazioni da spedire verso il client.
- **Server**
 - per accedere ai metodi e alle proprietà del server.
- **Session**
 - per registrare informazioni relative a sessioni.
- **Application**
 - per gestione di dati comuni a tutte le istanze di esecuzione dello script.
- **FileSystem**
 - Permette di manipolare files e directories

Gestire la sessione e l'applicazione

▪ Sessione

-collezionare informazioni di tracciamento dell'utente.

```
<% session("cognome")="Rossi"  
session("nome")="Carlo"%>
```

▪ Applicazione

-memorizza informazioni globali e visibili quindi da tutti gli utenti.

```
<% application.lock  
application("valore")=application("valore")+1  
application.unlock%>
```



Fine IV lezione

Componenti e linguaggi

GIS e Geo WEB: piattaforme e architetture