UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB http://www.disit.org

UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

Big Data

Michela Paolucci

University of Florence, Department of Information Engineering,

DISIT Lab, http://www.disit.org, http://www.sii-mobility.org,

paolo.nesi@unifi.it, michela.paolucci@unifi.it,

MASTER BIG DATA, ANALYTICS AND TECHNOLOGIES FOR MANAGEMENT

Outline

- XML: *Extensible Markup Language*
  - Introduzione
  - Classi e Istanze
  - Proprietà
  - Applicazioni XML
- JSON: *Javascript Object Notation*
- *JS: Javascript*

- *ETL processes: Extract Transforming and Loading processes*
- *IoT*

XML: Extensible Markup Language

Introduzione

Classi e istanze

Proprietà

XML: cosa è

- XML: Extensible Markup Language:
  - è un meta-*linguaggio* che consente la creazione di linguaggi di mark-up
  - consente la rappresentazione di documenti e dati strutturati su supporto digitale
  - è uno dei più potenti e versatili sistemi per la creazione, archiviazione, preservazione e disseminazione di documenti digitali

…. ma è anche una famiglia di tecnologie complementari

XML: le origini

- XML è stato sviluppato dal World Wide Web Consortium (http://www.w3.org)
- Le specifiche sono state rilasciate come *W3C Recommendation* nel 1998 e aggiornate nel 2008
- XML deriva da SGML (Standard Generalized Markup Language), un linguaggio di mark-up dichiarativo sviluppato dalla International Standardization Organization (ISO) e pubblicato ufficialmente nel 1986 con la sigla ISO 8879
- XML nasce come un sottoinsieme semplificato di SGML orientato alla utilizzazione su World Wide Web…
  … ma ha assunto ormai un ruolo autonomo e una diffusione ben maggiore del suo progenitore
- XML è abbastanza generale per poter essere utilizzato nei più disparati contesti

XML: caratteristiche (1)

- XML permette di esplicitare la struttura di un documento in modo formale mediante marcatori (mark-up) che vanno inclusi all'interno del testo
- XML è un metalinguaggio di mark-up, cioè un linguaggio che permette di definire sintatticamente altri linguaggi di mark-up.

- A differenza di HTML, XML non ha tag predefiniti e non serve per definire pagine Web né per programmare
  - … serve esclusivamente per definire altri linguaggi

- In realtà, XML non è altro che un insieme standard di regole sintattiche per modellare la struttura di documenti e dati. Queste regole, dette ***specifiche***, definiscono le modalità secondo cui è possibile crearsi un proprio linguaggio di mark-up. Le specifiche ufficiali sono state definite dal W3C (World Wide Web Consortium, http://www.w3.org/XML)

Concetto di mark-up / tag

- Nel documento ogni elemento non vuoto (contenente cioè altri elementi o caratteri) deve essere marcato da un **tag iniziale** e da un **tag finale**
- Ogni tag è costituito da caratteri delimitatori e dal nome dell'elemento
- Sintassi di un elemento:

Il concetto di metalinguaggio

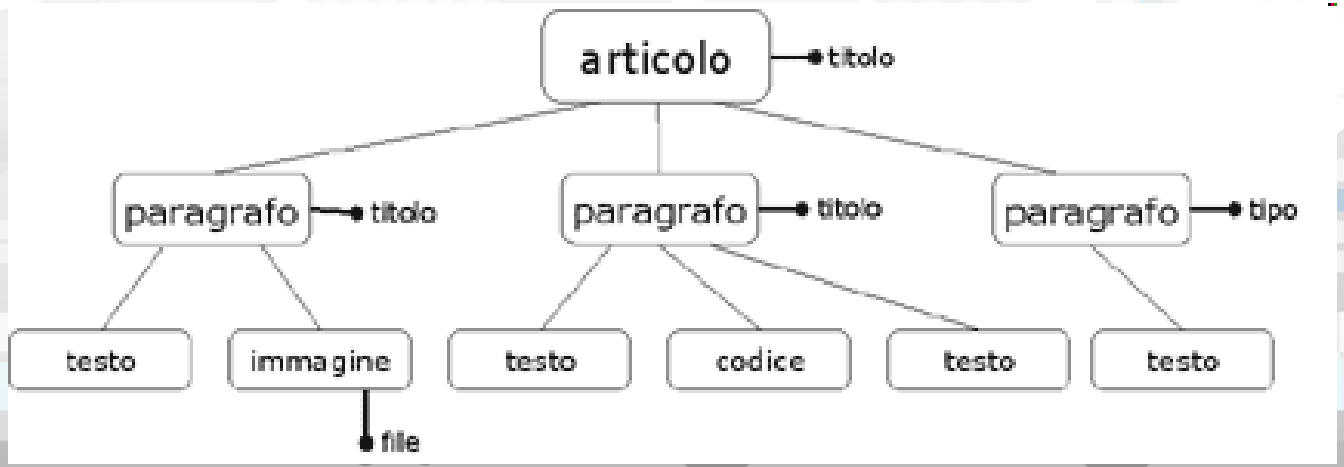
- XML è un **metalinguaggio**
  - XML definisce un insieme regole (meta)sintattiche, attraverso le quali è possibile descrivere formalmente un linguaggio di mark-up, detto "applicazione XML"
  - ogni applicazione XML eredita un insieme di caratteristiche sintattiche comuni
  - ogni applicazione XML a sua volta definisce una sintassi formale particolare
  - ogni applicazione XML è dotata di una semantica specificata in modo non formale

XML: caratteristiche (2)

- XML è indipendente dal tipo di piattaforma hardware e software su cui viene utilizzato
- XML permette la rappresentazione di qualsiasi tipo di documento (e di struttura testuale) indipendentemente dalle finalità applicative
- XML è indipendente dai dispositivi di archiviazione e visualizzazione
  - un documento XML può essere archiviato su qualsiasi tipo di supporto digitale (attuale e… futuro!)
  - un documento XML può essere visualizzato su qualsiasi dispositivo di output

XML: caratteristiche (3)

- XML adotta un formato di file di tipo testuale: sia il mark-up sia il testo, sono stringhe di caratteri
- XML si basa sul sistema di codifica dei caratteri ISO 10646/UNICODE
- Un documento XML è "leggibile" da un utente umano senza la mediazione di software specifico
- Concretamente, un documento XML è un file di testo che contiene una serie di tag, attributi e testo, secondo regole sintattiche ben definite.

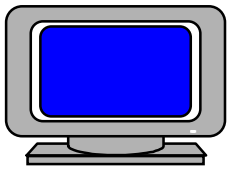


XML: caratteristiche (4)

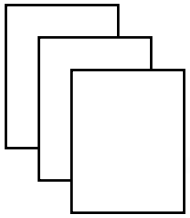
File XML

```
<Title> Titolo </title>

<p>Paragarafo ..
<p>Paragrafo
```



On-line WWW



Carta

CD-ROM

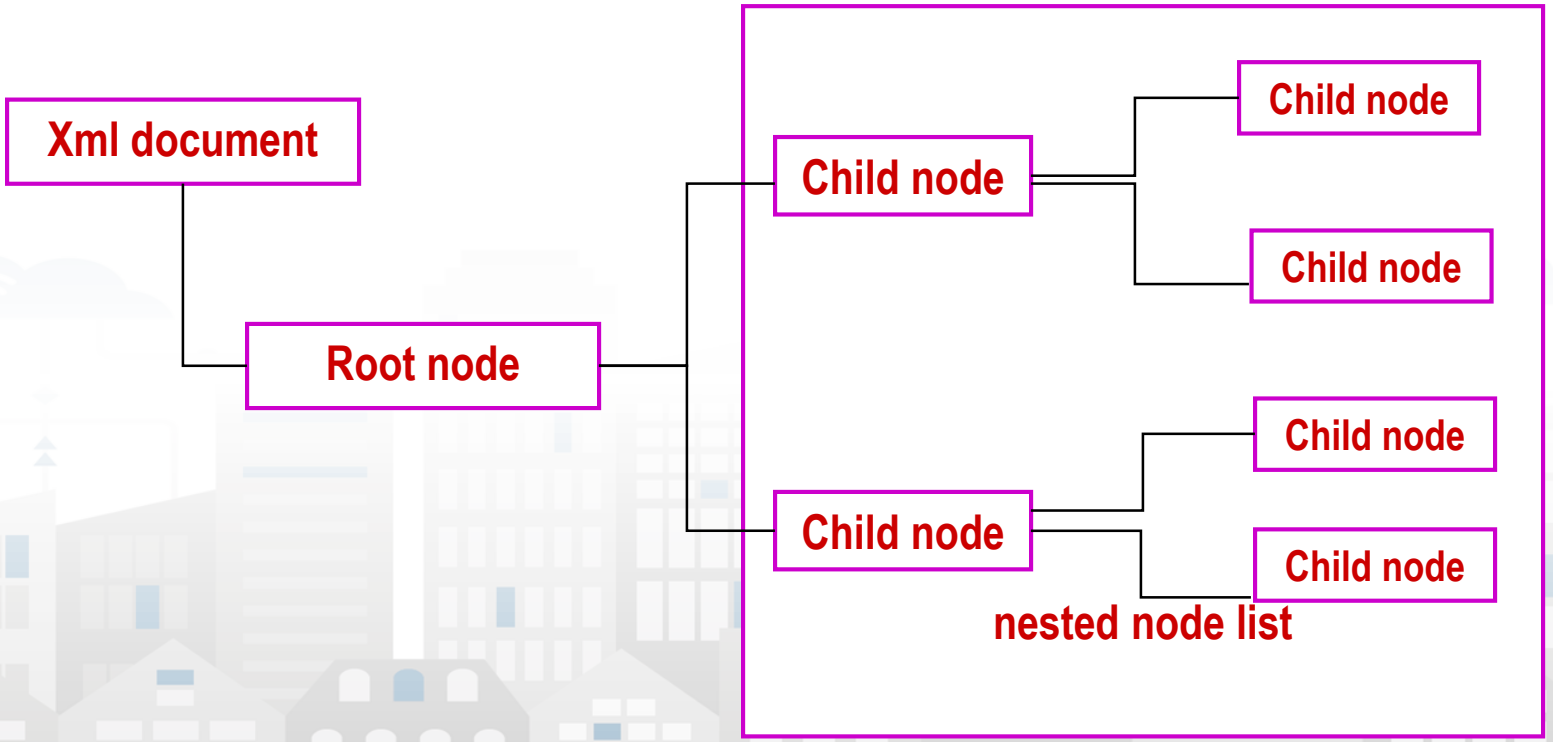
DBMS NATIVI O RELAZIONALI

DOCUMENT MANAGEMENT SYSTEMS

XML secondo il W3C

- Deve permettere l'interscambio di dati attraverso la rete internet
- Deve supportare per una grande varietà di applicazioni
- Deve essere compatibile con SGML
- Deve essere di estrema semplicità di elaborazione per le macchine
- Deve avere caratteristiche opzionali prossime allo 0
- Deve essere leggibile dagli umani
- La progettazione deve essere semplice ed intuitiva
- La progettazione deve essere formale e concisa
- Deve essere facile da creare
- Deve esser indipendente dalla piattaforma

La struttura gerarchica ordinata



Esempio: articolo

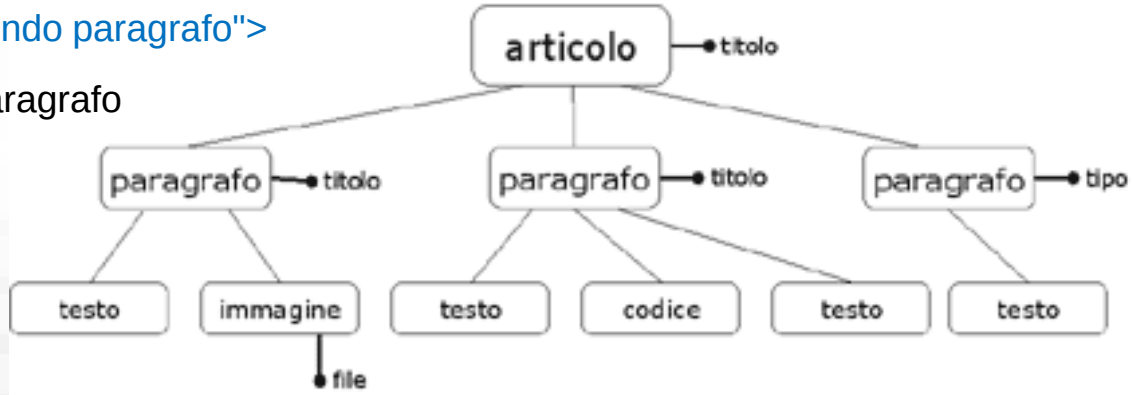
```
<?xml version = "1.0" ?>
  <articolo titolo = "Titolo dell'articolo">

    <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>
```



Strutture XML: gli elementi (1)

- I componenti strutturali di un documento sono denominati elementi (element)
- Ogni nodo dell'albero del documento è un (tipo di) elemento
- Ogni (tipo di) elemento è dotato di un nome (detto identificatore generico) che lo identifica
- Ciascun (tipo di) elemento rappresenta un componente logico del documento e può contenere altri (tipi di) elementi (sotto-elementi) o del testo
- L'organizzazione degli elementi segue un ordine gerarchico (ad albero) che prevede un elemento principale, chiamato root element o radice
- La radice contiene l'insieme degli altri elementi del documento
- E' possibile rappresentare graficamente la struttura di un documento XML tramite un albero, generalmente noto come ***document tree***

Strutture XML: gli elementi (2)

- Esiste uno e uno solo elemento radice (corrispondente al nodo radice dell'albero), che non è contenuto da nessun altro e che contiene direttamente o indirettamente tutti gli altri
- Ogni elemento, escluso l'elemento radice, deve essere contenuto da un solo elemento (elemento padre) e può contenere altri sotto-elementi (elementi figli) e/o stringhe di caratteri
- Esiste un sottoinsieme di elementi che non contengono altri elementi e che possono:
  - essere vuoti
  - contenere esclusivamente stringhe di caratteri
- I (tipi di) elementi possono avere associate altre informazioni che ne descrivono le proprietà. Queste informazioni sono chiamate attributi

Strutture XML: gli attributi

- Ad ogni elemento possono essere associati uno o più attributi che ne specificano ulteriori caratteristiche o proprietà non strutturali:
  - la lingua del suo contenuto testuale
  - un identificatore univoco
  - un numero di ordine
  - etc.

- Gli attributi XML sono caratterizzati da:
  - un nome che li identifica
  - un valore

Strutture XML: le entità (1)

- Un documento XML (in quanto oggetto digitale) ha una struttura fisica
- Dal punto di vista fisico un documento è composto da unità di archiviazione che sono denominate entità (entity)
- Esiste almeno una entità in ogni documento XML: la document entity, che contiene il documento stesso
- In generale una entità è 'una qualsiasi sequenza di byte considerata indipendentemente dalla sua funzione strutturale':
  - un singolo carattere UNICODE
  - una stringa di testo XML (caratteri e mark-up)
  - un intero file XML esterno
  - un intero file non XML (es. immagini digitali, etc.)
- È possibile ad esempio rappresentare nel contenuto di un documento caratteri non presenti sulla tastiera mediante entità

Strutture XML: le entità (2)

- Le entità vanno definite con apposite dichiarazioni nel DTD (o nel XML-schema)
- Una entità ha un nome e un contenuto
- In un documento l'inserimento di una entità avviene mediante un riferimento a entità che ne specifica il nome
- Un processore XML sostituirà automaticamente il contenuto dell'entità al posto del riferimento

XML: documenti ben formati e validi (1)

- XML richiede un certo rigore sugli aspetti sintattici
- Ogni documento XML deve essere ben formato (well formed)
- Un documento, in generale, è ben formato se:
- la sua struttura è implicita nel markup
- rispetta i vincoli di buona formazione indicati nelle specifiche
- Più in dettaglio:
  - Ogni documento XML deve contenere un unico elemento di massimo livello (root) che contenga tutti gli altri elementi del documento. Le sole parti di XML che possono stare all'esterno di questo elemento sono i commenti e le direttive di elaborazione (per esempio, la dichiarazione della versione di XML)
  - Ogni elemento deve avere un tag di chiusura o, se vuoti, possono prevedere la forma abbreviata (/>)
  - Gli elementi devono essere opportunamente nidificati, cioè i tag di chiusura devono seguire l'ordine inverso dei rispettivi tag di apertura
  - XML fa distinzione tra maiuscole e minuscole, per cui i nomi dei tag e degli attributi devono coincidere nei tag di apertura e chiusura anche in relazione a questo aspetto
  - I valori degli attributi devono sempre essere racchiusi tra singoli o doppi apici
- Un documento XML ben formato non richiede la presenza di un DTD o xml-schema

XML: documenti ben formati e validi (2)

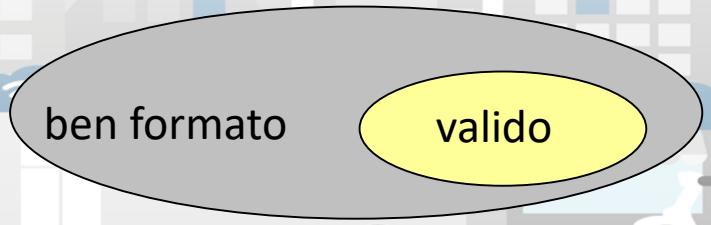
- Oltre ad essere ben formato un documento XML può anche essere **valido**

- Per stabilire la validità di un documento xml è necessario definire una **grammatica (tipo di documento)** per il linguaggio di mark-up che abbiamo ideato, a cui poi si farà riferimento per la validazione del documento xml

- Una **grammatica** è un insieme di regole che indica quali vocaboli (elementi) possono essere utilizzati e con che struttura è possibile comporre frasi (documenti)

XML: documenti ben formati e validi (3)

- Una **grammatica** definisce uno specifico linguaggio di mark-up => Se un documento XML rispetta le regole definite da una grammatica è detto **valido** per un particolare linguaggio
- Un documento ben formato può non essere valido rispetto ad una grammatica, mentre un documento valido è necessariamente ben formato
- Un documento valido per una grammatica può non essere valido per un'altra grammatica
- Una grammatica si definisce attraverso i due principali approcci:
  - **Dtd** - Document Type Definition
  - **XML Schema**



XML: documenti ben formati e validi (4)

- Un documento è **valido** se:
  - si riferisce a una DTD esplicita mediante un Doctype declaration (o ad un xml-schema)
  - Soddisfa i vincoli sintattici del DTD o xml-schema (nome, sequenza, occorrenze ed attributi degli elementi)
- Il controllo di validità viene effettuato da un apposito programma detto parser
- I parser si possono dividere in due categorie:
  - parser **non validante**: verifica soltanto se un documento è ben formato
  - parser **validante:** oltre a verificare che un documento sia ben formato, verifica anche se è corretto rispetto ad una data grammatica (dtd o xml-schema)
- La maggior parte degli editor XML più recenti ha un parser integrato o si appoggia su parser esterni per effettuare la convalida dei documenti

Come si crea un documento XML

- Un documento XML contiene il mark-up (sotto forma di coppie di tag) che rappresenta linearmente la struttura gerarchica degli elementi, i loro eventuali attributi, e i caratteri del testo

- L'inserimento di mark-up e caratteri deve rispettare i vincoli di buona formazione indicati nelle specifiche XML

- Un documento valido deve rispettare anche i vincoli sintattici definiti nel DTD o xml-schema

Aspetti di sintassi generale e vincoli

- Un documento XML è una stringa di caratteri UNICODE in codifica UTF-8 o UTF-16 (http://www.unicode.org/)
- I nomi di elementi, attributi e entità sono sensibili alla differenza tra maiuscolo e minuscolo
- Il mark-up è separato dal contenuto testuale mediante caratteri speciali: **< > &**
- Vincoli di buona formazione:
  - I caratteri speciali non possono comparire come contenuto testuale e devono essere eventualmente sostituiti mediante i riferimenti a entità: **< > &**
  - Esiste un solo elemento radice
  - Tutti gli elementi non vuoti devono presentare sia il tag iniziale sia il tag finale
  - Tutti gli elementi devono essere correttamente annidati
  - Tutti i valori di attributo devono essere racchiusi tra apici doppi o singoli

XML: documenti ben formati (1)

- Anche la scelta dei nomi dei tag deve seguire alcune regole:
  - Un tag può iniziare con un lettera o un underscore (_) e può contenere lettere, numeri, il punto, l'underscore (_) o il trattino (-). Non sono ammessi spazi o altri caratteri.
  - XML è sensibile all'uso di maiuscolo e minuscolo, quindi i tag <prova> e <Prova> sono considerati diversi.
- Per quanto riguarda il contenuto:
  - un documento XML può contenere potenzialmente qualsiasi carattere dell'alfabeto latino, cifre e punteggiatura. Normalmente vengono accettati come caratteri validi in un documento XML i primi 128 caratteri della codifica ASCII (lettere dell'alfabeto latino minuscole e maiuscole, cifre, segni di punteggiatura, ecc.)
  - Se un documento contiene caratteri che non rientrano tra questi (es.: lettere accentate, simboli di valuta, ecc.) è necessario specificare lo schema di codifica utilizzato tramite elementi speciali detti direttive di elaborazione o processing instruction
  - es: <?xml version="1.0" encoding="iso-8859-1"?>
- Le specifiche di XML prevedono esplicitamente la possibilità di utilizzare la codifica Unicode per rappresentare anche caratteri non latini, come ad esempio i caratteri greci, cirillici, gli ideogrammi cinesi e giapponesi

XML: documenti ben formati (2)

- Oltre alle direttive di elaborazione, in un documento XML possiamo trovare i commenti che seguono la stessa sintassi dell'HTML, sono cioè racchiusi tra le sequenze di caratteri <!-- e --> e possono trovarsi in qualsiasi punto del documento.
- Potrebbe essere necessario inserire in un documento XML dei caratteri particolari che potrebbero renderlo non ben formato
  - Ad esempio, se dobbiamo inserire del testo che contiene il simbolo < corriamo il rischio che possa venire interpretato come l'inizio di un nuovo tag. Esempio:

<testo> il simbolo < indica minore di </testo>

In questo caso, XML prevede l'uso delle entità che consentono di sostituire altri caratteri. Cinque entità sono predefinite e consentono l'uso di altrettanti caratteri riservati all'interno di un documento:

& definisce il carattere & | **<** definisce il carattere <

> definisce il carattere > **"** definisce il carattere " **'** definisce il carattere '

Sfruttando le entità, l'Esempio precedente diventa:

<testo> il simbolo < indica minore di </testo>

XML: documenti ben formati (3)

- In alcune situazioni gli elementi da sostituire con le entità possono essere molti, il che rischia di rendere illeggibile il testo (not human readable). Si consideri il caso in cui un blocco di testo illustri proprio del codice XML:
  - <codice>
    <libro>
    <capitolo>
    </capitolo>
    </libro>
    </codice>

- In questo caso, al posto di sostituire tutte le occorrenze dei simboli speciali con le corrispondenti entità è possibile utilizzare una **sezione CDATA** (un blocco di info che viene considerato sempre come testo)
- Per indicare una sezione CDATA è sufficiente racchiuderla tra le sequenze di caratteri **<![CDATA[** e **]]>**:
- <codice>
  <![CDATA[
  <libro>
  <capitolo>
  </capitolo>
  </libro>
  ]]>
  </codice>

La forma di un documento XML

- Ogni documento XML inizia con un prologo che contiene:
  - una XML declaration
  - eventualmente una Doctype declaration (la dichiarazione della DTD o del xml-schema a cui il documento si riferisce)
  - eventualmente una serie di processing instruction

- Forme di XML declaration:
  - <?xml version="1.0"?>
  - <?xml version="1.0“ encoding=“UTF-8”?>

Esempio: articolo

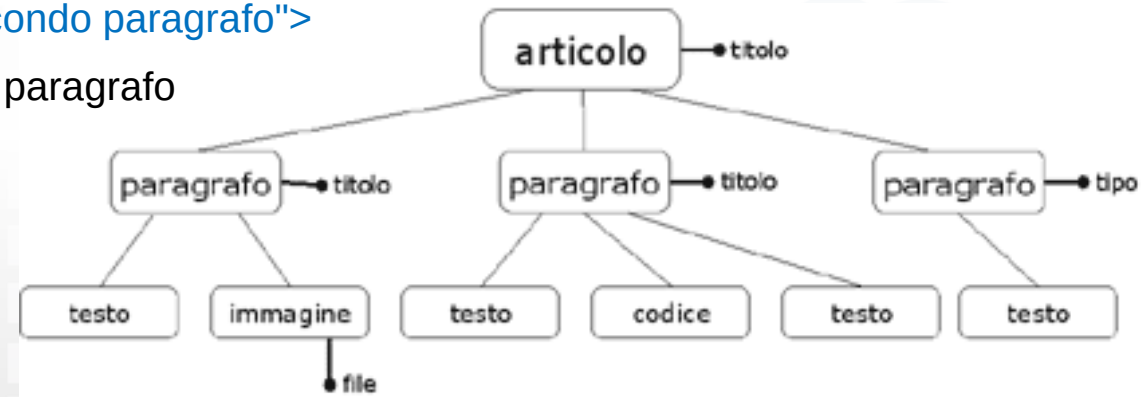
```
<?xml version="1.0" ?>
  <articolo titolo="Titolo dell'articolo">

    <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>
```



La codifica degli elementi (1)

- Nel documento ogni elemento non vuoto (contenente cioè altri elementi o caratteri) deve essere marcato da un **tag iniziale** e da un **tag finale**
- Ogni tag è costituito da caratteri delimitatori e dal nome dell'elemento
- Sintassi di un elemento:

La codifica degli elementi (2)

```
<text>
<div1>
<p>Subito, con le prime parole che le rivolse, volle avvisarla che non intendeva compromettersi in una relazione troppo seria…</p>
<p>La sua famiglia? Una sola sorella non ingombrante né fisicamente né moralmente, piccola e pallida, di qualche anno più giovane di lui…</p>
…
</div1>
</text>
```

La codifica degli elementi (3)

- La relazione lineare tra i tag rappresenta la relazione gerarchica tra gli elementi
- Per ogni elemento, se il suo tag iniziale è nel contenuto di un elemento P allora il suo tag finale deve essere nel contenuto del medesimo elemento P
- Detto altrimenti: le coppie di tag devono annidarsi correttamene e mai sovrapporsi

La codifica degli elementi (4)

SBAGLIATO!!!

Subito, con le prime parole che le rivolse, volle avvisarla che non intendeva compromettersi in <emph>una relazione troppo seria…

La sua famiglia?</emph>Una sola sorella non ingombrante né fisicamente né moralmente, piccola e pallida, di qualche anno più giovane di lui…

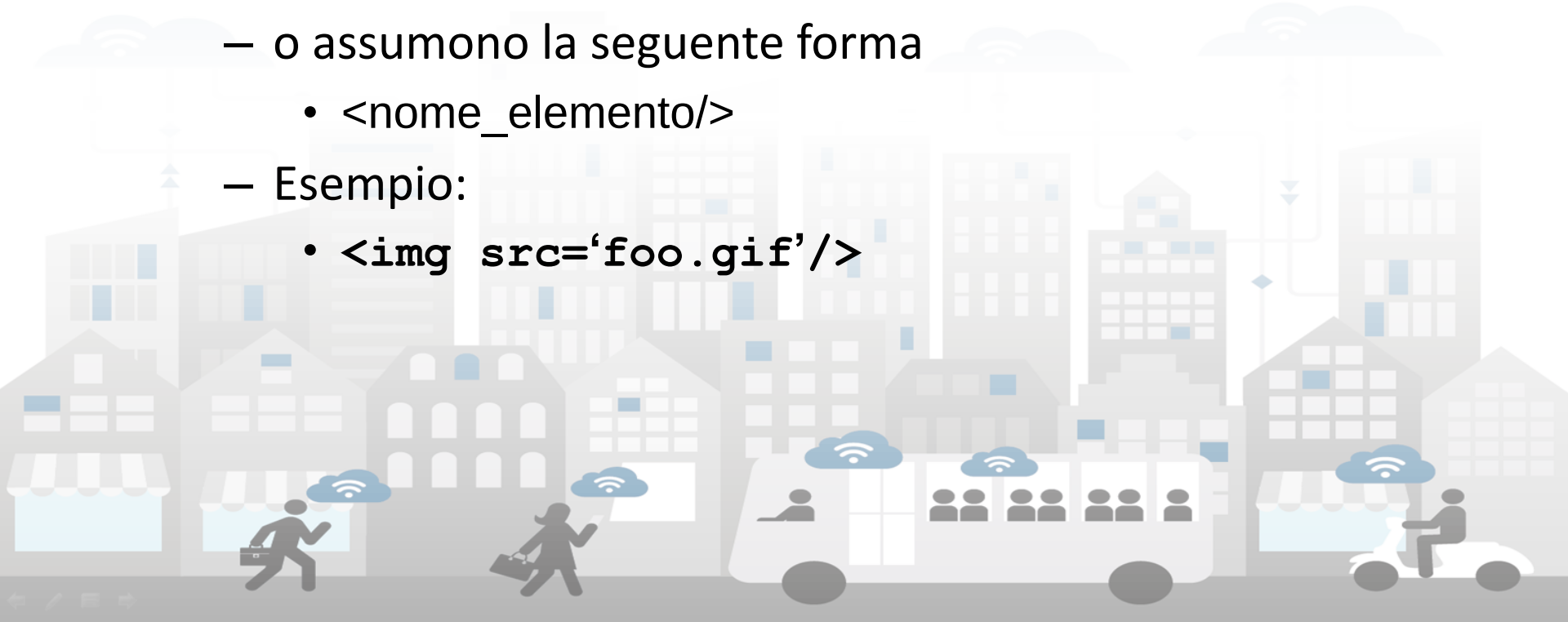
CORRETTO!!!

Subito, con le prime parole che le rivolse, volle avvisarla che non intendeva compromettersi in <emph>una relazione troppo seria… </emph>

 <emph>La sua famiglia?</emph> Una sola sorella non ingombrante né fisicamente né moralmente, piccola e pallida, di qualche anno più giovane di lui…

La codifica degli elementi (5)

- Gli elementi vuoti
  - o sono rappresentati da entrambi i tag
    - …<nome_elemento> </nome_elemento>…
  - o assumono la seguente forma
    - <nome_elemento/>
  - Esempio:
    - ****



La codifica degli attributi (1)

- Ogni elemento XML può avere uno o più attributi
- Un attributo ha un nome e un valore, che può assumere diverse tipologie
- Gli attributi devono essere associati agli elementi all'interno del tag iniziale dopo l'identificatore
  - <elemento attributo = "valore"> contenuto… </elemento>
- Altri eventuali attributi vanno collocati dopo il primo separati da uno o più spazi
- Non possono esservi più istanze dello stesso attributo per un elemento

La codifica degli attributi (2)

<text resp="Italo Svevo" n="Senilità">

Subito, con le prime parole che le rivolse, volle avvisarla che non intendeva compromettersi in una relazione troppo seria…

La sua famiglia? Una sola sorella non ingombrante né fisicamente né moralmente, piccola e pallida, di qualche anno più giovane di lui…

<pb n="5"/>

…

</text>

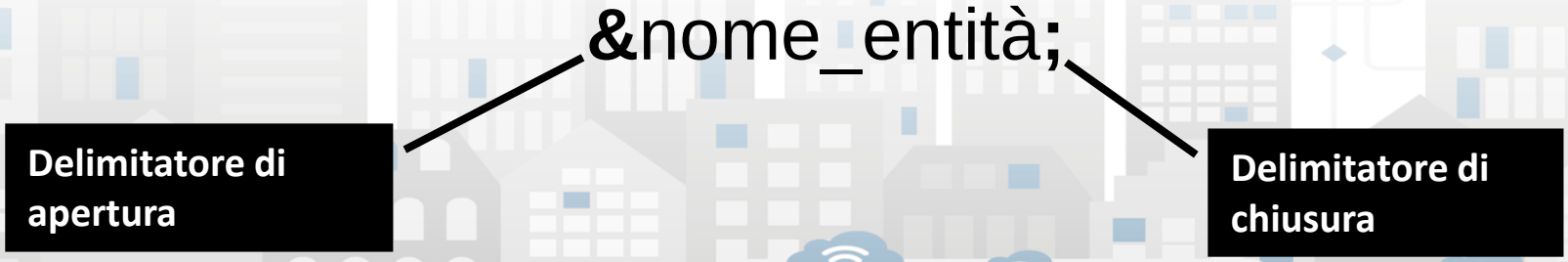
La codifica delle entità

- Per definire un'entità personalizzata si utilizza la dichiarazione: **<!ENTITY>**
- Il seguente esempio mostra la definizione di un'entità **&html;** che rappresenta un'abbreviazione per la stringa HyperText Markup Language:
  - **<!ENTITY** html "HyperText Markup Language"**>**

- Grazie a questa dichiarazione possiamo utilizzare l'entità
- **&html;** al posto dell'intera stringa all'interno del documento XML che fa riferimento a questa grammatica

Il riferimento alle entità

- L'inclusione di una entità all'interno di un documento SGML si effettua mediante un **riferimento a entità** (entity reference)
- La sintassi di un riferimento, valida sia per entità esterne sia interne, è la seguente



Il riferimento alle entità

- In questi esempi i caratteri accentati sono stati sostituiti da riferimenti a entità carattere:

```
<p>La sua famiglia? Una sola sorella non ingombrante
n&eacute; fisicamente n&eacute; moralmente, piccola
e pallida, di qualche anno pi&ugrave; giovane di
lui…</p>

<testo>
  il simbolo &lt; indica minore di
</testo>
```

Esempio: pagina html (1)

www.unifi.it


```
<!DOCTYPE html>
<html lang="it">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>UniFI - Università degli Studi di Firenze</title>
<meta name="description" content="UniFI - L'Università degli Studi di Firenze è una università statale italiana, fondata nel 1321 come Studium Generale">
<meta name="keywords" content="università,laurea,studio,study,studiare,Firenze,university,florence,studente,student,ateneo,didattica,ricerca,relazioni interna
<link rel="image_src" href="http://www.unifi.it/salomoneFB.png">
<link rel="alternate" href="http://www.unifi.it/backend.php" type="application/rss+xml" title="Università degli Studi di Firenze">
<link rel="shortcut icon" href="http://mdthemes.unifi.it/global/favicon.ico">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link media="screen" rel="stylesheet" type="text/css" href="http://mdthemes.unifi.it/azimuth/css/global.home2016-2.responsive.min.css">
<link media="print" rel="stylesheet" type="text/css" href="http://mdthemes.unifi.it/azimuth/css/stampahome.min.css">
<meta name="google-site-verification" content="SF3qktGLJy85YeUk1ityUlz4UO_t2Lb1VikCsFBcVkA">
</head>
<body>
  <div id="contenitore">
    <div id="logostampa"><img src="http://mdthemes.unifi.it/azimuth/images/testastampa.png" alt="Università degli Studi - Firenze" width="191" height="100"></d
    <div id="logobase"><a href="index.php"><img src="http://mdthemes.unifi.it/azimuth/images/head.png" alt="Logo UniFi" width="480" height="147" id="logo"></a>
    <div id="cambiolingua"><a href="changelang-eng.html">english<br> version</a></div>
    <div id="sol">
<a href="cmpro-v-p-10028.html">servizi online</a> <img src="http://mdthemes.unifi.it/azimuth/images/frecciasol.png" alt="SOL - Servizi on line" width="15" hei
</div>
    <div id="finistraricerca">
<form action="index.php?module=SEAF&amp;func=search" method="post">
<div>
<input type="hidden" value="www.unifi.it" name="sitesearch"><label for="q" class="hidden labelcasellaricerca">Cerca</label>
<input id="q" type="text" class="text" name="q" size="35" maxlength="100" value="cerca informazioni o persone" onfocus="this.value='';"><input id="gs" type="i
</div>
</form>
</div>
    <div id="userlinks"><!-- [user-links] --></div>
    <div id="fotocover"><ul class="rslides" id="slider">
<li><a href="cmpro-v-p-8736.html"><img src="upload/coverimages/fetch-UID-.INBOX.b posta%20evasa.jpg" alt="rinnovo iscrizioni" title="rinnovo iscrizioni" width
<li><a href="cmpro-v-p-3214.html"><img src="upload/coverimages/home%20tutor%20junior.jpg" alt="tutor junior" title="tutor junior" width="858" height="349"></a
<li><a href="cmpro-v-p-2695.html#cum laude"><img src="upload/coverimages/fetch-UID-.jpg" alt="Firenze Cum-Laude" title="Firenze Cum-Laude" width="858" height=
<li><a href="cmpro-v-p-10850.html"><img src="upload/coverimages/fetch-UID-.INBOX.jpg" alt="Welcome Service" title="Welcome Service" width="858" height="349"><
<li><a href="http://www.unifi.it/art-2149-elezioni-del-senato-accademico.html"><img src="upload/coverimages/home%20elezioni%20senato.jpg" alt="Elezioni del Se
</ul></div>
    <ul id="quadrati">
<li id="ateneo"><a href="cmpro-l-s-31.html">ateneo</a></li>
<li id="dida"><a href="cmpro-l-s-38.html">didattica</a></li>
<li id="ricerca"><a href="cmpro-l-s-33.html">ricerca</a></li>
<li id="rel-int"><a href="cmpro-l-s-32.html">ateneo<br> nel mondo</a></li>
<li id="inno"><a href="cmpro-l-s-39.html">innovazione<br> e imprese</a></li>
<li id="orientamento"><a href="cmpro-l-s-55.html">orientamento<br> e placement</a></li>
<li id="studenti"><a href="cmpro-l-s-27.html">studenti</a></li>
<li id="viv-uni"><a href="cmpro-l-s-40.html">vivere<br> l'università</a></li>
<li id="news"><a href="news.html">news</a></li>
<!--
<li id="avvisi"><a href="index.php?module=NEWSpro&func=list&catid=2" _mce_href="index.php?module=NEWSpro&func=list&catid=2">notices</a> -->
<li id="agenda"><a href="agenda.html">agenda</a></li>
<li id="biblio"><a href="http://www.sba.unifi.it/">biblioteche</a></li>
<li id="fup"><a href="http://www.fupress.com">firenze<br> university<br> press</a></li>
<li id="museo"><a href="http://www.msn.unifi.it/">museo<br> di storia<br> naturale</a></li>
</ul>
<div id="foto-sx-alto" class="baseimg"><p><img src="upload/sub/coverpagine/home/sxalto.jpg" alt="Copertina foto piccola 1" width="119" height="119"></p></div>
    <div id="foto-dx-alto" class="baseimg"><p><img src="upload/sub/coverpagine/home/dxalto.jpg" alt="Copertina foto piccola 2" width="118" height="119"></p></d
    <div id="foto-centrale" class="baseimg"><div class="contenitorevideo"><iframe width="489" height="241" src="https://www.youtube.com/embed/GTf-c-Lsipk?rel=0
    <div id="foto-sx-basso" class="baseimg"><p><a href="cmpro-v-p-2922.html"><img class="yuodsspfpswzanrhpqzw" src="upload/sub/coverpagine/home/avviso-1.jpg" a
    <div id="notizie">
<h2>News</h2>
 <div class="blocco-notizia">
   <a href="art-2158-architettura-via-ai-corsi-della-scuola-mediterranea-di-fez.html">
  <span class="grassetto">Architettura, via ai corsi della Scuola Mediterranea di Fez</span>
```

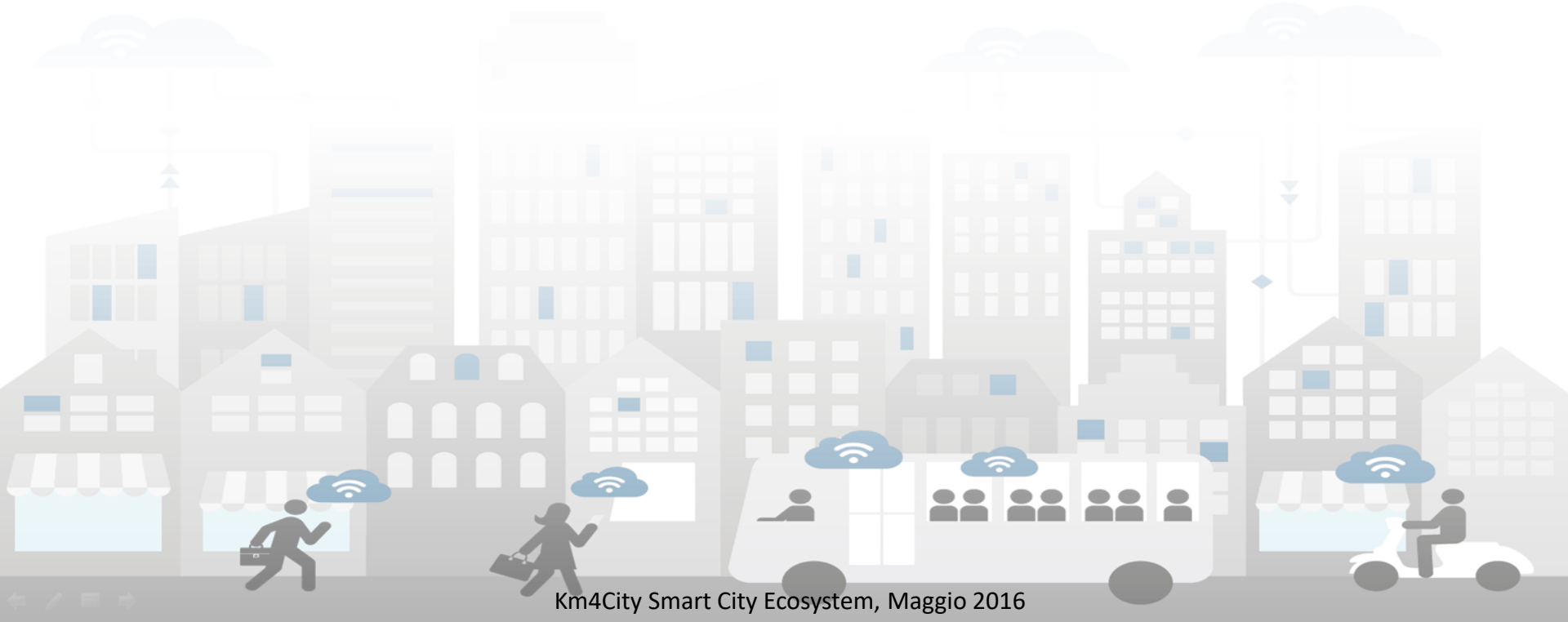
Esempio: pagina html (2)

www.unifi.it

- Presenza di tag predefiniti
  - <html>
  - <head>
  - <meta>
  - <body>
  - <link>
  - <a>
  -
  - <form>
  - <input>
  - …

```
<!DOCTYPE html>
<html lang="it">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>UniFI - Università degli Studi di Firenze</title>
<meta name="description" content="UniFI - L'Università degli Studi di Firenze è una
università statale italiana, fondata nel 1321 come Studium Generale">
<meta name="keywords" content=
"università,laurea,studio,study,studiare,Firenze,university,florence,studente,student,at
eneo,didattica,ricerca,relazioni internazionali,innovazione e lavoro,studenti,vivere
l'università,news,avvisi,agenda,servizi online,biblioteche,Firenze university
press,museo di storia naturale">
<link rel="image_src" href="http://www.unifi.it/salomoneFB.png">
[...]
</head>
<body>
  <div id="contenitore">
   <div id="logostampa"><img src="
   http://mdthemes.unifi.it/azimuth/images/testastampa.png" alt="Università degli
   Studi - Firenze" width="191" height="100"></div>
   <div id="logobase"><a href="index.php"><img src="
   http://mdthemes.unifi.it/azimuth/images/head.png" alt="Logo UniFi" width="480"
   height="147" id="logo"></a></div>
   <div id="cambiolingua"><a href="changelang-eng.html">english<br> version</a></div>
   <div id="sol">
<a href="cmpro-v-p-10028.html">servizi online</a> <img src="
http://mdthemes.unifi.it/azimuth/images/frecciasol.png" alt="SOL - Servizi on line"
width="15" height="15">
</div>
   <div id="finistraricerca">
<form action="index.php?module=SEAF&amp;func=search" method="post">
<div>
<input type="hidden" value="www.unifi.it" name="sitesearch"><label for="q" class=
"hidden labelcasellaricerca">Cerca</label>
<input id="q" type="text" class="text" name="q" size="35" maxlength="100" value="cerca
informazioni o persone" onfocus="this.value='';"><input id="gs" type="image" alt=
"Cerca" src="http://mdthemes.unifi.it/azimuth/images/cerca.gif" name="btnG">
</div>
```

TASSONOMIA



Tassonomia (1)

- 'Branca della scienza che studia i metodi di ordinamento in un sistema degli elementi, delle conoscenze, dei dati, delle teorie appartenenti a un determinato ambito scientifico.' <Treccani>
- 'A collection of controlled vocabulary terms organized into a hierarchical structure. Each term in a taxonomy is in one or more parent/child (broader/narrower) relationships to other terms in the taxonomy.' <ANSI/NISO - American National Standards Institute National Information Standards Organization) >

Tassonomia (1)

- Si trattadi una lista gerarchica di termini, ordinati dal più generale al più particolare
- Si pensi ad esempio alla classificazione degli esseri viventi di Linneo
  - 7 categorie sistematiche, ordinate in modo gerarchico (dalla più generica alla più specifica): Regno, Phylum (o divisione), Classe, Ordine, Famiglia, Genere, Specie

	Uomo	Leone	Gatto
Regno	Animal	Animal	Animal
Phylum	Chordate	Chordate	Chordate
Classe	Mammal	Mammal	Mammal
Ordine	Primate	Carnivore	Carnivore
Famiglia	Hominidae	Felidae	Felidae
Genere	*Homo*	*Panthera*	*Felis*
Specie	*Sapiens*	*Leo*	*Domesticus*

Esempio: Climate Bonds Taxonomy

- Climate Bonds Standard
  - Governance
    - Climate Bond Standards Board
    - Climate Bonds Scientific Framework
    - Technical Working Groups
    - Industry Working Group
  - Sector Criteria
    - Sector Criteria Available for Certification
      - Solar
      - Water
      - Wind
      - Low Carbon Buildings
      - Low Carbon Transport
      - Geothermal
    - Sector Criteria Available Soon
      - Bioenergy
      - Hydropower
      - Land Use
      - Marine
    - Public Consultation
  - Types of Bond
  - Taxonomy
  - FAQs
- Climate Bonds Certification
  - List of Certified Bonds
  - How to Get Certified
  - Certification Assurance
    - Become a Verifier
    - Approved Verifiers

Climate Bond Taxonomy

Climate Bonds Initiative

https://www.climatebonds.net/standards/taxonomy

ENERGY	TRANSPORT	WATER	LOW CARBON BUILDINGS	INFORMATION TECHNOLOGY & COMMUNICATIONS	WASTE & POLLUTION CONTROL	NATURE BASED ASSETS	INDUSTRY & ENERGY-INTENSIVE COMMERCIAL
Solar	Rail	Built (grey) infrastructure	Residential	Power management	Recycling facilities	Agricultural land	Manufacturing
Wind	Vehicles	Green and hybrid infrastructure	Commercial	Broadband	Recycled products & circular economy	Forests (managed and unmanaged)	Energy efficiency processes
Geothermal	Mass transit		Retrofit	Resource efficiency	Waste to energy	Wetlands	Energy efficiency products
Hydropower	Bus rapid transport		Products for building carbon efficiency	Teleconferencing	Methane management	Degraded Lands	Retail and wholesale
Bioenergy	Water-bourne transport				Geosequestration	Other land uses (managed and unmanaged)	Data centres
Wave and Tidal	Alternative fuel infrastructure					Fisheries and aquaculture	Process & fugitive emissions
Energy distribution & management						Coastal infrastructure	Energy efficient appliances
Dedicated transmission							Combined heat & power

Climate Bond Certified

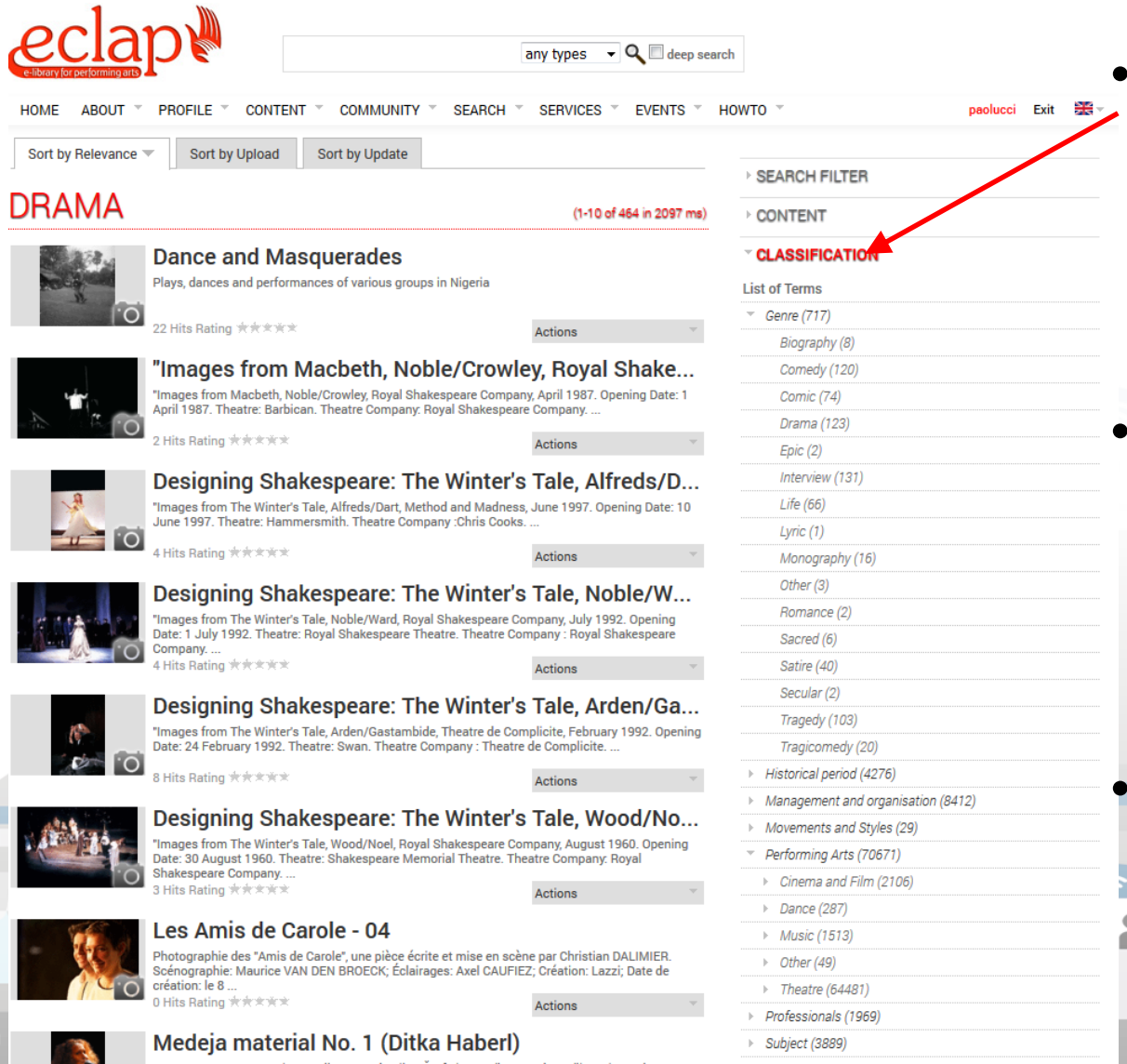
Certification Criteria approved
Criteria under development
Due to commence

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB http://www.disit.org

ECLAP (www.eclap.eu)

(European Collected Library of Artistic Performance)

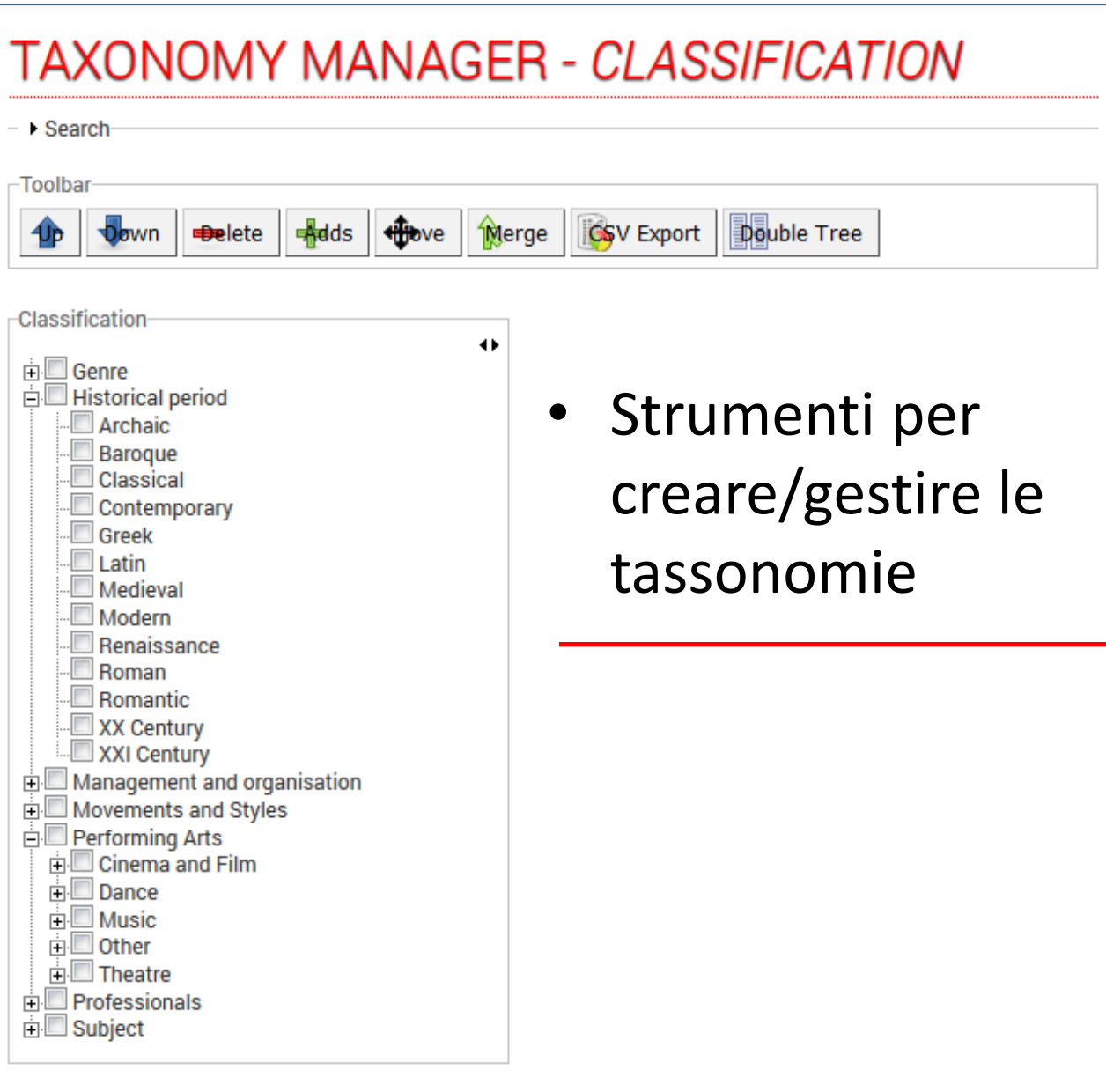
UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA



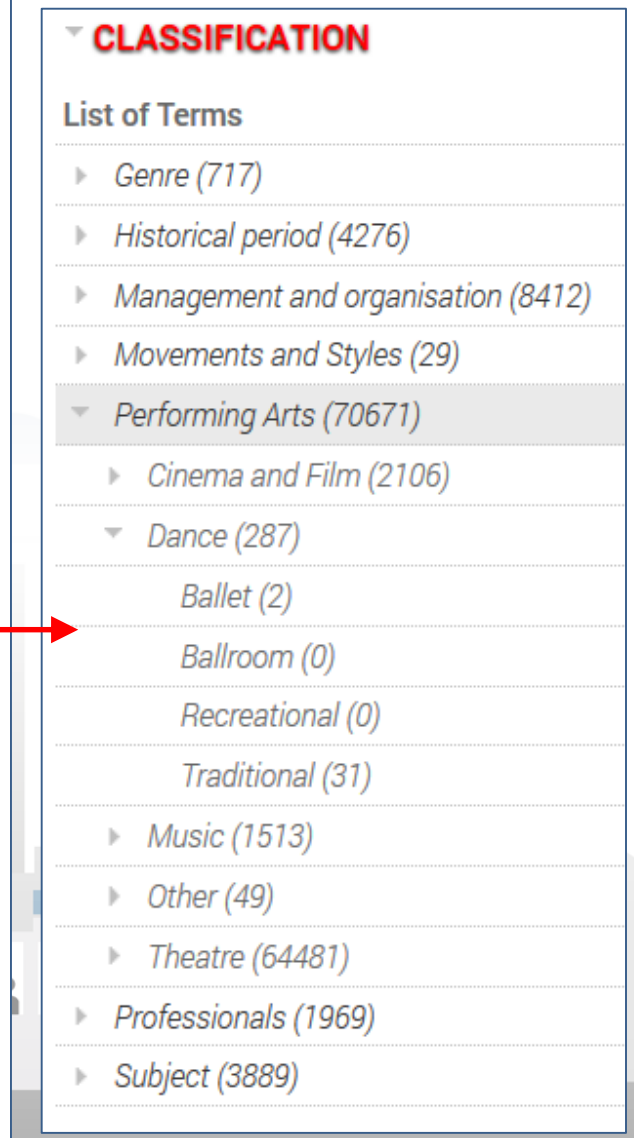
- Organizzazione dei contenuti multimediali all'interno di una Best Practice Network
- I contenuti sono classificati in base ai metadati (Dublin Core, etc.) e alle voci tassonomiche ad essi associate
- In questo modo è possibile anche effettuare azioni di ricerca mirate

ECLAP (www.eclap.eu)

TAXONOMY MANAGER - *CLASSIFICATION*



- Strumenti per creare/gestire le tassonomie

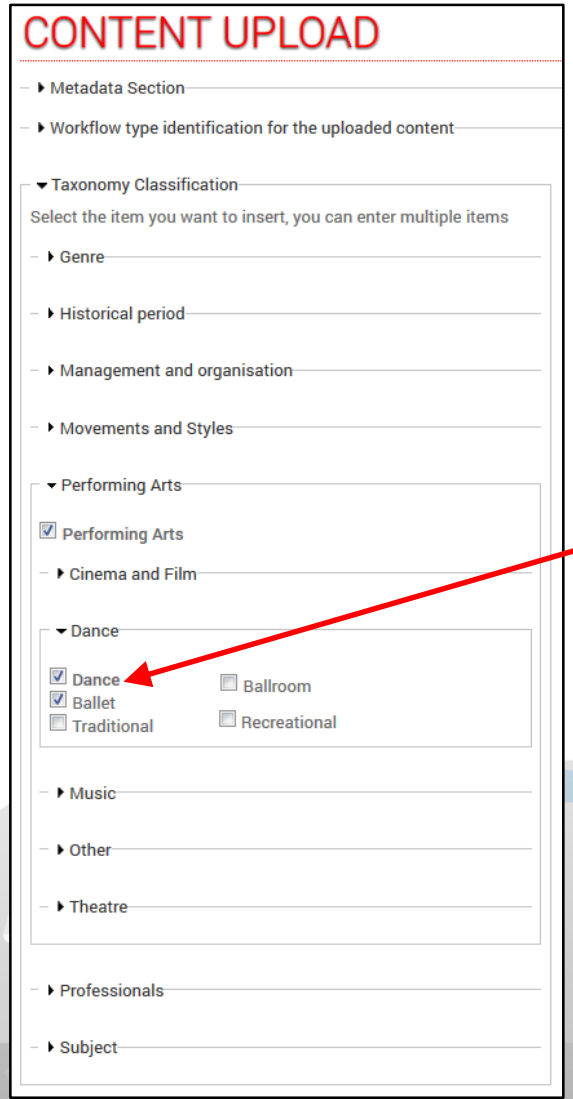


UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB http://www.disit.org

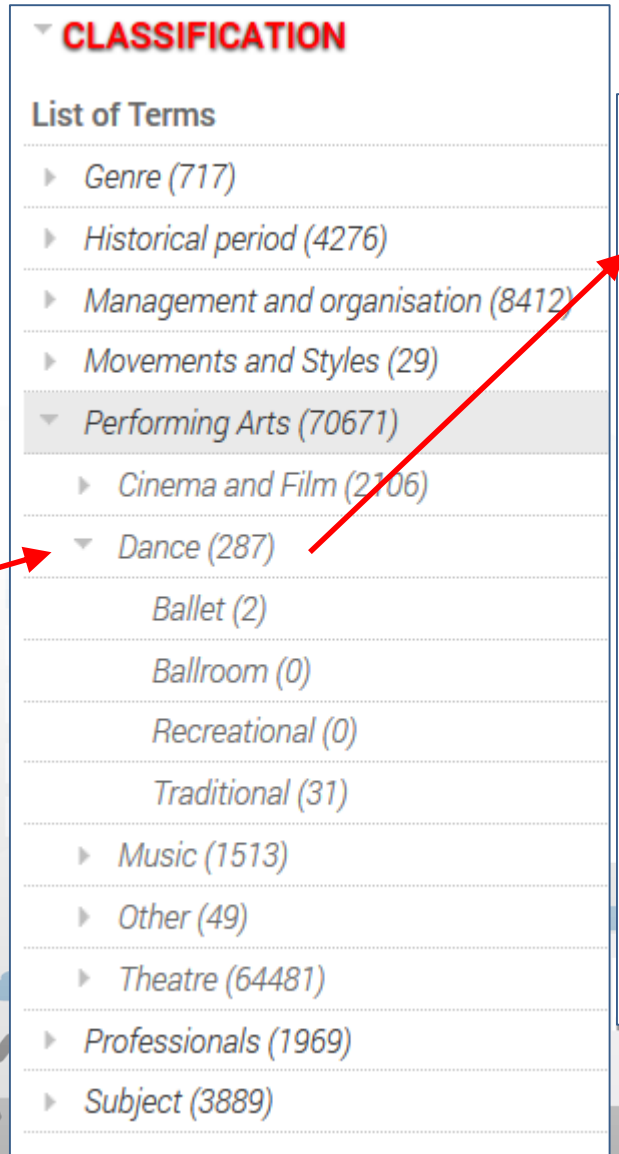
ECLAP (www.eclap.eu)

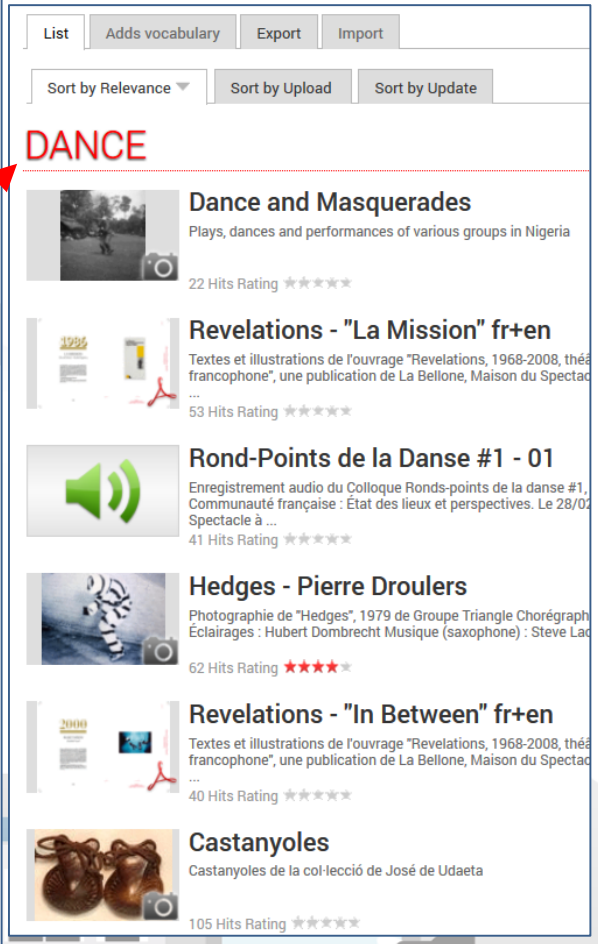
UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

Caricamento di un contenuto



Ricerca del contenuto tramite tassonomia





Eclap taxonomy: sintassi


```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <taxonomy>
    <taxonomy_name>Classification</taxonomy_name>
      <category_lev1>Genre</category_lev1> [...]
      <category_lev1>Historical period</category_lev1> [...]
      <category_lev1>Management and Organization</category_lev1>[...]
      <category_lev1>Movement and Styles</category_lev1> [...]
      <category_lev1>Performing Arts</category_lev1>
            <category_lev2>Cinema and Film</category_lev2> [...]
            <category_lev2>Dance</category_lev2>
                  <category_lev3>Ballet</category_lev3>
            <category_lev3>Ballroom</category_lev3>
            <category_lev3>Recreational</category_lev3>
            <category_lev3>Traditional</category_lev3>
            <category_lev2>Music</category_lev2> [...]
            <category_lev2>Other</category_lev2> [...]
            <category_lev2>Theater</category_lev2> [...]
      <category_lev1>Professionals</category_lev1> [...]
     <category_lev1>Subject</category_lev1> [...]
    </taxonomy>
</document>
```

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB http://www.disit.org

Mobile Medicine

http://mobmed.axmedis.org

UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

Mobile Medicine
Università degli Studi di Firenze
powered by axmedis
Log in/Create account

Search any types Advanced Search GetPlayer Help Wiki

Sviluppo: DISIT Lab

Coordinatore: prof. P.Nesi
e-mail: invia
DISIT Url: http://www.disit.dsi.unifi.it/

Mobile Medicine Support

You can download the Mobile medicine application for you iPhone and iPod directly on the Apple Store on your mobile or on web via http://itunes.apple.com/it/app/mobile-medicine/id359865882?mt=8
For the PDA windows mobile version, you can download it from the help page http://mobmed.axmedis.org/drupal/?q=en-US/help
Mobile Medicine e' una Social Network molto intuitiva (in fase di sperimentazione) e molto facile da usare per la condivisione e distribuzione multicanale di contenuti digitali multimediali e crossmediali a fini educazionali e di supporto alle emergenze ospedaliere, e di pronto soccorso. Pertanto questo help e' limitato al alcuni aspetti, per un approfondimento di consiglia la consultazione del manuale, specialmente per gli utenti che dovranno gestire gruppi di altri utenti e contenuti digitali, relazioni sociali, conoscenza medica, e forum di discussione, etc.

La versione per iPhone/iPod/iPad di Mobile Medicine Object Finder e' pronta la puoi scaricare da Apple Store, e' gratuita.
http://itunes.apple.com/it/app/mobile-medicine/id359865882?mt=8
La versione PDA Windows Mobile puo' essere scaricata dalla pagina di help:
http://mobmed.axmedis.org/drupal/?q=it/help

Per publicare un banner di Mobile Medicine nel vostro sito inserite

Mobile Medicine

Come primo passo si consiglia di leggere la pagina di Help

Buon lavoro e divertimento con Mobile Medicine,

Link Utili:

- Portale Mobile Medicine Social network
- Supporto tecnico: mobmed@dsi.unifi.it
- Registrazione di un nuovo utente
- Upload di contenuti per la pubblicazione
- Manuale di mobile medicine
- Scarico del player per PC

Languages

English

Keyword Cloud

Query Cloud

Classification

Open All | Close All

List of Terms

- Algorithms problems and resuscitation techniques
- Emergency hospital
  - Environmental emergencies and medical disaster
  - Cardiovascular Emergencies
  - Dermatological Emergencies
    - Approach to dermatology patients
    - Face and scalp
    - Infestations
    - Intertrigo
    - Generalized skin lesions
    - Hands and feet
  - Hematological and Oncological Emergencies
  - Endocrine Emergencies
  - Gastrointestinal Emergencies
  - Geriatric emergencies
  - Gynecological and obstetric emergencies
  - Infectious emergencies
  - Neurological Emergencies
  - ENT Emergencies
  - Pediatric emergencies
  - Pulmonary Emergencies
  - Emergency psychiatric disorders and psychosocial
  - Renal and genitourinary emergencies
  - Toxicological Emergencies

- Tassonomia Medica

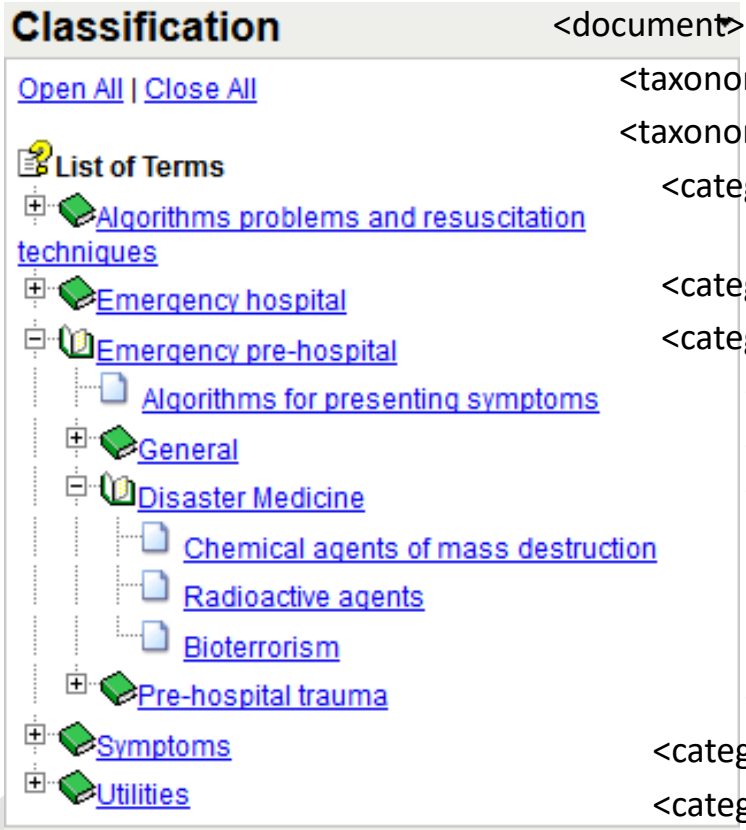
Classification

Open All | Close All

List of Terms

- Algorithms problems and resuscitation techniques
- Emergency hospital
- Emergency pre-hospital
- Symptoms
- Utilities

Mobile Medicine taxonomy: sintassi



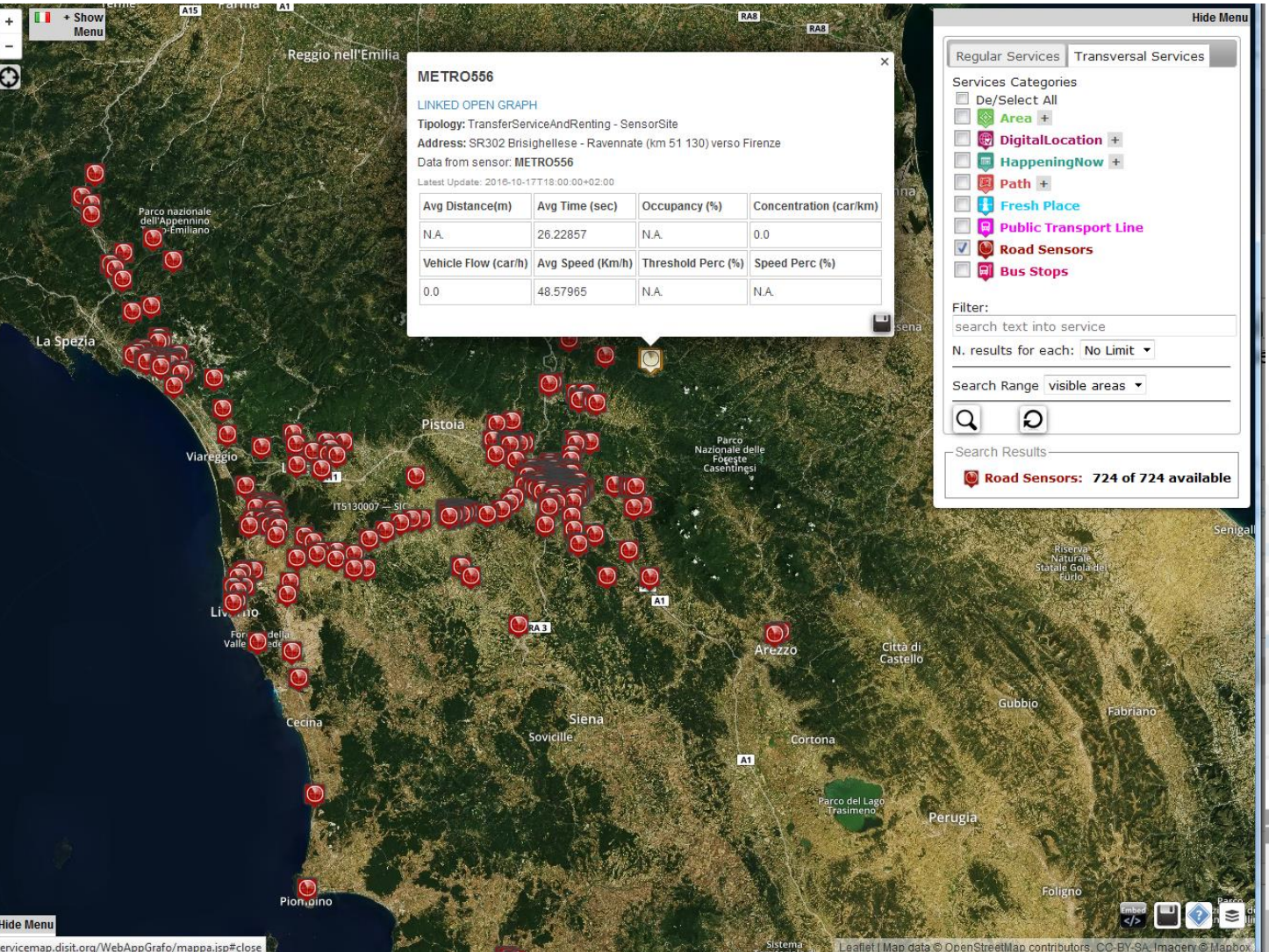

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <taxonomy>
    <taxonomy_name>Classification</taxonomy_name>
      <category_lev1>Algorithms and resuscitation
                    techniques</category_lev1> [...]
      <category_lev1>Emergency hospital</category_lev1> [...]
      <category_lev1>Emergency pre-hospital</category_lev1>
          <category_lev2>Algorithms for presenting symptoms</category_lev2>
          <category_lev2>General</category_lev2> [...]
          <category_lev2>Disaster medicine</category_lev2>
            <category_lev3>Chemical agents of mass destruction</category_lev3>
            <category_lev3>Radioactive agents</category_lev3>
            <category_lev3>Bioterrorism</category_lev3>
          <category_lev2>Pre-hospital trauma</category_lev2> [...]
      <category_lev1>Symptoms</category_lev1> [...]
      <category_lev1>Utilities</category_lev1> [...]
    </taxonomy>
</document>
```

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB http://www.disit.org

Service Map

UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

http://servicemap.disit.org http://km4city.org/



- Tassonomia usata per nascondere complessità all'utente e rendere i servizi più usabili
- Km4city ontology

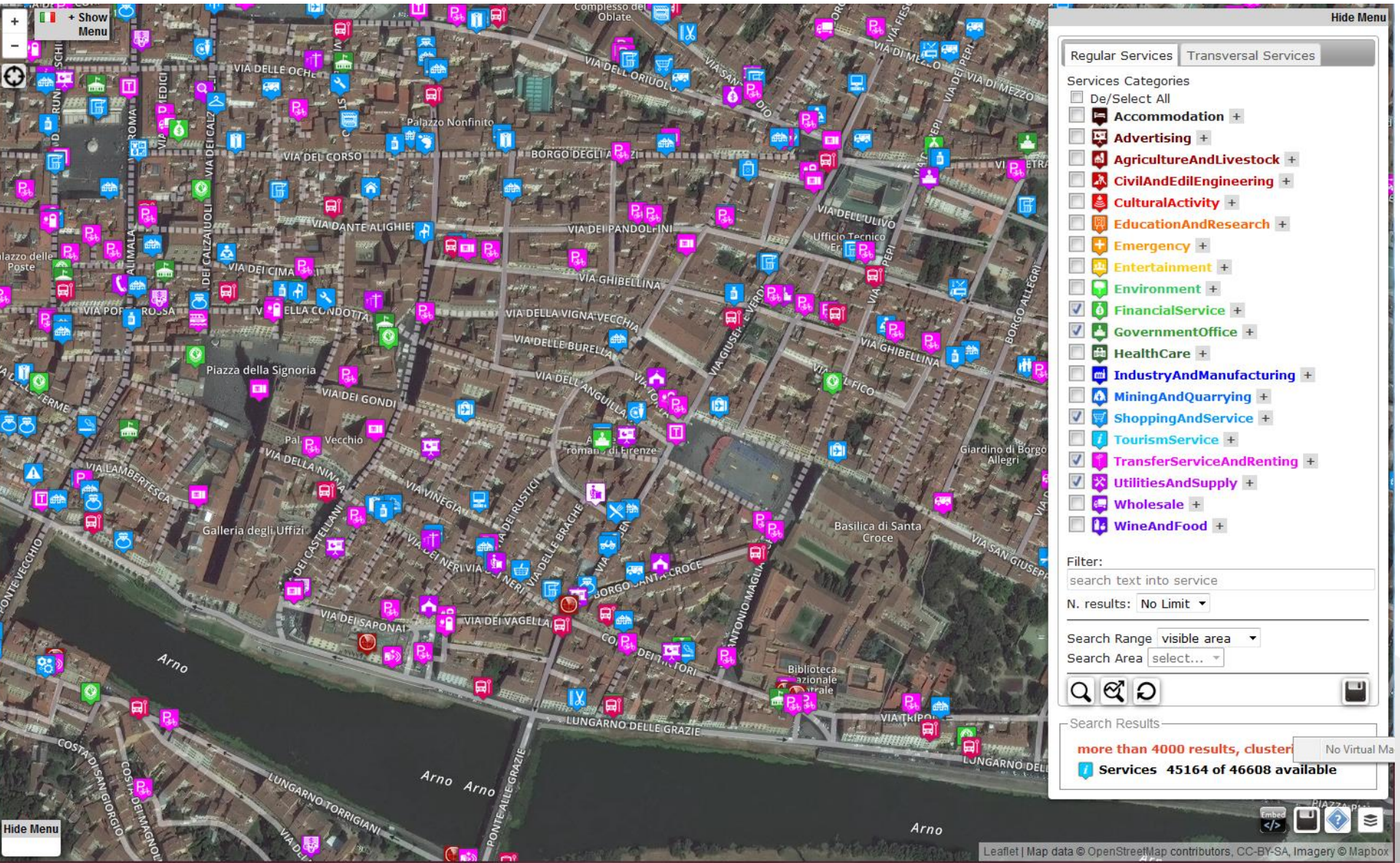
http://www.disit.org/6506

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB http://www.disit.org

Service Map

UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

http://servicemap.disit.org http://km4city.org/



Standard correlati a XML

Standard correlati a XML (1)

- XML adotta due linguaggi appositamente sviluppati per la specificazione di strutture ipertestuali complesse:
  - XML Linking Specification (XLink) specifica come costruire elementi e attributi di collegamento
  - XML Extended Pointer Specification (XPointer) specifica i sistemi di indirizzamento delle destinazioni di un link

Standard correlati a XML (2)

- La presentazione di un documento XML viene controllata da uno o più fogli di stile
- I linguaggi di stile utilizzabili con XML sono
  - **Extensible Stylesheet Language** (XSL)
  - **Cascading Style Sheet** (CSS)
- I CSS però hanno alcuni limiti:
  - Non possono cambiare l'ordine con il quale verranno visualizzati gli elementi di markup
  - Non posso effettuare operazioni logiche e computazionali
  - Non possono operare su più documenti contemporaneamente

Standard correlati a XML (3)

- Famiglia di raccomandazioni che premette di definire trasformazioni e presentazioni di documenti XML
- XSLT, Extensible Stylesheet Language for Trasformation
  - Permette di definire delle regole per trasformare un documento XML
- XSL-FO, Extensible Stylesheet Language Formatting Objects
  - Permette di definire delle regole e delle specifiche di formattazione da applicare ad un documento XML
  - Formato adatto alla stampa, pdf,…
- XPATH
  - Permette di esprimere delle espressioni per indirizzare parti di un documento XML

XSLT

Extensible Stylesheet Language for Trasformation

- Trasforma un documento XML in un nuovo documento
  - XML, HTML, PDF,….
  - Maggiore flessibilità rispetto ai CSS
- Si possono applicare diverse trasformazioni XSL allo stesso documento XML
  - Ogni trasformazione produce degli output differenti
- Netta separazione tra contenuto del documento e presentazione

XSLT

- Accesso al nodo radice
  - <xsl:template match="/">
- Accesso ad un nodo prefissato
  - <xsl:template match="nome_nodo">
- Accesso al contenuto di un elemento
  - <xsl:value-of select="nome_nodo">
- Accesso al testo contenuto in un nodo
  - <xsl:value-of select="text()"/>
- Accesso ad un determinato attributo di un nodo
  - <xsl:value-of select="@nome_attr"/>
- Accesso al commento di un nodo
  - <xsl:value-of select="comment()"/>

XSLT - esempio

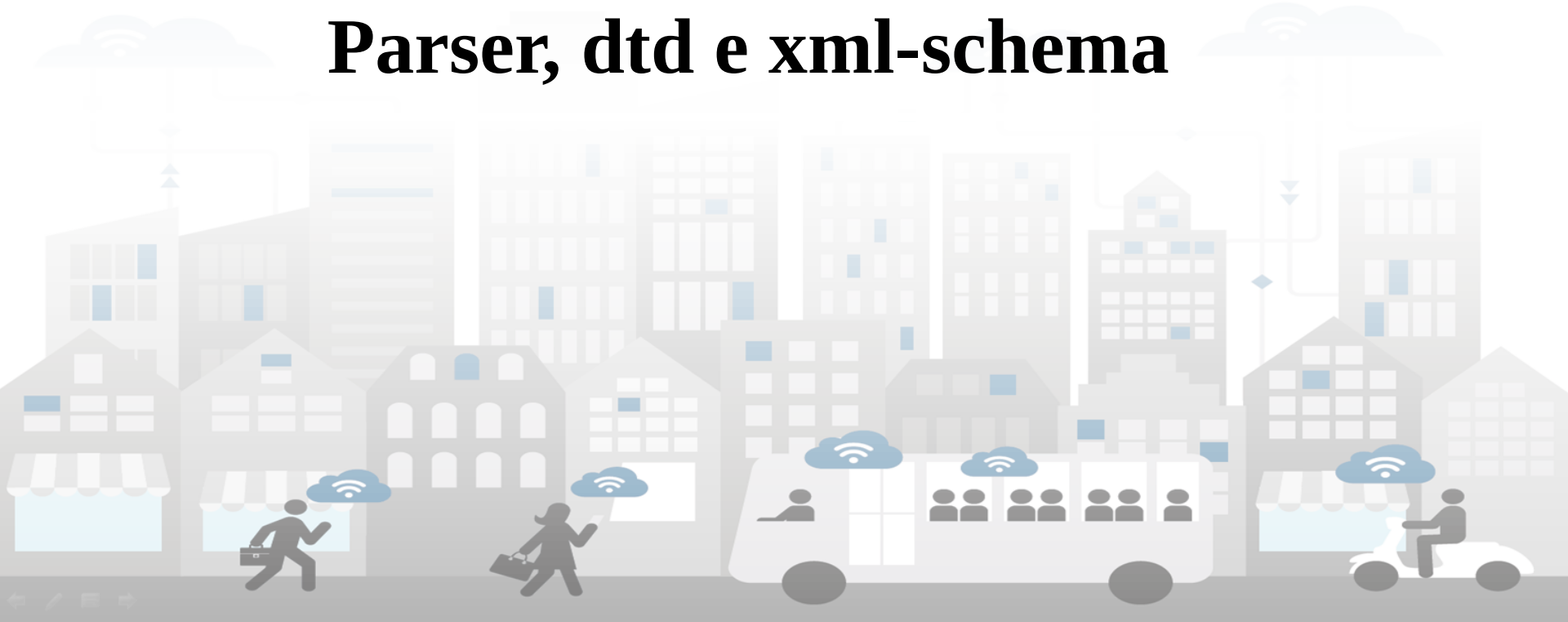
```
<?xml version="1.0" encoding="UTF-8"?><!-- Prologo XML -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/"> <html>
    <xsl:for-each select="//articolo">
      <b>Articolo : </b>
    <xsl:value-of select="@titolo"/>
      <br/>
    <xsl:for-each select="articolo/paragrafo">
      <b>- paragrafo: </b>
    <xsl:value-of select="titolo"/>
       -
    <xsl:value-of select="autore"/>
      <br/>
      </xsl:for-each>
      <br/>
      </xsl:for-each>
    </html></xsl:template></xsl:stylesheet>
```

Link utili

- http://www.w3.org/XML
- http://www.w3.org/2004/11/uri-iri-pressrelease.html.en
- http://www.ietf.org/rfc/rfc3986.txt
- http://www.w3.org/TR/xpath
- http://www.w3.org/TR/xlink/
- http://www.w3.org/TR/xptr/
- http://www.w3.org/TR/xslt

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
http://www.disit.org
UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

Extensible Markup Language (XML)
Parser, dtd e xml-schema



Parser XML

- Permettono l'analisi sintattica di un documento XML
- **Parser validanti**
  - Permettono di verificare il contenuto di un documento attraverso un file esterno
    - XML Schema
    - DTD
- **Parser non validanti**
  - Delegano il controllo della struttura del documento all'applicazione
- Un parser può esporre i propri servizi principalmente attraverso due tipi di interfacce
  - **SAX (Simple Api for XML)**
  - **DOM (Document Object Model)**

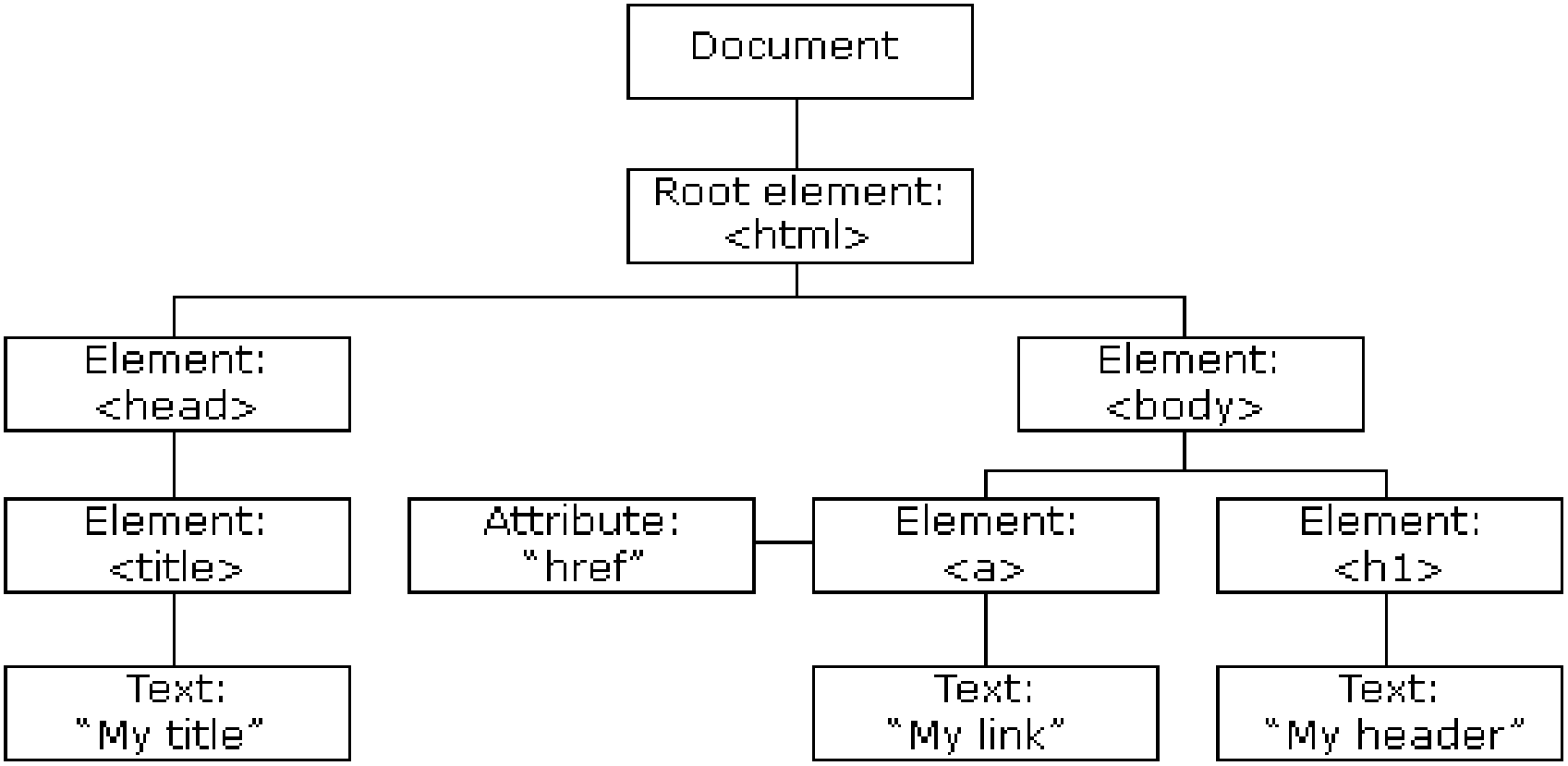
Parser SAX

- Proposto dalla mailing list XML-Dev
  - .. recepita ed implementata da diversi produttori (Microsoft, IBM, Sun, …)
- Basato su un modello di programmazione orientato agli eventi
- Un documento XML viene visto come un flusso di dati
  - Gli eventi vengono segnalati, analizzati e processati man mano che si presentano
  - Inizio e fine del documento
  - Inizio e fine di un elemento
  - Errori
  - …
- Bisogna indicare le azioni da compiere al verificarsi di un determinato evento

Parser DOM

- Standard ufficiale del W3C per la rappresentazione di documenti strutturati
- Basato su un modello di programmazione orientato agli oggetti
  - Tree-based approach per la navigazione dei documenti XML
  - Costruzione esplicita dell'albero sintattico
  - Accesso casuale per la ricerca e la modifica degli elementi
- La struttura costruita dal parser viene completamente contenuta in memoria
  - …. il rilascio della memoria è a carico del programmatore!
- Specifiche suddivise in vari livelli
  - Ogni livello può contenere moduli opzionali o obbligatori

Albero DOM


Document
Root element: <html>
Element: <head>
Element: <body>
Element: <title>
Attribute: "href"
Element: <a>
Element: <h1>
Text: "My title"
Text: "My link"
Text: "My header"

DOM vs SAX

- Applicazioni SAX
  - Richiedono poche risorse di sistema
  - Molto efficienti
  - Adatte per documenti XML di dimensioni consistenti o per dispositivi con memoria limitata
  - Non adatte se si ha la necessità di modificare il documento XML o ricordare eventi precedenti
- Applicazioni DOM
  - API semplici da utilizzare
  - Accesso casuale agli elementi del documento XML
  - Adatte per applicazioni che necessitano di navigare e/o modificare il documento
  - Richiedono un alto costo computazionale
    - …. la struttura rimane in memoria fino al rilascio….

Esempio DOM - validazione

```
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
public class DOMParser01 {
    public static void main(String args[]){
            if (args.length != 1) {
                System.err.println("Usage: java DOMParser01 filename");
                System.exit(1);
            }
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            factory.setValidating(true);
            try {
                DocumentBuilder builder = factory.newDocumentBuilder();
                builder.parse( new File(args[0]) );
                System.out.println();
                System.out.println(args[0] + " is a valid and well-formed XML file");
            } catch (Exception ex) {
                System.err.println(ex.getMessage());
                    System.err.println();
                    System.err.println(args[0] + " is NOT a valid and well-formed XML file");
            }
    }
}
```

Esempio DOM - visualizzazione

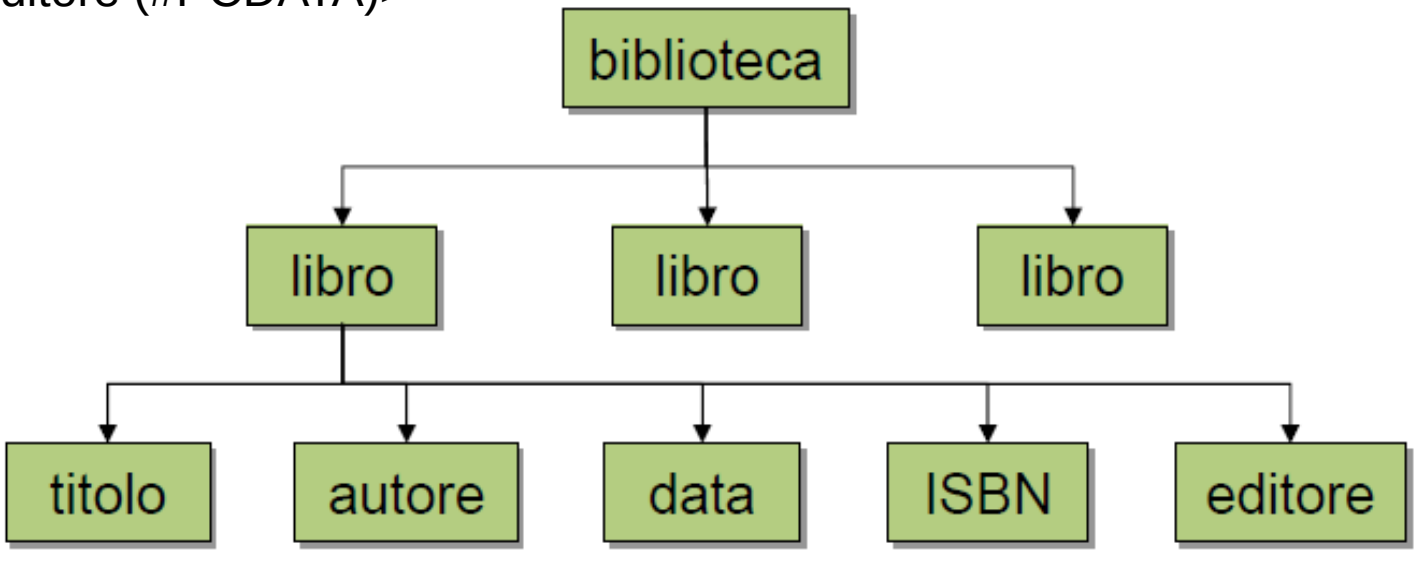
```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
public class DOMParser02{
    public static void main(String args[]){
            try {
                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                  factory.setValidating(true);
                DocumentBuilder builder = factory.newDocumentBuilder();
                Document doc = builder.parse( new File(args[0]) );
                DOMSource source = new DOMSource(doc);
                TransformerFactory tFactory = TransformerFactory.newInstance();
                Transformer transformer = tFactory.newTransformer();
                 transformer.transform(source, new StreamResult(System.out));
            } catch (Exception ex) {
                  System.err.println(ex.getMessage());
            }
    }
}
```

DTD

- Grammatica che definisce quali tag sono validi in un documento XML
- Eredità di SGML
- Per descrivere un dtd è necessario utilizzare una sintassi particolare, diversa da quella descritta per i documenti xml
- Due modi possibili di specificare un DTD
  - All'interno del documento XML
  - All'esterno del documento XML
- Inconvenienti:
  - Non viene espresso mediante XML
  - Non permette di esprimere tutti i vincoli

DTD

```
<!DOCTYPE biblioteca [
    <!ELEMENT biblioteca (libro+)>
    <!ELEMENT libro (titolo, autore+, data , ISBN, editore)>
    <!ELEMENT titolo (#PCDATA)>
    <!ELEMENT autore (#PCDATA)>
    <!ELEMENT data (#PCDATA)>
    <!ELEMENT ISBN (#PCDATA)>
    <!ELEMENT editore (#PCDATA)>
]>
```



Collegare un dtd ad un file xml

- Tramite un Dtd è possibile definire la grammatica per un linguaggio di mark-up.
- Esistono **due modi** per indicare il Dtd cui un documento XML fa riferimento:
  - **internamente al documento XML:**

```
<?xml version="1.0">
<!DOCTYPE articolo[
    ...Definizioni del Dtd...
]>
<articolo>
...Contenuto del documento XML...
</articolo>
```

  - **esternamente al documento XML**:

```
<!DOCTYPE Report SYSTEM “Report.dtd”>
<!DOCTYPE Report PUBLIC “Report.dtd”>
```

Dai DTD agli xml-schema

- L'uso dei Dtd per definire la grammatica di un linguaggio di markup non sempre è del tutto soddisfacente
- A parte il fatto che la sintassi utilizzata per definire un Dtd non segue le regole stesse di XML, i Dtd non consentono di specificare:
  - un tipo di dato per il valore degli attributi
  - il numero minimo o massimo di occorrenze di un tag in un documento
  - altre caratteristiche che in determinati contesti consentirebbero di ottenere un controllo ancora più accurato sulla validità di un documento XML
- Queste limitazioni hanno spinto alla definizione di approcci alternativi per definire grammatiche per documenti XML. Tra questi approcci, il più noto è XML Schema

Come definire un xml-schema (1)

- **Analogamente ad un Dtd**, un XML Schema è una descrizione formale di una grammatica per un linguaggio di markup basato su XML
- Tuttavia, se abbiamo bisogno di un maggiore controllo sugli elementi che possono trovarsi all'interno di uno specifico tipo di documento XML, i Dtd non risultano più sufficienti
- **A differenza di un Dtd**, che utilizza una propria sintassi specifica, un XML Schema utilizza la stessa sintassi XML per definire la grammatica di un linguaggio di mark-up. Questo è invece indice dell'estrema flessibilità di XML.
- Un XML-Schema è un documento XML che descrive la grammatica di un linguaggio XML utilizzando un linguaggio di mark-up specifico

Come definire un xml-schema (2)

- In quanto documento XML, uno XML Schema ha un root element che contiene tutte le regole di definizione della grammatica
- La **struttura generale** di uno schema XML è la seguente:

```
<?xml version="1.0"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    ... Definizione della grammatica ...

  </xs:schema>
```

- L'elemento root del documento è rappresentato dal tag **<xs:schema>**. Esso indica al parser che in questo documento saranno utilizzati dei tag definiti dal namespace standard del W3C
- I namespace rappresentano un meccanismo per identificare tag appartenenti ad una specifica grammatica. Nel nostro caso questi **tag speciali** sono caratterizzati dal prefisso **xs:**

Come definire un xml-schema (3)

- XML Schema prevede il tag <xs:element> per la definizione degli elementi utilizzabili in un documento XML, specificando nell'attributo name il nome del relativo tag.
- All'interno di ciascun tag <xs:element> si può indicare il tipo di dato dell'elemento e definire gli eventuali attributi
- Ad esempio, la seguente definizione specifica l'elemento testo che può contenere soltanto stringhe:
  - <xs:element name="testo" type="xs:string" />
- **NOTA:** Questa dichiarazione corrisponde alla seguente dichiarazione Dtd:
- <!ELEMENT testo (#PCDATA)>
- Per comprendere meglio ed apprezzare la potenza degli XML Schema, occorre analizzare nel dettaglio il concetto di tipo di dato. Esistono due categorie di tipi di dato: **semplici e complessi**

XML-schema: tipo di dato semplice

- XML Schema introduce il concetto di tipo di dato semplice per definire gli elementi che non possono contenere altri elementi e non prevedono attributi.
- Si possono usare tipi di dato semplici predefiniti oppure è possibile personalizzarli.
- Alcuni tipi di dato predefiniti sono riportati nella tabella

xs:string	Stringa di caratteri
xs:integer	Numero intero
xs:decimal	Numero decimale
xs:boolean	Valore booleano
xs:date	Data
xs:time	Ora
xs:uriReference	URL

EX: <xs:element name="quantita" type="xs:integer" />

<quantita>123</quantita> …OK

<quantita>uno</quantita> ….NO

XML-schema: tipo di dato semplice personalizzato

- Attraverso XML-Schema è possibile definire tipi di dato semplici personalizzati come derivazione da quelli predefiniti
- Se, ad esempio, si ha bisogno di limitare il valore che può essere assegnato all'elemento <quantita>, è possibile definirlo nel seguente modo:
  -
```
<xs:element name="quantita" >
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="100" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```
- In questo caso, la dichiarazione indica che l'elemento <quantita>:
  - è di tipo semplice
  - prevede una restrizione sul tipo di dato intero predefinito accettando valori compresi tra 1 e 100

XML-schema: tipo di dato complesso

- I **tipi di dato complessi** si riferiscono ad elementi che possono contenere altri elementi e possono avere attributi
- Definire un elemento di tipo complesso corrisponde a definire la relativa struttura
- Lo schema generale per la definizione di un **elemento di tipo complesso** è il seguente:

```
<xs:element name="NOME_ELEMENTO">
  <xs:complexType>
    ... Definizione del tipo complesso ...
    ... Definizione degli attributi ...
  </xs:complexType>
</xs:element>
```

XML-schema: def. tipo di dato complesso (1)

- I tipi di dato complesso sono elementi che ne possono contenere altri.
- E` possibile definire la sequenza di elementi che possono stare al suo interno utilizzando uno dei seguenti **costruttori di tipi complessi**:
  - **<xs:sequence>** Consente di definire una sequenza ordinata di sottoelementi
  - **<xs:choice>** Consente di definire un elenco di sottoelementi alternativi
  - **<xs:all>** Consente di definire una sequenza non ordinata di sottoelementi

File.xsd

```
<xs:element name="articolo">
<xs:complexType>
 <xs:sequence>
       <xs:element name="paragrafo"/>
       <xs:element name="testo"/>
 </xs:sequence>
</xs:complexType>
</xs:element>

<xs:complexType>
<xs:choice>
       <xs:element name="paragrafo"/>
       <xs:element name="testo"/>
 </xs:choice>
</xs:complexType>
```

File.xml

```
[…]
<articolo>
  <paragrafo>
      Questo è il paragrafo 1.. Introduzione…
  </paragrafo>
  <testo>
    Test effettivo contenuto nel paragrafo
  </testo>
</articolo>
[…]
<articolo>
  <testo>
    Test effettivo contenuto nell`articolo
  </testo>
</articolo>
[…]
```

XML-schema: def. tipo di dato complesso (2)

- Per ciascuno dei costruttori visti (**sequence, choice, all**) e per ciascun elemento è possibile definire il numero di occorrenze previste utilizzando gli attributi **minOccurs** e **maxOccurs**.
- Esempio: se l'elemento *testo* può essere presente una o infinite volte all'interno di un paragrafo possiamo esprimere questa condizione nel seguente modo:
  - ```
    <xs:element name="paragrafo">
      <xs:complexType>
        <xs:element name="testo" minOccurs="1"
        maxOccurs="unbounded"/>
      </xs:complexType>
    </xs:element>
    ```
- In questo caso il valore **unbounded** indica che non è stabilito un massimo numero di elementi testo che possono stare all'interno di un paragrafo
    ```

XML-schema: def. degli attributi del tipo di dato complesso

- La definizione degli attributi è basata sull'uso del tag **<xs:attribute>**, come nel seguente esempio:
  - <xs:attribute name="titolo" type="xs:string" use="required" />
- dove l'attributo use serve per specificare alcune caratteristiche come la presenza obbligatoria (**required**) o un valore predefinito (**default**) in combinazione con l'attributo value.
- Si noti che: se non si specifica esplicitamente l'obbligatorietà dell'attributo, esso è considerato opzionale

XML-schema: def. modulare degli elementi (1)

- XML Schema prevede di rendere modulare la definizione della struttura di un documento XML con la dichiarazione di tipi e elementi
- Questo contribuisce a fornire una **struttura modulare** allo schema, più ordinata, più comprensibile e semplice da modificare: un XML Schema diventa una sequenza di dichiarazioni di tipi ed elementi
- Esempio:

```
<xs:complexType name="nome_tipo">
 ...
 </xs:complexType>
```

- Il riferimento ad una dichiarazione di tipo viene fatta come se fosse un tipo predefinito, come mostrato nel seguente esempio:
- <xs:element name="nome_elemento" type="nome_tipo" />

XML-schema: def. modulare degli elementi (2)

- La possibilità di dichiarare elementi e tipi di dato implica l'esistenza di un **ambito di visibilità**
- I componenti di uno schema dichiarati al livello massimo, cioè come sotto elementi di root, sono dichiarati a livello globale e possono essere utilizzati nel resto dello schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/... ">
    <xs:complexType name="paragrafoType">
                    […]
    </xs:complexType>
    <xs:element name="articolo">
      <xs:complexType>
          <xs:sequence>
              <xs:element name="paragrafo"
                  type="paragrafoType"
                  maxOccurs="unbounded"/>
        </xs:sequence>
          <xs:attribute name="titolo" type="xs:string"
                  use="required"/>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

XML-schema: namespace (1)

- Una delle caratteristiche auspicabili nella creazione di un nuovo linguaggio è la possibilità di integrare elementi derivanti da grammatiche diverse (definite in xml-schema differenti) in modo da **riutilizzare parti di grammatiche** già definite
- Tuttavia la **composizione di linguaggi** pone almeno due tipi di problemi:
  - un documento che usa due grammatiche presenta il **problema della validazione**: a quale schema si deve fare riferimento per validare un documento XML "ibrido"?
  - due linguaggi potrebbero avere tag ed attributi con lo stesso nome, anche se utilizzabili in contesti diversi: come fare a risolvere questo tipo di **ambiguità**?
- La soluzione a questi problemi deriva dai **namespace**. Un namespace è un insieme di nomi di elementi e nomi di attributi identificati univocamente da un identificatore

XML-schema: namespace (2)

- **L'identificatore univoco** individua l'insieme dei nomi distinguendoli da eventuali omonimie in altri namespace
- Il concetto non è nuovo nell'informatica. Esempio: **definizione** dei nomi dei campi in una tabella di un database. Non è possibile avere campi con lo stesso nome all'interno di una tabella, ma è possibile avere gli stessi nomi in tabelle diverse. In questo modo si risolve l'ambiguità tra due campi omonimi facendoli precedere dal nome della tabella (il namespace)
- Se in un documento XML si utilizzano **elementi definiti in schemi diversi** abbiamo bisogno di un meccanismo che permetta di identificare ciascun namespace e il relativo XML Schema che lo definisce

Collegare un xml-schema ad un file xml

- A partire da una grammatica definita tramite un XML-Schema, è possibile sfruttare un parser XML validante per **verificare la validità** di un documento XML
- Il parser avrà bisogno:
  - del documento XML da validare
  - dello schema XML rispetto a cui effettuare la validazione
- Ci sono diversi modi per fornire al parser informazioni sullo schema da usare per la validazione. Uno di questi è il seguente:
  - inserire nel documento XML un riferimento allo schema da usare associato all'elemento root, come nel seguente esempio:

<articolo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="articolo.xsd" titolo="documento XML" >, con :

  - **xmlns:xsi** indica un URL che specifica la modalità con cui si indicherà il riferimento allo schema XML
  - **xsi:noNamespaceSchemaLocation** indica il nome e l'eventuale percorso del file contenente lo schema XML di riferimento

XML-schema: sintassi dei namespace (1)

- In un documento XML si fa riferimento ad un namespace utilizzando un attributo speciale (**xmlns**) associato al root element:
  - <articolo **xmlns**="http://www.dominio.it/xml/articolo">
- Questo indica che l'elemento articolo ed i suoi sottoelementi usano i nomi definiti nel namespace identificato dall'identificatore http://www.dominio.it/xml/articolo
- L'identificatore di un namespace può essere rappresentato da una qualsiasi stringa **univoca**. Solitamente si usa **un URI** (Uniform Resource Identifier)

XML-schema: sintassi dei namespace (2)

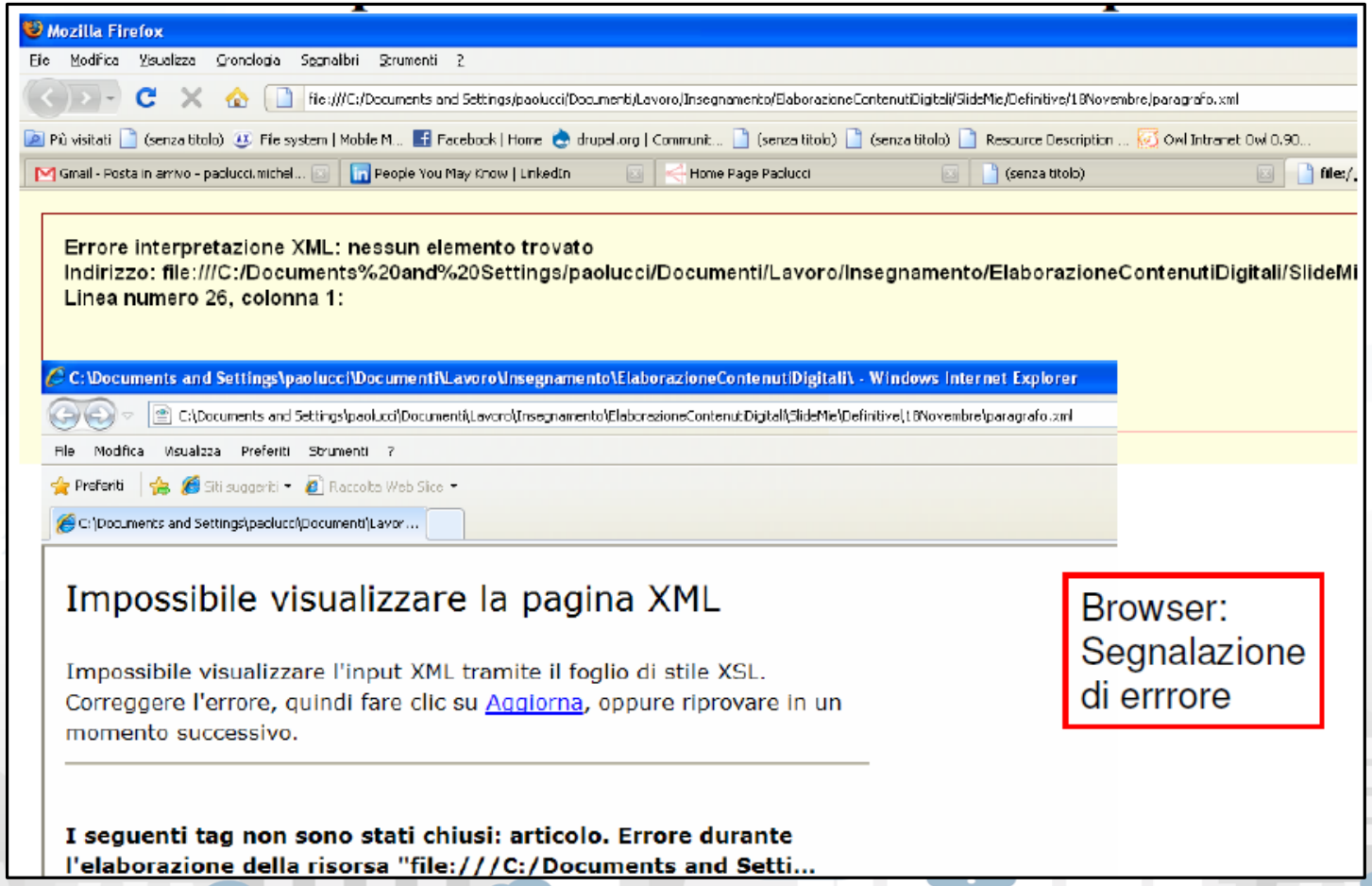
- Per mettere in relazione un namespace con il relativo XML Schema occorre dichiararlo nel root element:
  - <articolo
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:art="http://www.dominio.it/xml/articolo"
    xmlns:bibl="http://www.dominio.it/xml/bibliografia"
    xsi:schemaLocation="http://www.dominio.it/xml/articolo
    articolo.xsd"
    xsi:schemaLocation="http://www.dominio.it/xml/bibliografia
    bibliografia.xsd“>

  dove:
    - **xmlns:xsi** specifica la modalità con cui viene indicato il riferimento allo schema
    - **xsi:schemaLocation** indica il namespace ed il file in cui è definito il relativo XML Schema separati da uno spazio
- E’ possibile **combinare più namespace** facendo in modo che ciascun elemento utilizzato faccia riferimento al proprio namespace
- Si noti che quando si fa **riferimento ad un namespace**, questo riferimento vale per l'elemento corrente e per tutti gli elementi contenuti, a meno che non venga specificato un diverso namespace

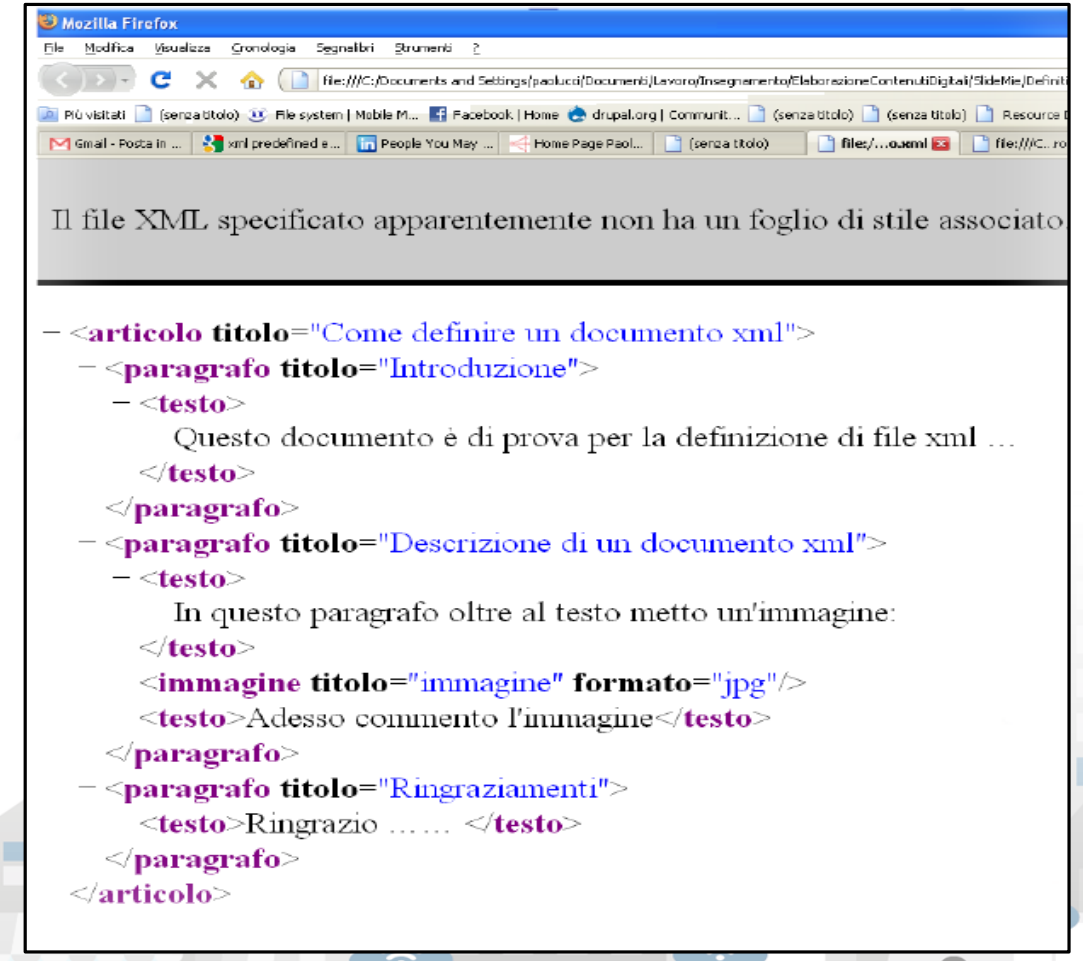
Esempi XML

- Per verificare che un file XML sia ben formato
  - Scaricare il plugin per firefox: OpenXMLVievew
  - Usare direttamente Internet Explorer
- Per la validazione
  - Xml-spy (http://www.altova.com/xml-editor/) è proprietario ma esiste una versione free per 30 gg
  - Online schema validator
    - https://www.corefiling.com/opensource/schemaValidate/

Firefox + Open XML Viewer

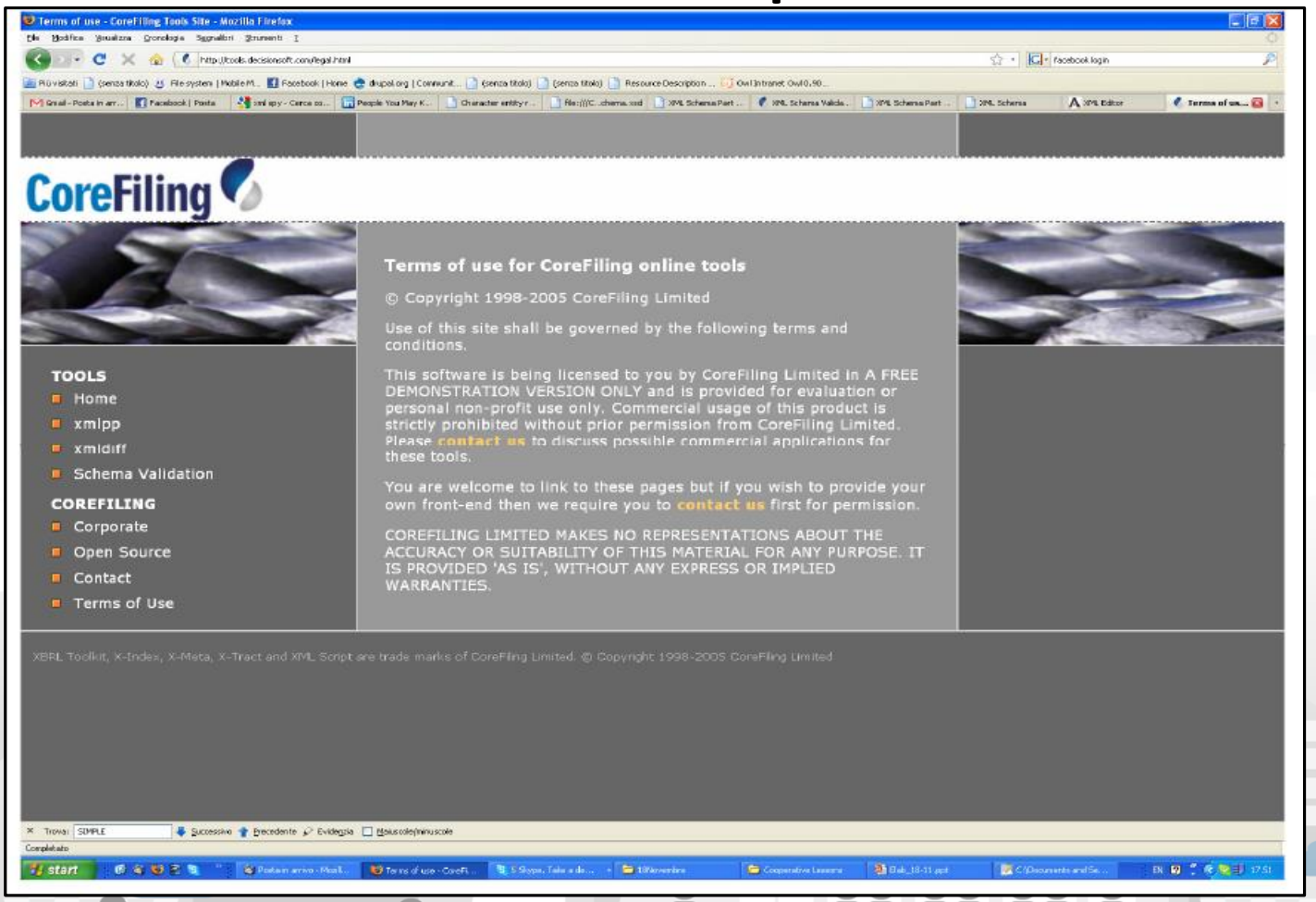


Firefox + Open XML Viewer



Verificare che il documento XML sia ben formato aprendolo sul browser

Internet Explorer



Extensible Markup Language (XML)

Applicazioni xml

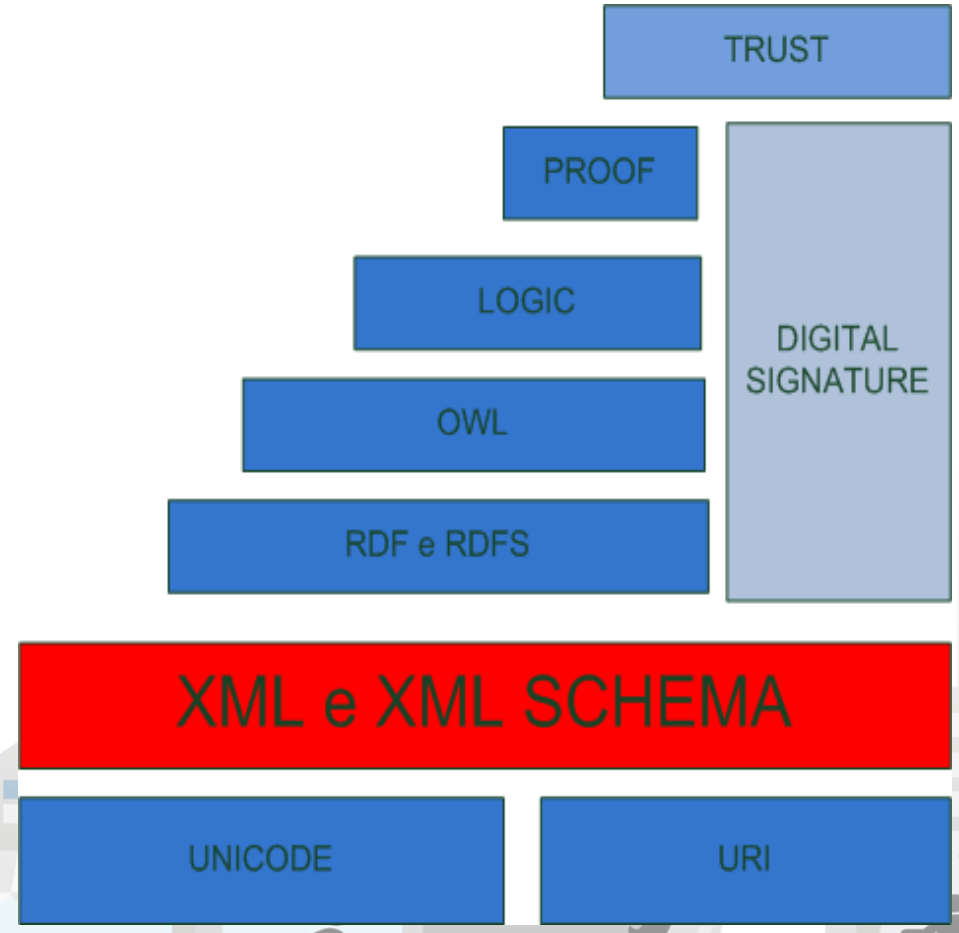
XML - Applicazioni

- Svariati contesti applicativi
  - Web Semantico
    - SKOS, RDF, OWL, OWL2
  - RSS, Really Simple Syndication
  - Scalable Vector Graphics (SVG)
  - Simple Object Access Protocol (SOAP)
  - Synchronized Multimedia Integration Language (SMIL)
  - WSDL, Web Services Description Language
  - Documenti di definizione di metadati
  - UDDI, Universal Description Discovery and Integration
  - … e molti altri ancora

XML – Applicazioni: web semantico

- Termine coniato da Tim Berners-Lee
- Estende l'attuale World Wide Web da machine-rapresentable a **machine-understandable**
  - Non lo sostituisce… ma lo migliora associando informazioni di carattere semantico alle risorse della rete
- Il Semantic Web è una ragnatela di informazioni connesse tramite relazioni **semantiche**
  - Logiche di connessione più elaborate rispetto ai collegamenti ipertestuali di HTML
  - L'idea è di generare dei documenti che possano essere letti al tempo stesso da esseri umani e da agenti automatici
- Massiccio utilizzo di linguaggi e grammatiche XML per rendere il formato machine-readable

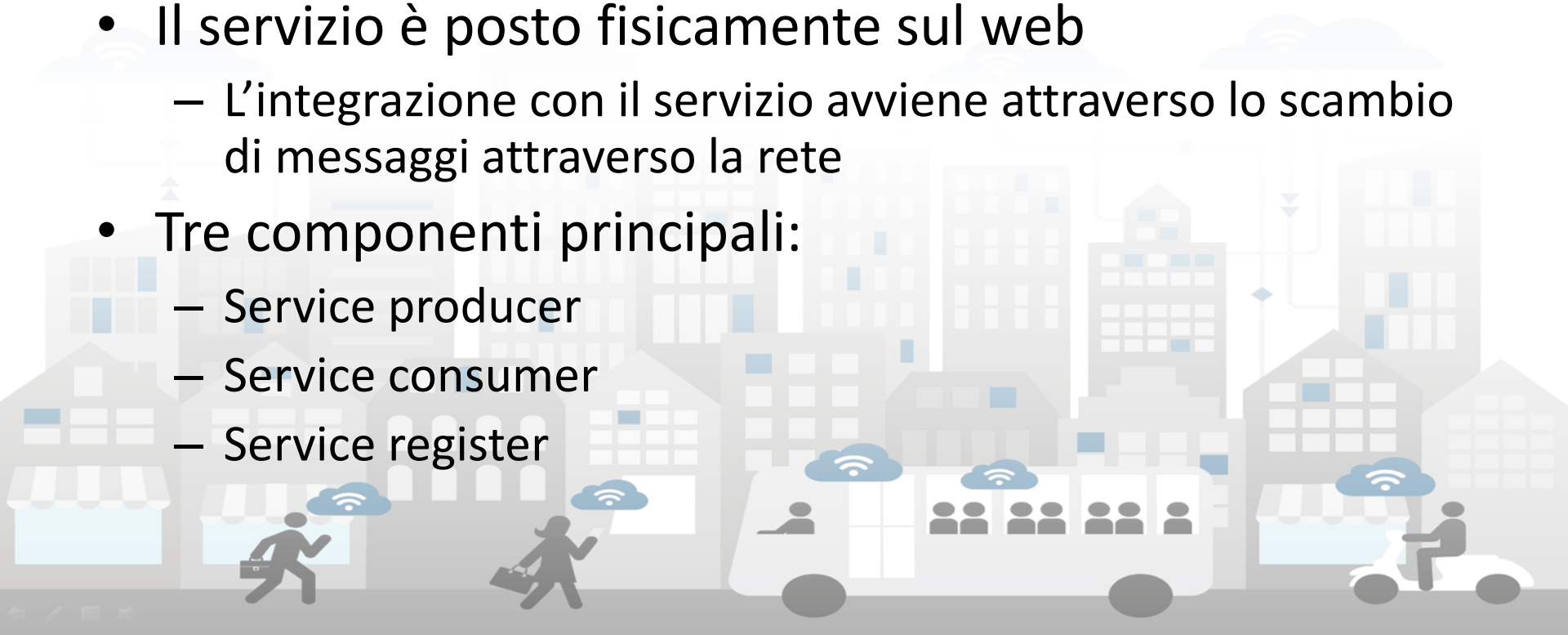
XML – Applicazioni: web semantico



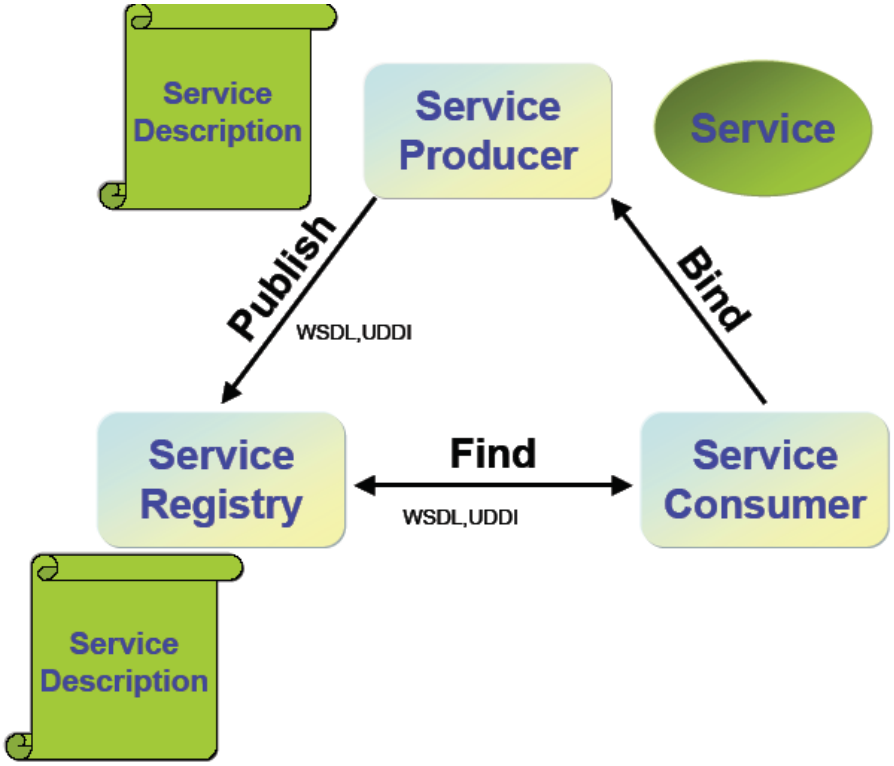
- Massiccio utilizzo di XML e Xml Schema
- Fornisce al web semantico l'interoperabilità sintattica
- Nodi concettuali
  - RDF e OWL sono basati su XML ma hanno un DTD che ne limita l'espressività e aggiunge semantica ai singoli nodi

Applicazioni XML: Web Service

- Un web service è un oggetto che offre un servizio (o alcuni servizi) ai client sulla rete attraverso una interfaccia ben definita
- Il servizio è posto fisicamente sul web
  - L'integrazione con il servizio avviene attraverso lo scambio di messaggi attraverso la rete
- Tre componenti principali:
  - Service producer
  - Service consumer
  - Service register



Applicazioni XML: Web Service



Web Service: Tecnologie

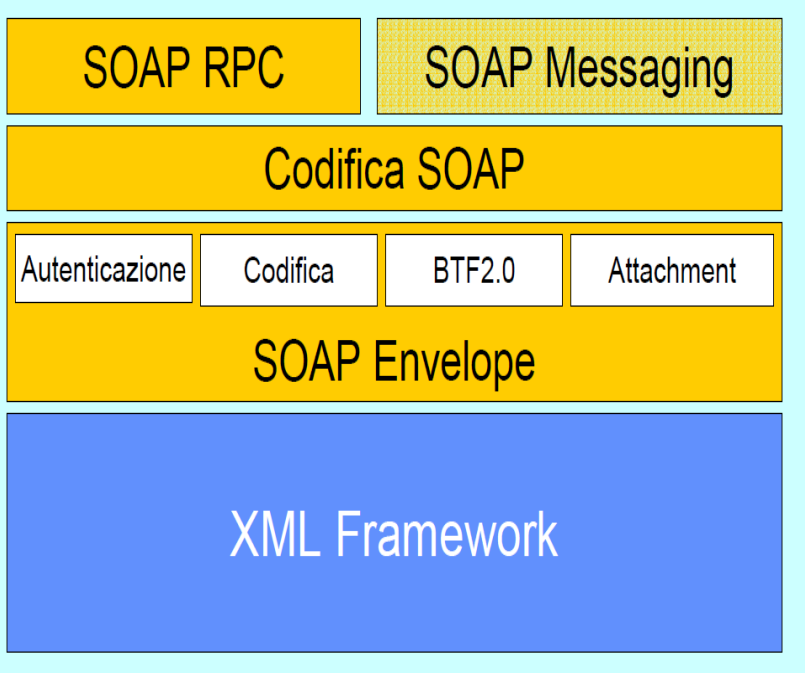
- SOAP, Simple Object Access Protocol
  - Protocollo per lo scambio di informazioni in una architettura distribuita
- WSDL, Web Services Description Language
  - Standard utilizzato per la descrizione di un servizio web
- UDDI, Universal Description Discovery and Integration
  - Standard utilizzato per la pubblicazione dei servizi in rete

- Tutte e tre le tecnologie utilizzano lo standard XML

SOAP, Simple Object Access Protocol (1)

- Protocollo che nasce nel dicembre 1999, da collaborazioni fra Microsoft e IBM
- interoperabilità di applicazioni su piattaforme diverse
- Sfrutta tecnologie già ampiamente consolidate e funzionanti:
  - HTTP
    - Per trasportare le informazioni
  - XML
    - Per esprimere le informazioni

SOAP, Simple Object Access Protocol (2)




```
<soap:Envelope
    xmlns:soap="http://schemas.xmloap.org/soap/envelope" soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">

<soap:Body
    xmlns:mybook="http://www.books.com/soapbook">
      <mybook:GetBookPrice>
        <mybook:ISBN>12-3456-789-
    </mybook:ISBN>
      </mybook:GetBookPrice>
  </soap:Body>
</soap:Envelope>
```

WSDL, Web Services Description Language

- Linguaggio basato su XML che serve a definire e descrivere web-service
- Descrive un servizio fornendo le informazioni necessarie per sviluppare client che intendono utilizzarlo seguendo la sintassi XML
- Specifica:
  - Locazione (URL del servizio)
  - Operazioni offerte
    - Descrizione astratta
    - Descrizione concreta

WSDL – Descrizione astratta

- Types
  - Per definire i tipi di dati utilizzati all'interno del documento
- Message
  - Definizione astratta dei dati scambiati tra requestor e provider, contenente i parametri di richiesta e di risposta
- Port type
  - Di solito uno per WSDL
  - Corrisponde al servizio stesso
  - Composto da una serie di elementi operation
- Operation
  - Specifica i nomi delle operazioni, gli input e gli output
  - Vengono specificati i messaggi scambiati durante l'operazione

WSDL – Descrizione concreta

- Binding
  - Fornisce i dettagli per l'implementazione delle operazioni contenute in un portType, specificando il protocollo utilizzato ed il formato dei messaggi scambiati
- Port
  - Specifica l'indirizzo di rete del servizio con cui effettuare la connessione
- Service
  - Insieme di porte correlate che spesso offrono modalità differenti di accesso alla stessa porta

WSDL – esempio

```
<message name="get_userRequest">
      <part name="id" type="xs:integer" />
</message>
<message name="get_userResponse">
      <part name="utente" type="tns:utente" />
</message>
<portType name="gestioneUtentiType">
 <operation name="getUserById">
  <input message="tns:get_userRequest" />
  <output message="tns:get_userResponse" />
 </operation>
</portType>
```

```
<binding name="gestioneUtentiBinding"
    type="tns:gestioneUtentiType">
<soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="getUserById">
  <soap:operation soapAction=”definizioneAction"
    style="rpc"/>
    <input>
     <soap:body use="encoded"
    namespace=“mynamespaceWS"
    encodingStyle="http://schemas.xmlsoap.org/soap/en
    coding/"/>
    </input>
    <output>
     <soap:body use="encoded" namespace="
    mynamespaceWS"
    encodingStyle="http://schemas.xmlsoap.org/soap/en
    coding/"/>
    </output>
 </operation>
</binding>
```

Web Service: UDDI

- Universal Description Discovery and Integration
- Base di dati indicizzata, basta su XML
- Permette alle aziende di pubblicare i propri dati e servizi offerti sulla rete internet
- Registrazione
  - Pagine gialle
    - Categorizzazione dei servizi (tassonomie di servizi standardizzate)
  - Pagine bianche
    - Contatti e dettagli dell'azienda
  - Pagine verdi
    - Informazioni tecniche sui servizi

Applicazioni XML: SMIL (1)

- Le prime specifiche (SMIL 1.0) vengono pubblicate nel 1998 come W3C recommendation
- Il 15 giugno del 2001 viene pubblicato SMIL 2.0 come working draft W3C
- Cerca di superare alcuni limiti di HTML nella gestione dell'occupazione della pagina
  - Non esiste un sistema di coordinate temporali
    - Si può fare qualche tentativo con javascript
  - Non esiste un sistema di coordinate spaziali che consenta di collocare degli oggetti nella pagina

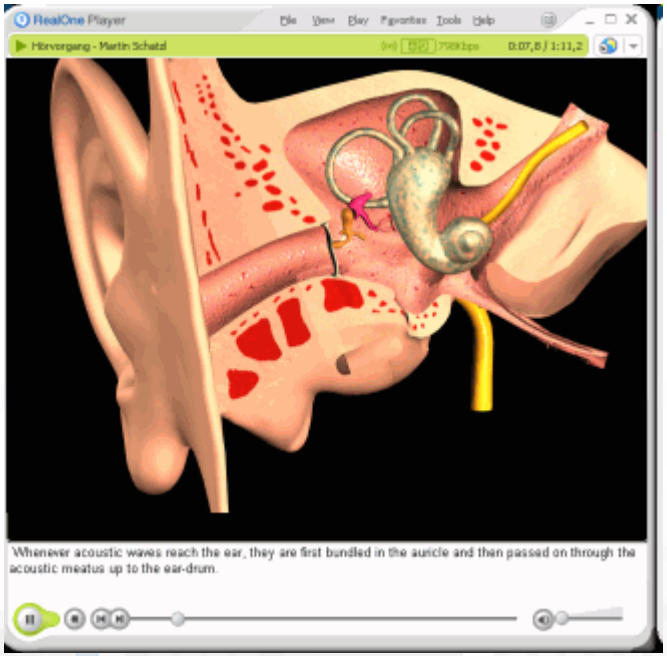
Applicazioni XML: SMIL (2)

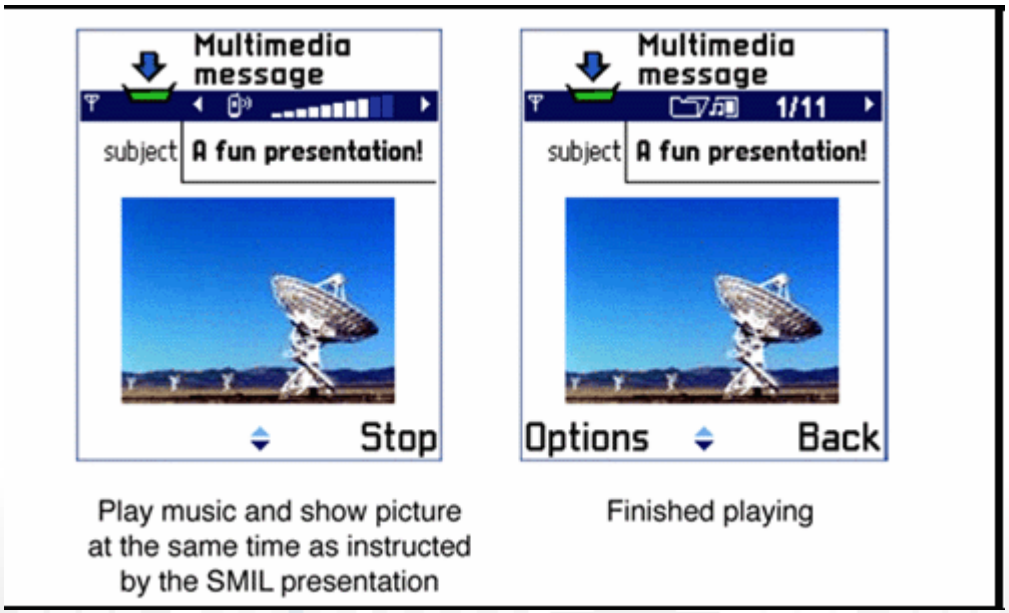
```
<smil>
<head>
<meta name= "Pippo" content= "Pluto" />
<layout>
<root-layout width="300" height="200"/>
</layout>
</head>
<body>
<img src="images/image1.jpg" dur="2s" />
<img src="images/image2.jpg" dur="1.5s" />
<img src="images/image3.jpg" dur="2.5s" />
<!-- ... -->
</body>
</smil>
```

SMIL permette:

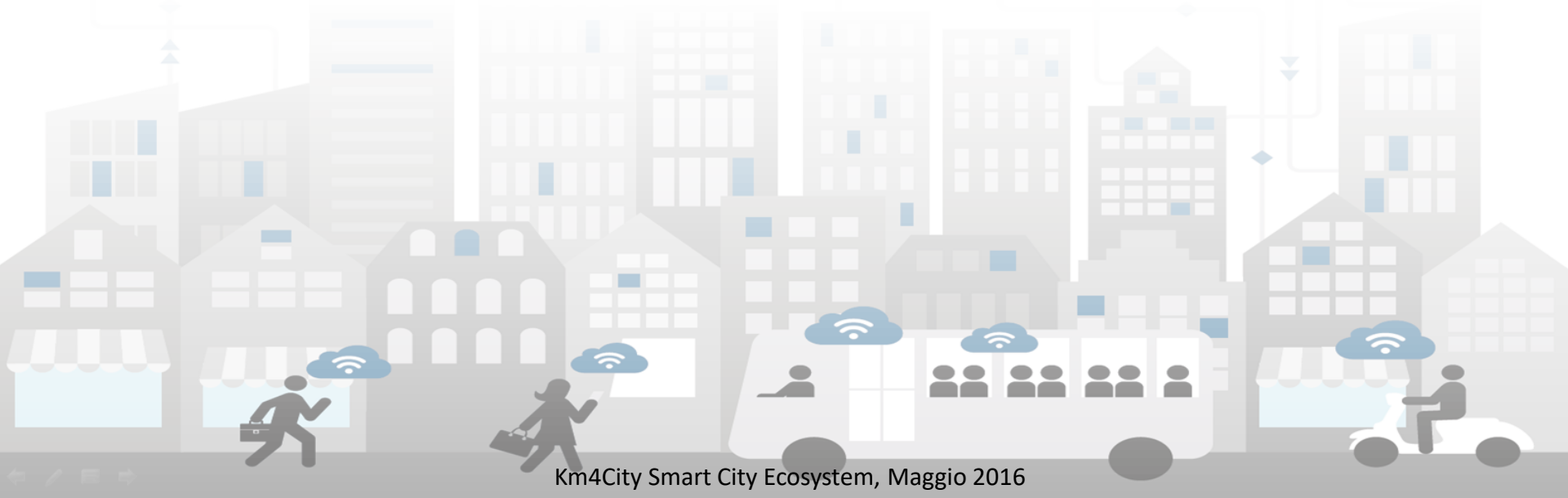
- Disporre gli oggetti multimediali in punti precisi dello schermo
- Descrivere il comportamento temporale dei diversi elementi di una presentazione
- Modificare la riproduzione secondo alcuni parametri relativi alla stazione di lavoro dell'utente
- Inserire dei link ad altre presentazioni o parti di esse

Applicazioni XML: SMIL (3)





JSON
JavaScript Object Notation

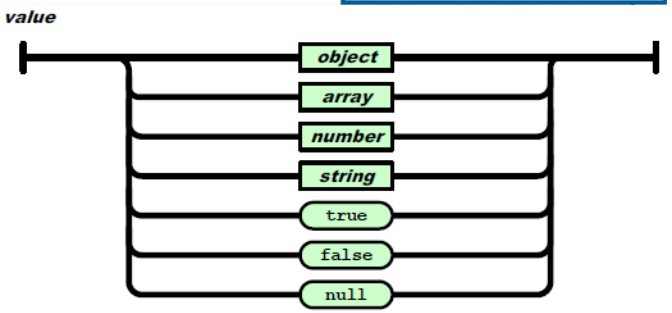


JSON (JavaScript Object Notation)

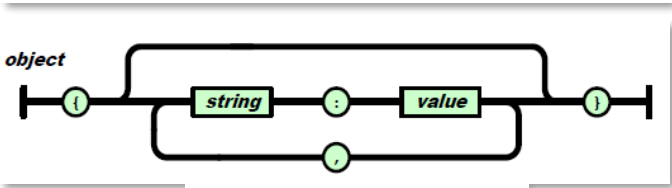
- Nasce per memorizzare dati, trasferire informazioni, rappresentare i dati in maniera da poterli trasferire tra programmi anche diversi
- Nasce dalla modalità di rappresentazioni degli oggetti in javascript
- E' una alternativa a XML per la trasmissione delle informazioni
- E' diventato uno standard: http://www.json.org
  - RFC: https://tools.ietf.org/html/draft-zyp-json-schema-03
- ESEMPIO :
  - **XML:** <nome>Mario</nome><cognome>Rossi</cognome>
  - **JSON (stringa di car. UNICODE)**: '{nome: "Mario", cognome: "Rossi"}'
  - **Javascript (oggetto)**: {nome: "Mario", cognome: "Rossi "}

JSON – Sintassi (cenni)

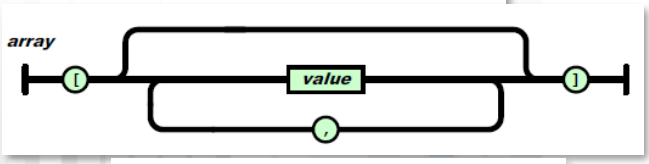
- JSON values:
  - {*object*, *array*, *number*, *string*, true, false,null}



- Caratteristiche di un Oggetto:
  - Racchiuso tra graffe
  - Contiene una serie di coppie ***nome: valore*** separate da virgola:
    - Il ***nome*** è un stringa
    - Il ***valore*** puo' essere una stringa o a sua volta un oggetto




```
{
   nome: "Mario",
   cognome: "Rossi"
}
```



- Array:
  - Racchiuso tra quadre
  - Contiene una collezione di elementi separati da virgola

```
"phoneNumber":
[
    {
      "type": "home",
      "number": "055 111111"
    },
    {
      "type": "fax",
      "number": "055 111111"
    }
]
```

Esempio JSON

JSON

```
{
    "firstName": "Andrea",
    "lastName": "Rossi",
    "age": 27,
    "address":
    {
        "streetAddress": "via S. Marta 3",
        "city": "Firenze",
        "state": "IT",
        "postalCode": "50100"
    },
    "phoneNumber":
    [
        {
          "type": "home",
          "number": "055 111111"
        },
        {
          "type": "fax",
          "number": "055 111111"
        }
    ]
}
```

JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string"
    },
    "lastName": {
      "type": "string"
    },
    "age": {
      "type": "integer"
    },
    "address": {
      "type": "object",
      "properties": {
        "streetAddress": {
          "type": "string"
        },
        "city": {
          "type": "string"
        },
        "state": {
          "type": "string"
        },
        "postalCode": {
          "type": "string"
        }
      }
    }
  },
  "required": [
    "firstName",
    "lastName",
    "age",
    "address"
  ]
}
```

JSON Validators

https://jsonformatter.curiousconcept.com

http://jsonschemalint.com/draft4/

http://jsonschema.net

- Alcuni esempi:
  - Verifica buona formazione JSON
  - Validazione schema – istanza JSON
  - Schema generator

Riferimenti / Approfondimenti

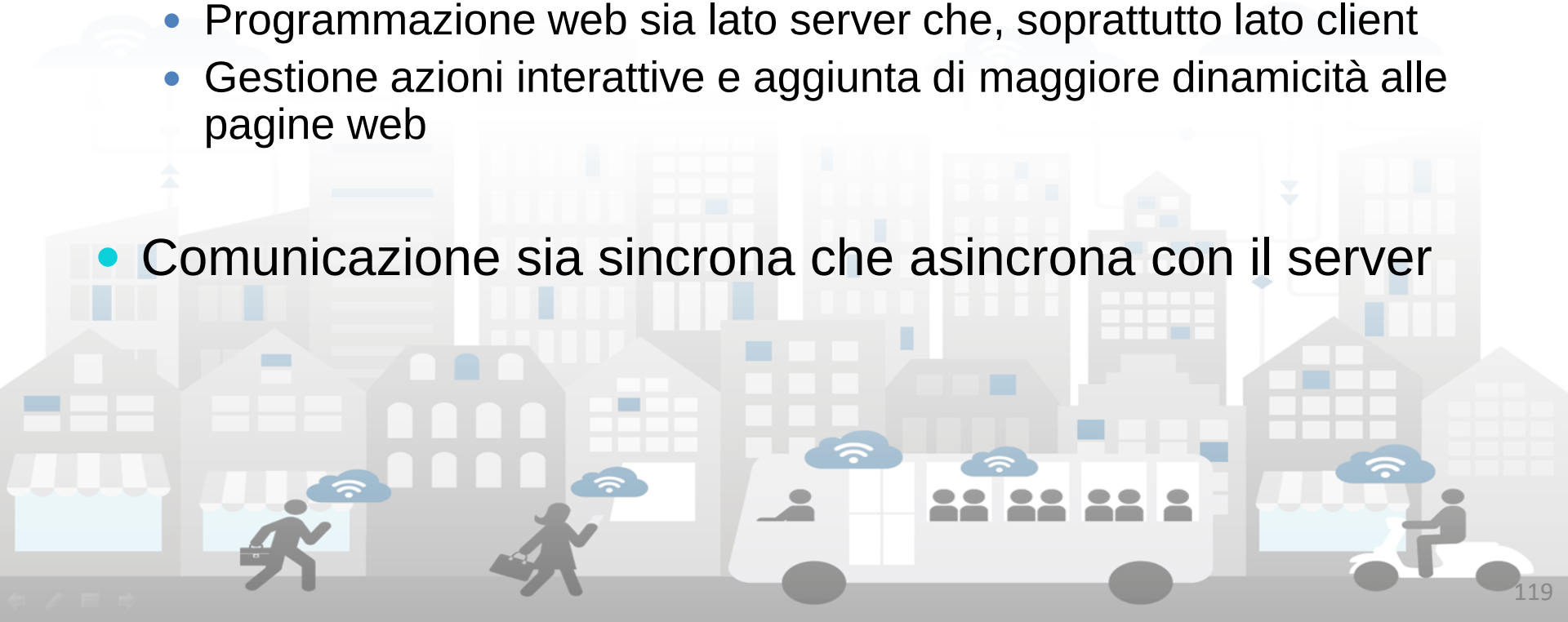
- JSON, http://www.json.org
- RFC, https://tools.ietf.org/html/draft-zyp-json-schema-03
- Alcuni JSON validators:
  - https://jsonformatter.curiousconcept.com
  - http://jsonschemalint.com/draft4/

JavaScript

- Nasce nel 1995 con il nome 'Mocha' grazie a Brendan Eich (fondatore di Netscape)
- Negli anni '90 si parla di Dinamic HTML (DHTML)
- Nel '97 nasce lo standard internazionale ECMA-262 (ECMAScript), per regolamentare le specifiche javascript, http://www.ecma-international.org
- Nel 2005 Jesse James Garrett rilascia un white paper in cui conia il nome 'Ajax' per descrivere una serie di tecnologie per creare applicazioni web, tra cui JavaScript
- Nel 2009 si ha la versione ECMAScript 5
- Giugno 2015 ECMAScript 6
- Giugno 2016 ECMAScript 7
- Giugno 2017 ECMAScript 8
- Riferimenti:
  - https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript
  - https://www.w3.org/standards/webdesign/script

JavaScript

- E' un linguaggio di scripting open source orientato agli oggetti e agli eventi
- Usato per:
  - Programmazione web sia lato server che, soprattutto lato client
  - Gestione azioni interattive e aggiunta di maggiore dinamicità alle pagine web
- Comunicazione sia sincrona che asincrona con il server

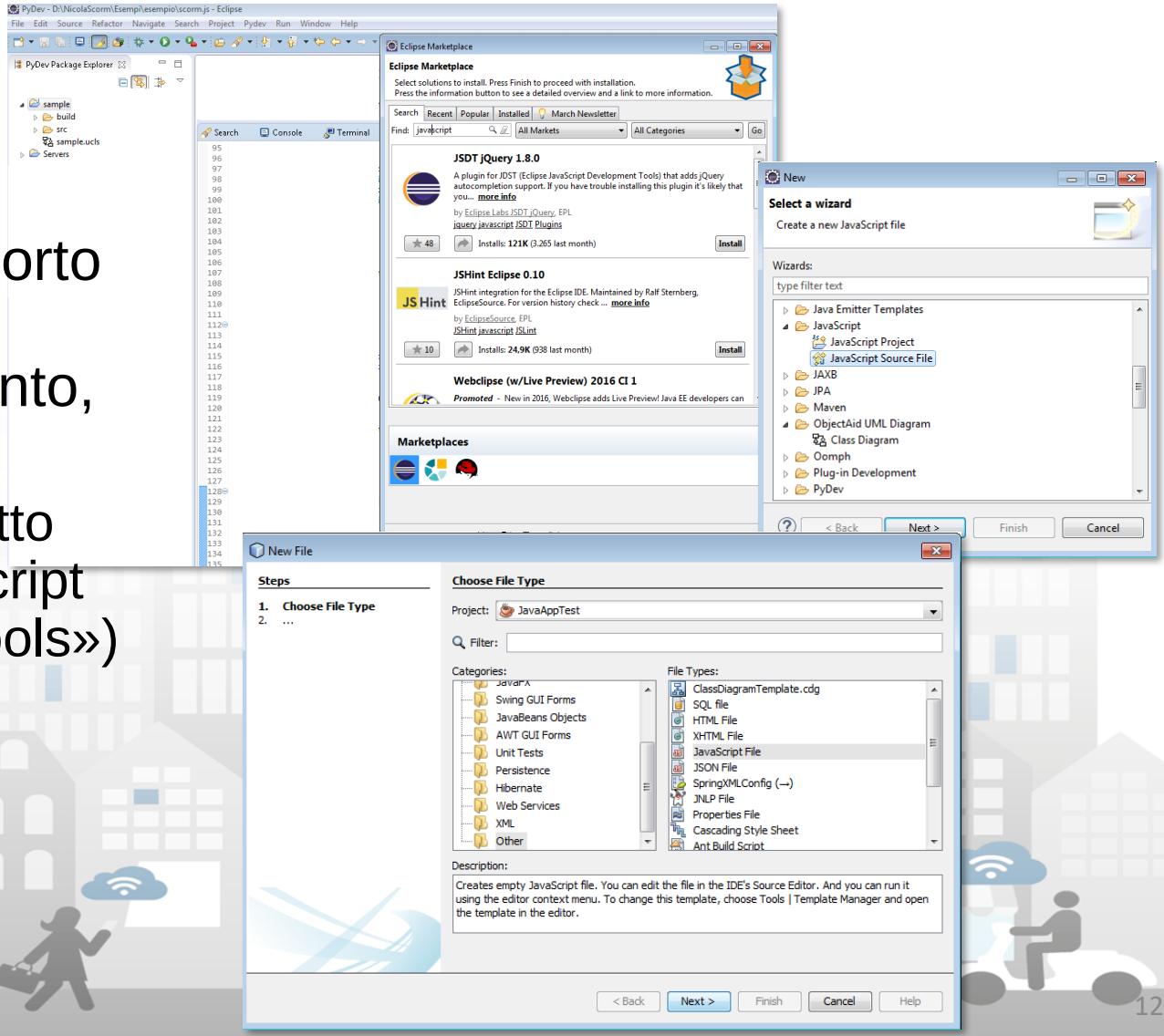


Concetto di script lato client

- Uno script lato client è un programma che affianca un documento HTML (embedded)
- Viene eseguito, sulla macchina del client, quando questo effettua il load della pagina HTML
- Azioni effettuate dagli script:
  - Modifica dinamica dei contenuti visualizzati
  - Controllo dinamico dei valori di input nei form
  - Possono essere attivati in base ad eventi effettuati dall'utente sulla pagina web (download, upload, movimenti del mouse, etc.)
  - Possono produrre elementi grafici
  - etc.
- Tipologie di Script:
  - Eseguiti una volta nel momento in cui l'utente carica la pagina
  - Eseguiti in base alla azioni effettuate dagli utenti (nel momento in cui si verificano)

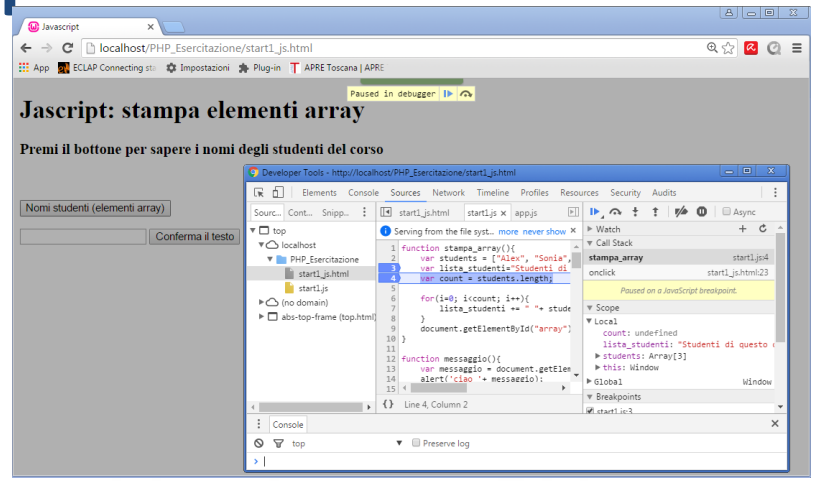
Strumenti di sviluppo

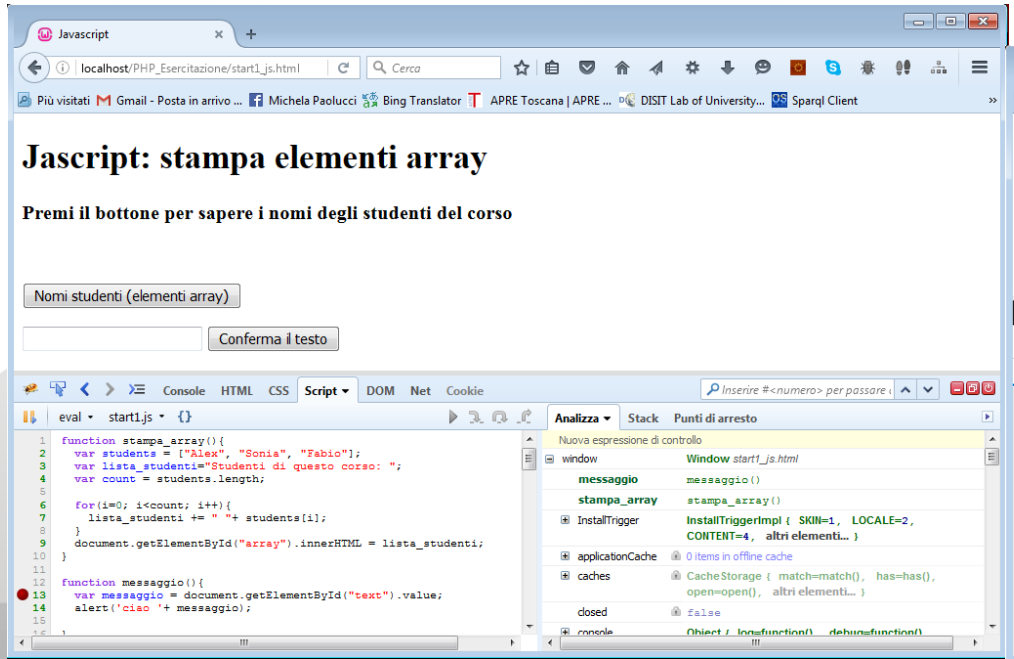
- Editor di testo
- Strumenti che forniscono il supporto a javascript (autocompletamento, etc.):
  - Eclipse (pacchetto «Eclipse JavaScript Development Tools»)
  - Netbeans
  - …

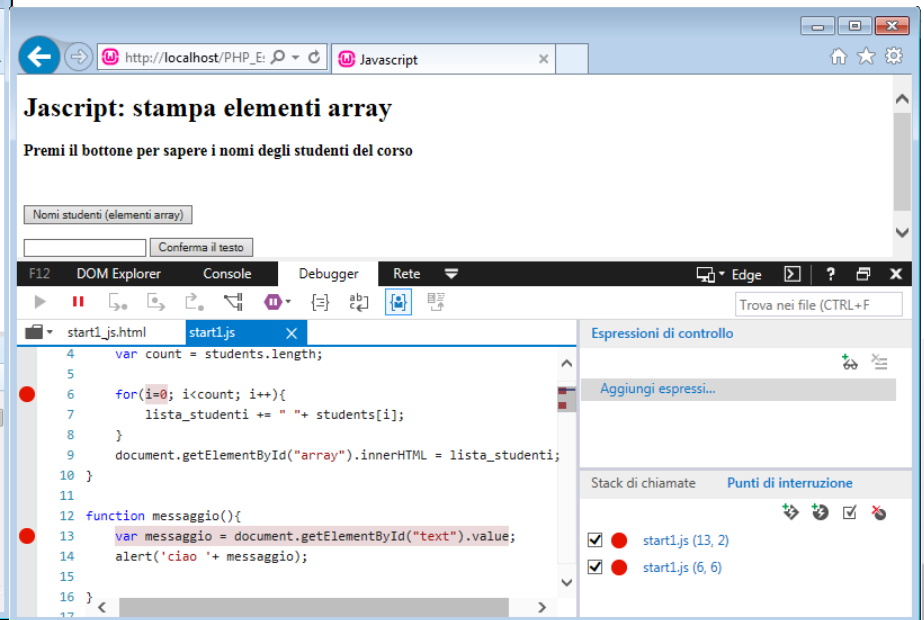


Strumenti di sviluppo nei browser

- Usare i Browser come strumenti per fare debug:
  - Crome
  - Firefox
  - Internet Explorer







Script in HTML

- E' necessario comunicare allo User Agent che tipo di linguaggio sto usando per lo script:
  - <META http-equiv="Content-Script-Type" content="type">

  CON 'type' = {"text/html", "image/png", "image/gif", "video/mpeg", "text/css", "audio/basic", etc.}
- Può essere incluso si internamente al codice HTML che esternamente (link ad un file):

file esterno

Codice js interno alla pagina html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>A document with SCRIPT</TITLE>
<META http-equiv="Content-Script-Type" content="text/tcl">
<SCRIPT type="text/vbscript" src="http://someplace.com/progs/vbcalc">
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
...some JavaScript...
</SCRIPT>
</BODY>
</HTML>
```

Esempio: Javascript in HTML ....

```
<!DOCTYPE>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
    <title>Form2</title>
    <script src="form4.js"></script>
  </head>
  <body >
    <h1>Form e jascript</h1>
    <script>alert('Benvenuto! Immetti i tuo dati!')</script>
    <form action="action4_js.php" method="POST">
      <table align="left">
       <tr>
         <td> <a href="javascript:alert('Scrivi qui il tuo Nome!')"> Il tuo Nome: </td>
         <td> <input type="text" name="name" value="" /> </td>
       </tr>
       <tr>
         <td> Il tuo Cognome: </td>
         <td> <input type="text" name="surname" value="" /> </td>
       </tr>
       <tr>
         <td> La tua e-mail: </td>
         <td> <input type="email" name="email" value ="" />  </td>
       </tr>
       <tr>
         <td> <input type="submit" value="Invia" onclick="alert('Ci stai inviando...!')" > </td>
          <td> <button type="button" onclick="conferma()">Informazioni...</button> </td>
       </tr>
      </table>
    </form>
  </body>
</html>
```

```
function conferma() {
    if (confirm("Vuoi proseguire?")) {
        alert("Controlla i dati e clicca su 'Invia'");
    }else {
        alert('Grazie per averci contattato...')
        window.location="http://www.disit.org"
    }
}
```

file esterno (riferimento assoluto o relativo)

1

Blocco di codice nella pagina HTML

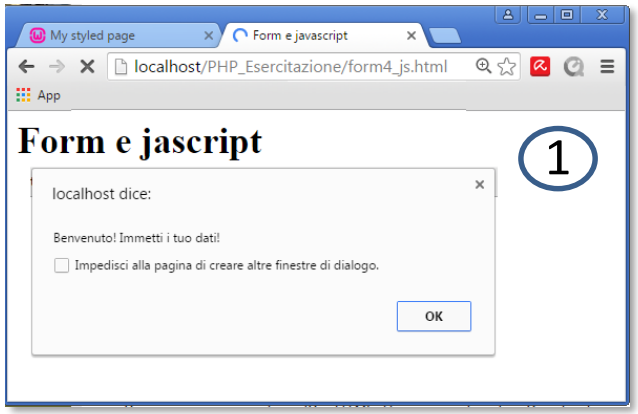
2a

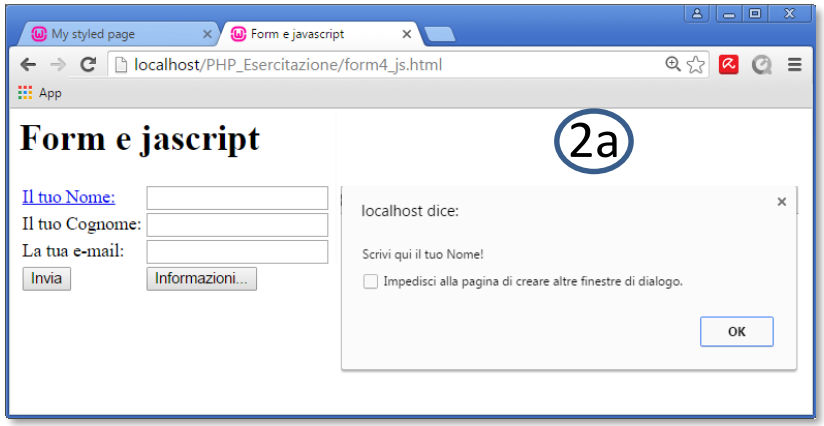
inline

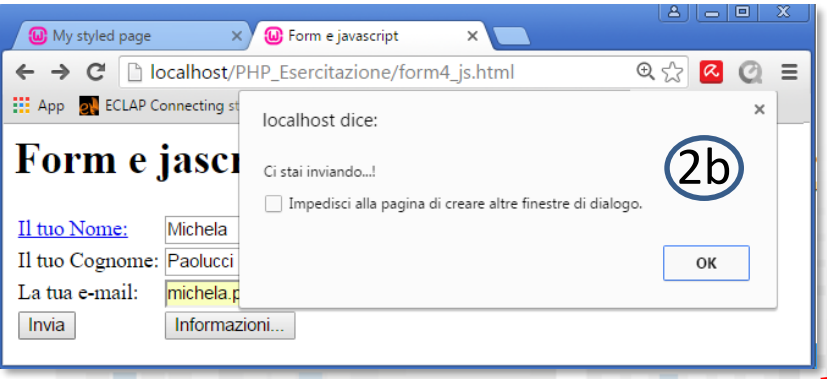
2b

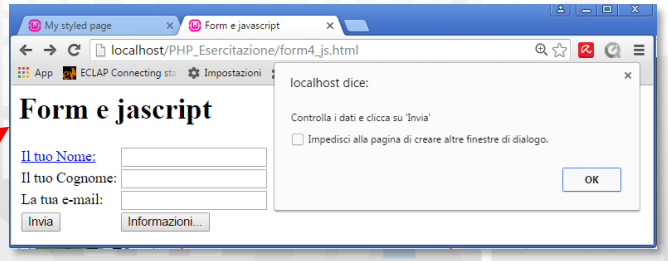
3

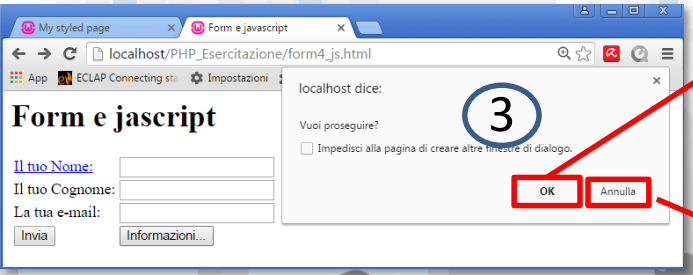
Funzione 'conferma()' definita nel file esterno

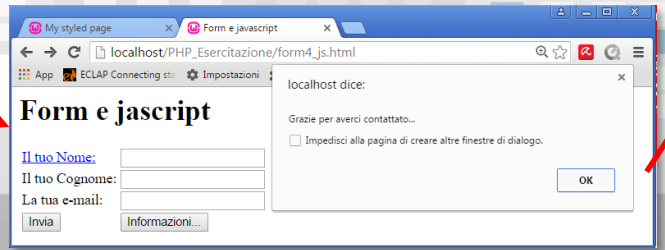

My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
Form e jascript
1
localhost dice:
Benvenuto! Immetti i tuo dati!
Impedisci alla pagina di creare altre finestre di dialogo.
OK


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
Form e jascript
2a
Il tuo Nome:
Il tuo Cognome:
La tua e-mail:
Invia
Informazioni...
localhost dice:
Scrivi qui il tuo Nome!
Impedisci alla pagina di creare altre finestre di dialogo.
OK


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
ECLAP Connecting st
Form e jasc
2b
Il tuo Nome:
Michela
Il tuo Cognome:
Paolucci
La tua e-mail:
Invia
Informazioni...
localhost dice:
Ci stai inviando...!
Impedisci alla pagina di creare altre finestre di dialogo.
OK


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
ECLAP Connecting st
Impostazioni
Form e jascript
Il tuo Nome:
Il tuo Cognome:
La tua e-mail:
Invia
Informazioni...
localhost dice:
Controlla i dati e clicca su 'Invia'
Impedisci alla pagina di creare altre finestre di dialogo.
OK


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
ECLAP Connecting st
Impostazioni
Form e jascript
3
Il tuo Nome:
Il tuo Cognome:
La tua e-mail:
Invia
Informazioni...
localhost dice:
Vuoi proseguire?
Impedisci alla pagina di creare altre finestre di dialogo.
OK
Annulla


My styled page
Form e javascript
localhost/PHP_Esercitazione/form4_js.html
App
ECLAP Connecting st
Impostazioni
Form e jascript
Il tuo Nome:
Il tuo Cognome:
La tua e-mail:
Invia
Informazioni...
localhost dice:
Grazie per averci contattato...
Impedisci alla pagina di creare altre finestre di dialogo.
OK


DISIT
Distributed Systems and Internet Technologies Lab
http://www.disit.dinfo.unifi.it
DISIT LAB OVERVIEW
Sii-Mobility

Commenti

- Esistono i seguenti metodi:
  - Commento in line
    //ecco un commento
  - Commento su più righe
    /*Commento
    su più righe
    …*/

Variabili

- Nomi delle variabili:
  - sono 'case sensitive'
  - Si possono usare le lettere (A .. Z, a .. z), I numeri (0 .. 9), il '_' (underbar, non come carattere iniziale
  - non possono contenere gli altri caratteri speciali:
    - Spazio, trattino (-), punto (.), punto interrogativo (?), dollaro ($), etc.
- Sintassi:
  - Dichiarazione (esplicita): **var** x = 10;
  - Dichiarazione implicita: x = 10;

  NOTA: se si abilita **lo strict mode,** si riceve una segnalazione nel caso in cui si usino variabili non dichiarate:
    - "use strict"
- Esempi validi:
  - **var** var1
  - **var** _variabile

spazi bianchi, ';', costanti

- Il ';' serve per determinare la fine di una espressione. Non è obbligatorio:
  - **var** x = 10;
  - **var** x = 10 /* questa dichiarazione è equivalente alla precedente*/
- Spazi bianchi:
  - Assumono un significato solo all'interno delle stringhe
- Costanti:
  - Non sono previste, si ricorre all'uso delle variabili
  - Dalla versione 6:
    - const PIGRECO = 3.14;

Tipi di dati

- Boolean
- Null
- Undefined
- Numeri
- Stringhe
- Array
- Object

Tipi di Dati: Boolean/Null/Undefined

- Il **Boolean** è il tipo di dato più semplice, può assumere due valori: True o False:
  - **var** myVariable = ***true***;
  - **var** myVariable = ***false***;

- Il tipo di dato Null prevede la notazione:
  - **var** x= ***null***;

- Il tipo di dato Undefined rappresenta un valore inesistente e prevede la notazione
  - **undefined**

Tipi di Dati: Numeri

- Esiste un unico tipo di dato:
  - Intero (se nn è specificata la parte decimale):
    - **var** negativo = -10;
    - **var** positivo = 596;
  - Decimale:
    - **var** decimale = -0.10;
  - Notazione scientifica:
    - **var** decimale 13e4;
  - Notazione Ottale/Esadecimale:
    - **var** ottale = 0134;
    - **var** esadecimale = 0x123;
  - Valori speciali:
    - Infinity | -Infinity

Tipi di Dati: Stringhe

- Tipo di variabile che contiene testo.
- Si hanno due modalità:
  - Tra apici (‘testo’), in questo caso se si vuole inserire un apice nella stringa, è necessario farlo precedere da backslash (\’) , il carattere backslash può essere inserito raddoppiandolo(\\)
    - **var** stringa = ‘testo dentro ad un file \’javascript\’… ’;
  - Tra virgolette ("testo"), in questo caso si possono usare i caratteri speciali del linguaggio di programmazione C (\n,\r,\\,\t, …) e si può includere il contenuto di altre stringhe:
    - **var** stringa = "metto un ritorno a capo \\nNel testo ";

Tipi di Dati: Array (1)

- Un array, contiene una serie di valori accessibili tramite un indice
- Definire un array, sintassi:
  - **var** studenti = [‘nome1’, ‘nome2’];
  - **var** studenti = **new Array()**;
    studenti = [‘Anna’, ‘Claudio’, ’Simone’];
  - **var** studenti = **new Array(**‘Anna’, ‘Claudio’, ’Simone’**)**;
- Esempi:
  - **var** studenti = [,‘Anna’, ‘Claudio’,];
    - si crea un array di quattro elementi in cui il primo e l’ultimo sono di tipo ‘undefined’
  - **var** array_tipi_diversi = [‘stringa’, 123, true, null];
  - **var** array_in_array = [‘stringa’, [1, 23, 4], null];
    - **var** elemento = array_in_array[2][3]; // vale 4
  - **var** matrice = [[1,2,3],[4,5,6],[7,8,9]]; //matrice 3x3
    - **var** elem = matrice [2][3]; //elem ha valore 6;

Manipolazione degli array: (1)

- Alcune proprietà/metodi:
  - (P) *length* - Restituisce la dimensione dell'array
    - **var** studenti = ['Anna', 'Claudio', 'Simone'];
    - **var** dimensione = studenti.length;
    - // risultato: dimensione=3;
  - (M) *concat* – Esegue la concatenazioni di due array
    - **var** studenti = ['Anna', 'Claudio', 'Simone'];
    - **var** nuovi = ['Mauro', 'Lara'];
    - **var** tutti = studenti.concat(nuovi);
    - //risultato: tutti= ['Anna', 'Claudio', 'Simone', 'Mauro', 'Lara']

Manipolazione degli array: (1)

- (M) *push* – Aggiunge un elemento in coda all'array
  - **var** studenti = ['Anna', 'Claudio', 'Simone'];
  - studenti.push('Lara');
  - //risultato: studenti = ['Anna', 'Claudio', 'Simone', 'Lara'];

- (M) *pop* – Rimuove un elemento dalla coda
  - **var** studenti = ['Anna', 'Claudio', 'Simone'];
  - studenti.pop();
  - //risultato: studenti = ['Anna', 'Claudio'];

Manipolazione degli array: (2)

- (M) *shift* - Rimuove il primo elemento dell'array
  - **var** studenti = ['Anna', 'Claudio', 'Simone'];
  - studenti.shift();
  - //risultato: studenti = ['Claudio', 'Simone'];
- (M) *reverse* – Inverte l'ordine degli elementi di un array
  - **var** studenti = ['Anna', 'Claudio', 'Simone'];
  - studenti.reverse();
  - //risultato: studenti = ['Simone', 'Claudio', 'Anna', 'Mauro', 'Fabio'];
- (M) *slice* – Seleziona gli elementi di un array in base alla posizione
  - **var** studenti = ['Simone', 'Claudio', 'Anna', 'Mauro', 'Fabio'];
  - studenti.slice(1,4);
  - //risultato: studenti = ['Claudio', 'Anna', 'Mauro'];

Tipi di Dati: Object (1)

- Anche gli oggetti sono variabili
- Sintassi:
  - var object_void = {};
  - var object = {proprietà1: "valore1, …, proprietàN: "valoreN" };
- Nomenclatura:
  - I nomi delle proprietà SE racchiusi tra doppi apici NON hanno restrizioni come quelli delle variabili
- Esempi:
  - var studente = {nome: "Andrea"; corso: "Sistemi Distribuiti"; AA: "2016"};
  - var stud = {"nome-stud" : "Andrea", "corso.ing" : "Sistemi Distribuiti", "AA/" : "2016"};

Tipi di Dati: Object (2)

- Si possono creare oggetti annidati:
  - var persona= {
    nome: "Andrea",
    cognome: "Rossi",
    indirizzo: { //oggetto composto da altre proprietà
    via: "S. Marta",
    numero: "3",
    città: "Firenze"
    }
    }
- A differenza degli array, gli indici sono le proprietà:
  - var nome_persona = persona["nome"];
  - var cognome_persona = persona.cognome;

Tipi di Dati: Object (3)

- Si possono creare metodi relativi agli oggetti:
  - var persona= {

```
nome: "Andrea",
cognome: "Rossi",
indirizzo: { //oggetto composto da altre proprietà
        via: "S. Marta",
        numero: "3",
        città: "Firenze"
} ,
nomeCognome: function(){
        return "nome e cognome: "  + persona.nome +
        persona.cognome;
}
}
```

- Richiamo il metodo:
  - var nome_cognome = persona.nomeCognome();

Tipi di Dati: Object (4)

- Costruttore:

```
function person(firstName, lastName, age, eyeColor) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.eyeColor = eyeColor;
        this.changeName = function (name) {
                        this.firstName = name;
        }
}
```

- Richiamo il metodo:
  - `var myMother = new person('Maria','Tirro', '55', 'blu');`
    `myMother.changeName("Doe");`
    `console.log(myMother.firstName); // scrivo su console`

Tipi di Dati: Object (5)

- Costruttore:
  - var persona= {
    nome: "Andrea",
    cognome: "Rossi",
    saluta: function(){
    alert("Benvenuto " + nome + " " + cognome);
    }
    }

- Richiamo il metodo:
  - document.getElementById("pulsante").addEventListener("click", persona.saluta);

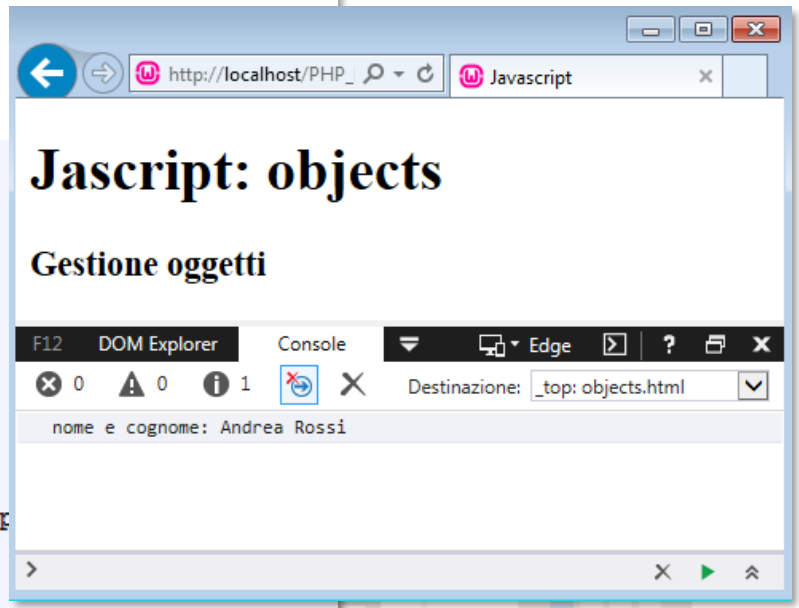
  //contesto DOM: lo associa al click dell'utente su un bottone

Tipi di Dati: Object (6)

E

```
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    </head>
    <body>
    <h1>Jascript: objects</h1>
    <h3>Gestione oggetti</h3>
    <script>
        var persona = {
            nome: "Andrea",
            cognome: "Rossi",
            indirizzo: { //oggetto composto da altre proprietà
                via: "S. Marta",
                numero: "3",
                città: "Firenze"
            },
            nomeCognome: function(){
                return 'nome e cognome: '+persona.nome + ' '+ p
            }
        }
        console.log(persona.nomeCognome());

    </script>
  </body>
</html>
```



Tipizzazione

- Se si dichiara una variabile senza specificarne il valore, essa assume il valore di default 'undefined'
- Il tipo di dato relativo ad una variabile cambiare tramite assegnazioni successive
  - **var** variabile;
  - variabile = 596;
  - variabile = "stringa";
  - variabile = true;
  - variabile = null;
- Dichiarazione di tipo iniziale:
  - **var** variabile2 = true; // tipo: boolen
- typeof: serve per verificare il tipo delle variabili:
  - "string", "boolean", "number", "function", "object", "undefined", "xml"

Operatori

- Aritmetici
- Logici
- Assegnazione
- Per Stringhe
- Relazionali
- …

Operatori Aritmetici

- Operatori binari

Operatore	Descrizione
+	addizione
-	sottrazione
/	divisione
*	moltiplicazione
%	Modulo o resto

- Operatori unari

Operatore	Descrizione
-	negazione
++	incremento
--	decremento

Operatori Logici

Operatore	Descrizione
&&	and
||	or
!	not

Operatori di Assegnazione

Operatore	Descrizione/esempio
=	x = 3+7, x assume il valore della espressione '3+7'
… ? ... : …	ternario x = *condizione* ? *val2* : *val3* x vale *val2* SE *condizione* è true, vale *val3*

Forma compatta	Descrizione
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Operatori per stringhe

- Ci sono due operatori per stringhe:
  - Il primo è l'operatore di concatenazione (‘+'), che restituisce la concatenazione dei suoi argomenti a destra e a sinistra:

```
var a = "Ciao ";
var b = a + "Mondo!"; // ora b contiene "Ciao Mondo!"
```

  - Il secondo è l'operatore di assegnazione concatenata (‘+='), che aggiunge alla fine dell'argomento sul lato destro l'argomento sul lato sinistro:

```
a = "Ciao ";
a += "Mondo!"; // ora a contiene "Ciao Mondo!"
```

Operatori & Tipi di dato (1)

- Conversioni implicite nel caso di conversione a partire da un tipo di dato a Boolean:

Tipo di dato	**Boolean**
undefined	false
null	false
numero	false se 0 o NaN (Not a Number), true in tutti gli altri casi
stringa	False se la stringa è vuota, true in tutti gli altri casi

Operatori Relazionali

Operatore	descrizione
<	minore
<=	minore o uguale
>	maggiore
>=	maggiore o uguale
==	uguale
!=	diverso
===	strettamente uguale
!==	strettamente uguale

Operatori & Tipi di dato (2)

- Conversioni implicite nel caso di conversione a partire da un tipo di dato a Numero:

Tipo di dato	**Numero**
undefined	NaN
null	0
boolean	1 se true 0 se false
stringa	intero, decimale, zero o NaN, in base alla stringa in esame

Operatori & Tipi di dato (3)

- Conversioni implicite nel caso di conversione a partire da un tipo di dato a Stringa:

Tipo di dato	**Numero**
undefined	"undefined"
null	"null"
boolean	"true" se true "false" se false
numero	"NaN" se NaN, "Infinity" se Infinity, "stringa" che rappresenta il numero negli altri casi

Istruzioni

- Uno script Javascript è costituito da una serie di istruzioni
- Una istruzione può essere un'assegnazione, una chiamata di funzione, un ciclo, …
- Le istruzioni terminano con un punto e virgola
- Le istruzioni si possono raggruppare in blocchi di istruzioni racchiudendole tra parentesi graffe
- Un gruppo di istruzioni è, a sua volta, un'istruzione

Funzioni

- Una funzione è un blocco di codice che può richiedere uno o più parametri in ingresso e può fornire un valore di uscita

- Javascript mette a disposizione numerose funzioni predefinite

Creare Funzioni

```
function <nome_funzione>(<parametri>) {
        <lista di azioni>
 }
```

- Esempio:

```
function messaggio(){
  alert('Stampo Messaggio… !');
}
```

- Le variabili definite in una funzione sono variabili locali, per accedere a variabili globali si usa l'istruzione const:
  - **const** nome = 'Chiara';

Usare le Funzioni

- Per utilizzare una funzione bisogna connetterla ad un evento nella pagina web e richiamarla (o invocarla)
- Tipi di eventi:
  - Page load
  - Interazioni con gli oggetti (elementi html, DOM) della pagina (Click su bottoni o link, movimenti del mouse, etc.)
  - …

Alcuni eventi HTML

- onload
- onclick
- onchange
- onmousemove, onmouseover, onmouseout, …
- onsubmit
- …

ESEMPIO

file.html

```
<!DOCTYPE>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    <script src="start1.js"></script>
    </head>
    <body >
    <h1>Jascript</h1>
    <p id="demo" onmouseover="stampa()" onmouseout="cancella()" > ??? </p>
    <div> <a href="javascript:alert('Benvenuto!')"/> Clicca qui</div>
  </body>
</html>
```

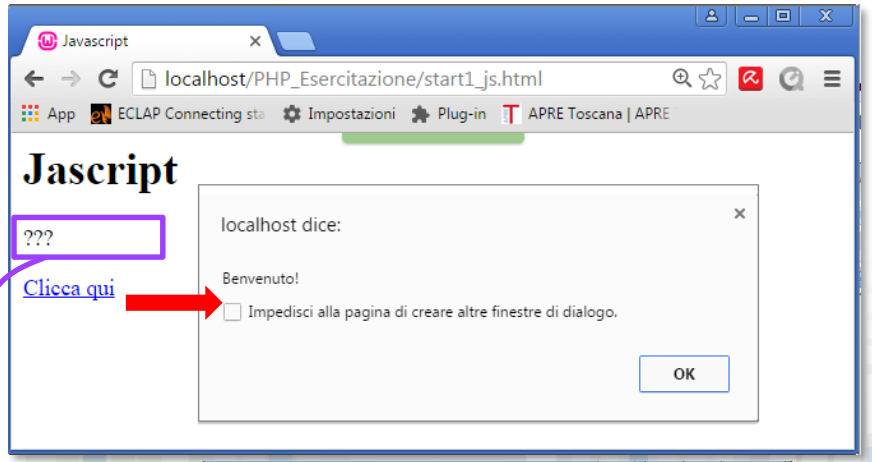
Eventi:

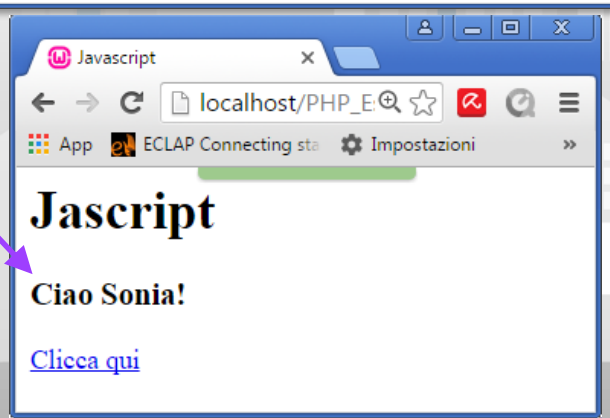
- onmouseover
- onmouseout

start1.js

```
function stampa(){
    var students = ["Alex", "Sonia", "Fabio"];
    document.getElementById("demo").innerHTML
    = "Ciao "+ students[1]+"!";
}

function cancella(){
    document.getElementById("demo").innerHTML = "???";
}
```

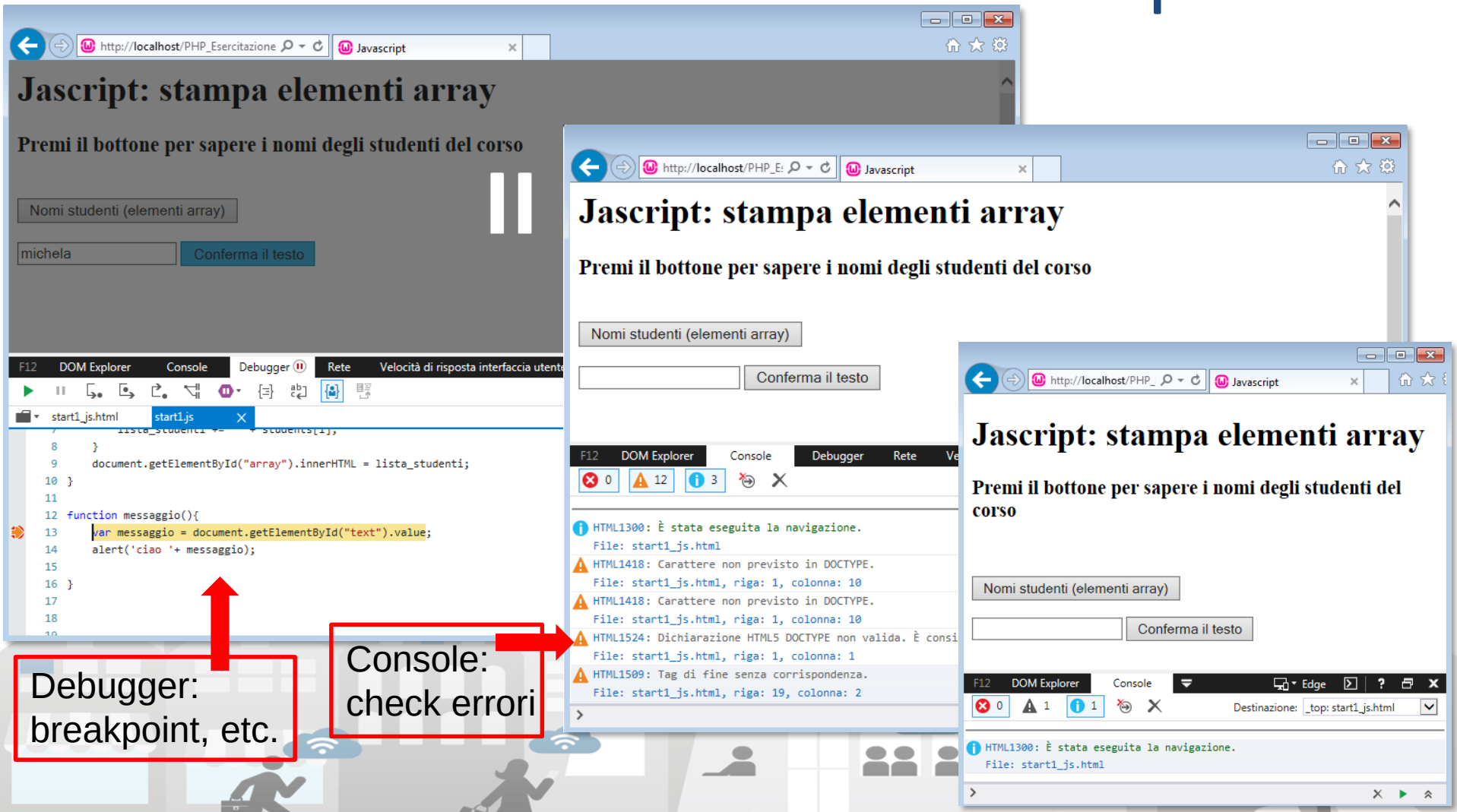




- Selettore:
  - document.getElementbyId("id")
- Proprietà:
  - innerHTML

NOTA: l'oggetto 'document', fa parte del DOM (Document Object Model) il modello a oggetti delle pagine HTML

ESEMPIO… usando Internet Explorer



Istruzioni: if else

- **Sintassi:**

```
if (condizione) {
  azione1 da effettuare;
  azione2;
}
else { //se la condizione non e'verificata
  altre azioni;
}
```

- Note:
  - Le parentesi graffe servono per raggruppare una serie di azioni
  - La clausola else { } è facoltativa, va usata nel caso ci sia una alternativa se if non soddisfa la condizione indicata fra le parentesi tonde
- Esempio:

```
if (a==b){
  alert('sono uguali');
}
else{
  alert('sono diversi');
}
```

elseif

- Sintassi:

```
if (a=='cond1'){
    alert('a vale: cond1');
  }
  elseif ('cond2') {
    alert('a vale: cond2');
  }
  ...
  else {
   alert('a vale: altro…');
  }
```

NOTE:
Si tratta di un'altra istruzione IF all'interno di un IF. Il server controlla se il primo *if* è vero, se è falso va sul *elseif*, se è falso anche questo continua con gli *elseif* fino a quando non trova una alternativa oppure l'istruzione finale *else* (non obbligatoria)

Cicli: for

Sintassi:

```
for (espressione iniziale; condizione; aggiornamento) {
        lista azioni;
}
```

Esempio:

```
for(i=0; i<students.length; i++){
        lista_studenti += " "+ students[i];
}
```

NOTA: Se la variabile non raggiunge la condizione inserita dentro il ciclo si crea un loop infinito.

Cicli: for-in (array)

Sintassi:

```
var nome_array = …
var indice;
for (indice in nome_array) {
        < azioni>
}
```

Esempio:

```
var valori = [10, 20, 30, 40]
var indice;
var totale = 0;
for(indice in valori)
        totale += valori[indice];
}
```

Cicli: for-of (array)

Sintassi:

```
var nome_array = …
var indice;
for (indice in nome_array) {
        <azioni>
}
```

Esempio:

```
var numeri = [12, 35, 20, 7, 4, 2];
var tot_num = 0;
var valore;

for (valore of numeri) {
        tot_num +=   valore;
}
```

Stampare elementi di un array in un div

- Cicli: for o while
- Eventi:
  - onmouseover
  - onmouseout
  - onclick (<button>)
  - …
- Selettore/proprietà:
  - document.getElementbyId("id").innerHTML = 'stringa' + '!';

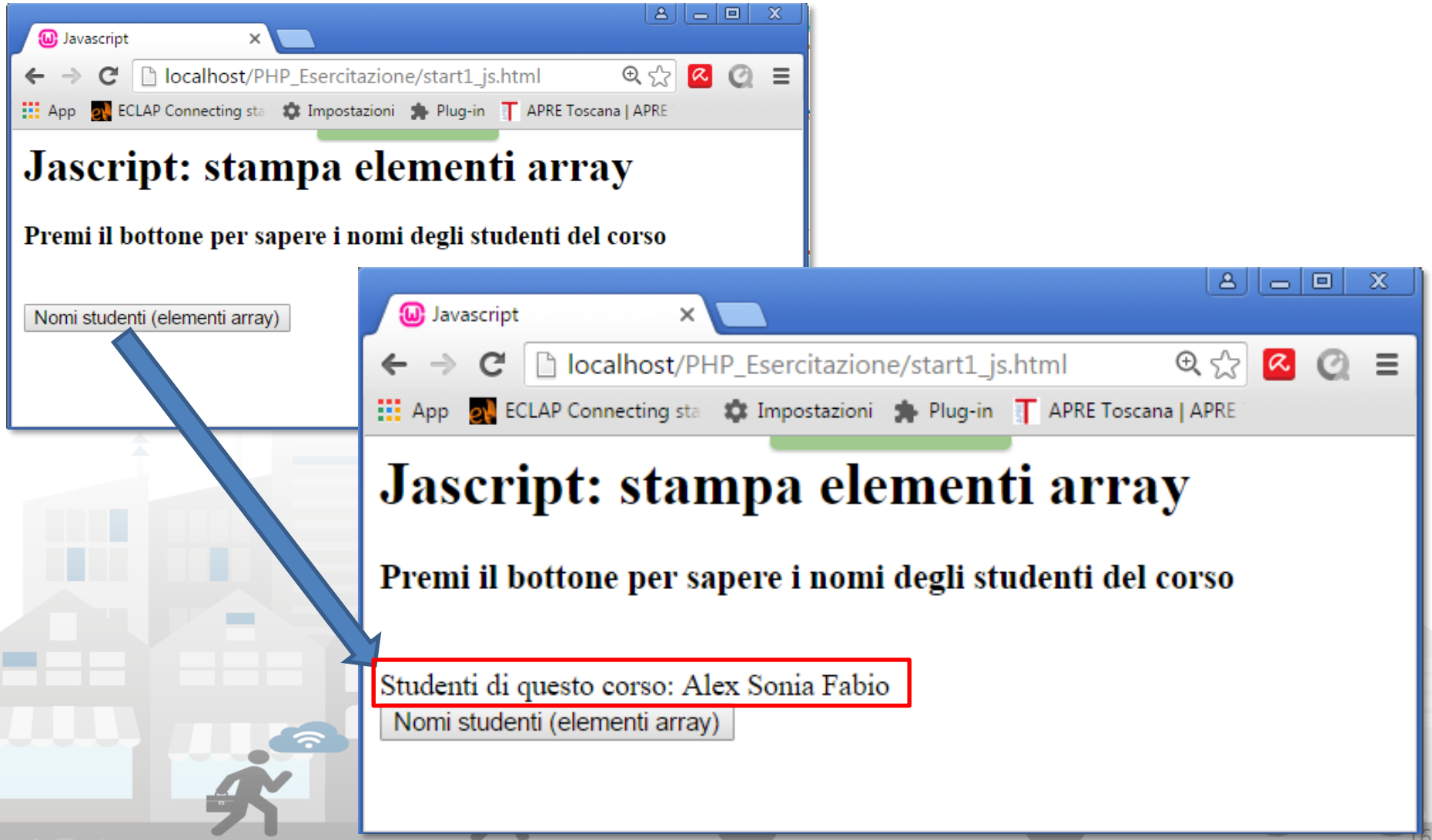
Stampare elementi di un array in un div (2)

```
<!DOCTYPE>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    <script src="start1.js"></script>
    </head>
    <body >
    <h1>Jascript: stampa elementi array</h1>
    <h3 > Premi il bottone per sapere i nomi degli studenti del corso   </h3>
    <br>
    <div id="array"> </div>
    <button onclick="stampa_array()">Nomi studenti (elementi array)</button>
  </body>
</html>
```

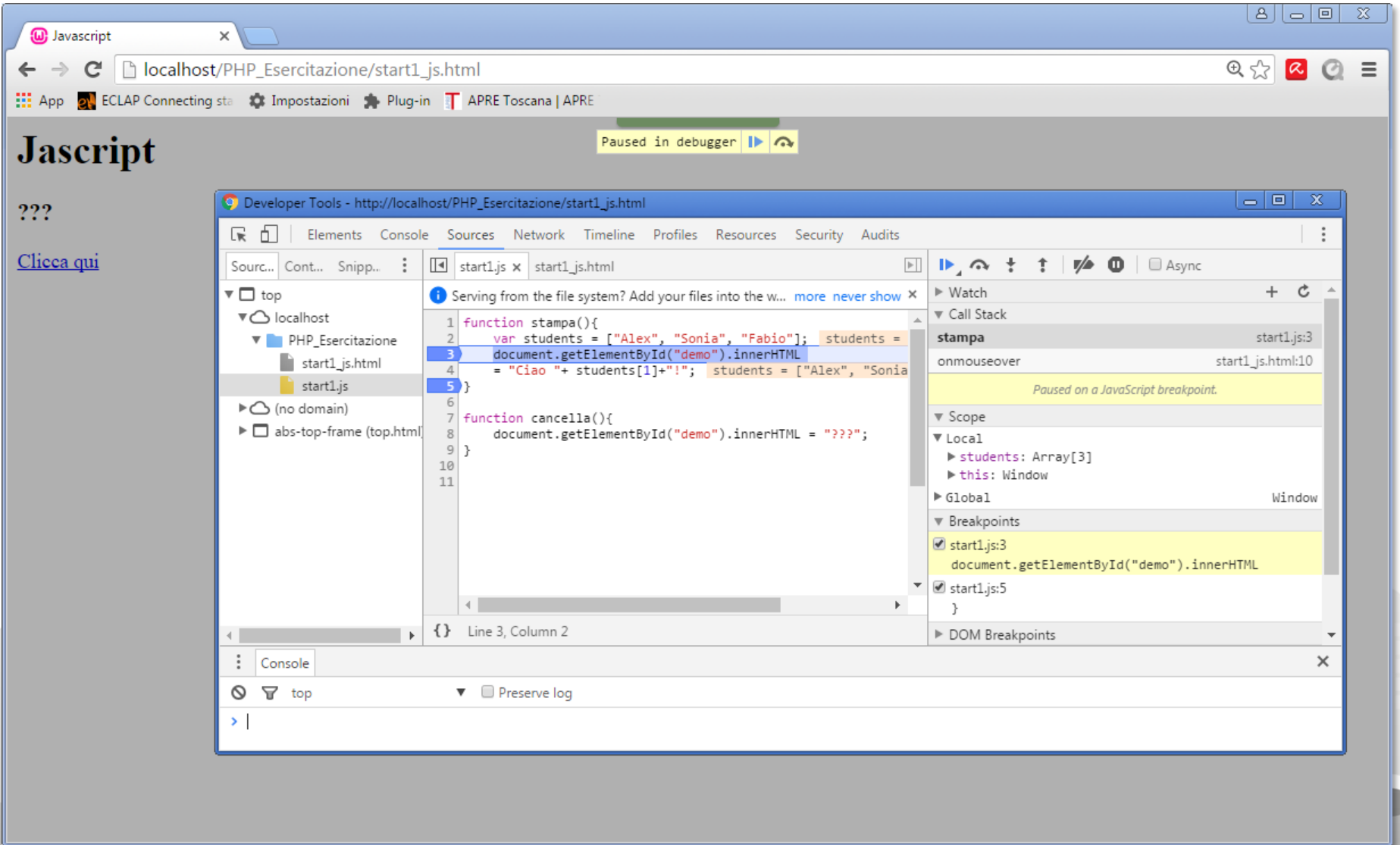
```
function stampa_array(){
    var students = ["Alex", "Sonia", "Fabio"];
    var lista_studenti="Studenti di questo corso: ";
    var count = students.length;

    for(i=0; i<count; i++){
        lista_studenti += " "+ students[i];
    }
    document.getElementById("array").innerHTML = lista_studenti;
}
```

Stampare elementi di un array in un div (3)

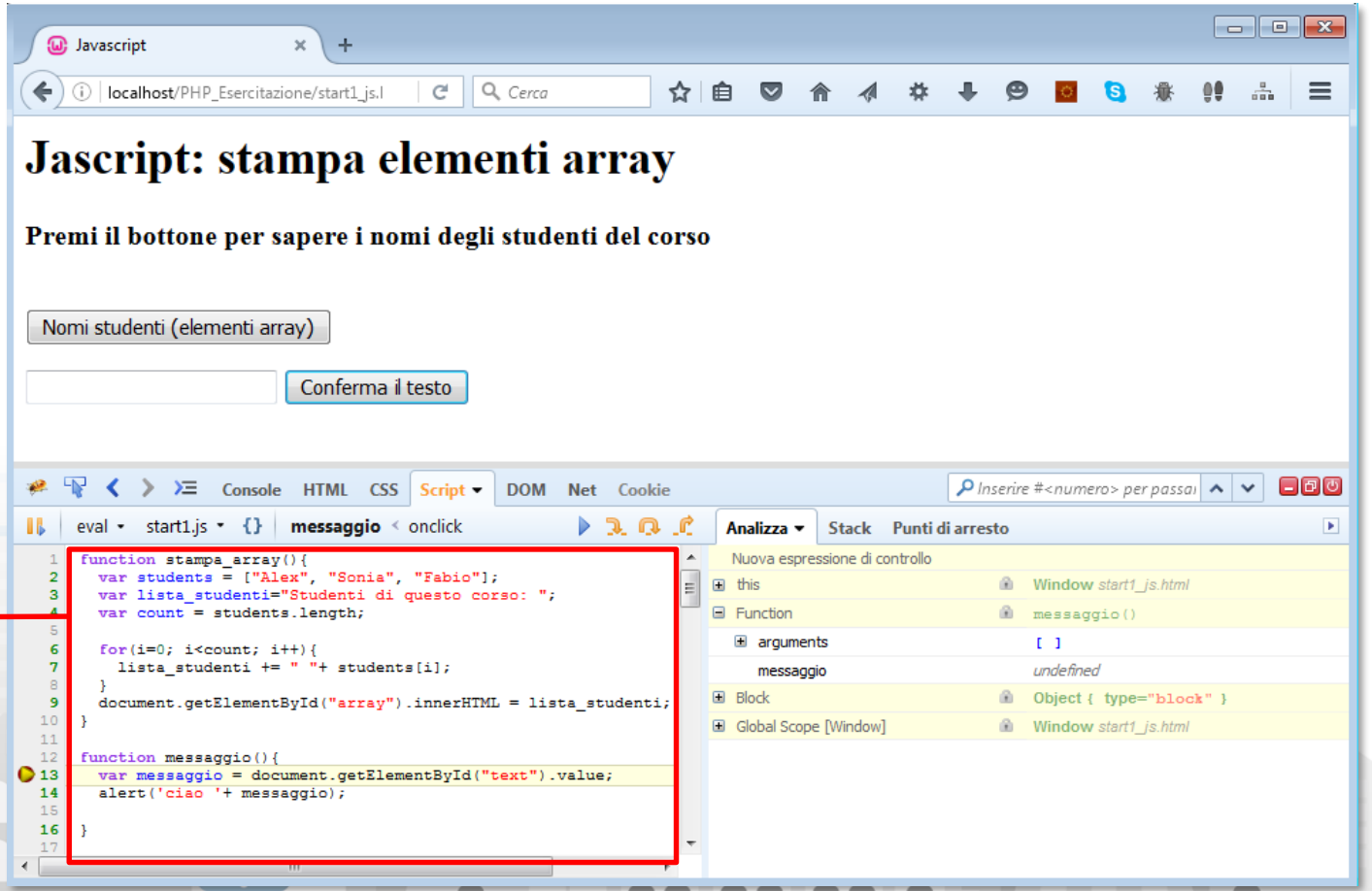


ESEMPIO… usando Crome



ESEMPIO… usando Firefox

- Attivazione del plugin 'Firebug'
- javascript
- Debugger: breakpoint, etc.



Cicli: While

- Sintassi:

```
while(condizione) {
    azione1;
    azione2;
    //azione per far variare la condizione
}
```

NOTE: Il ciclo while dura fino a quando la condizione è vera. Per far questo dobbiamo necessariamente far variare la condizione all'interno del ciclo

- Esempio:

```
var lista_numeri = '';
var numero = 1;
  while(numero <= 10){
       lista_numeri +=  ' '+ numero;
       numero +=1;
  }
```

In questo caso il ciclo while continua fino a quando numero non raggiunge il valore 10

Cicli: do while

- È' simile al ciclo *while* MA mentre il ciclo *while* può non essere eseguito, il ciclo *do while* esegue sempre, almeno per una volta. Questo perché il ciclo *do while* inserisce prima le azioni da fare e dopo la condizione. Il server esegue le prime istruzioni, poi legge la condizione e se è sempre vera esegue nuovamente le istruzioni

- **Sintassi:**

```
do {
 azione1;
 azione2;
 //azione per far variare la
condizione
}
while(condizione)
```

Interrompere un ciclo: break

- **Sintassi:**

```
while{
    azioni;
    if(condizione) break;
}
```

- **Esempio:**

```
var x= 0;
while (x < 10) {
    x++;
    if (x > 5) continue;
        console.log(x);// se x è maggiore di
                       // 5, il log non viene
                       // più eseguito
}
```

Interrompere un ciclo: continue

- **Sintassi:**

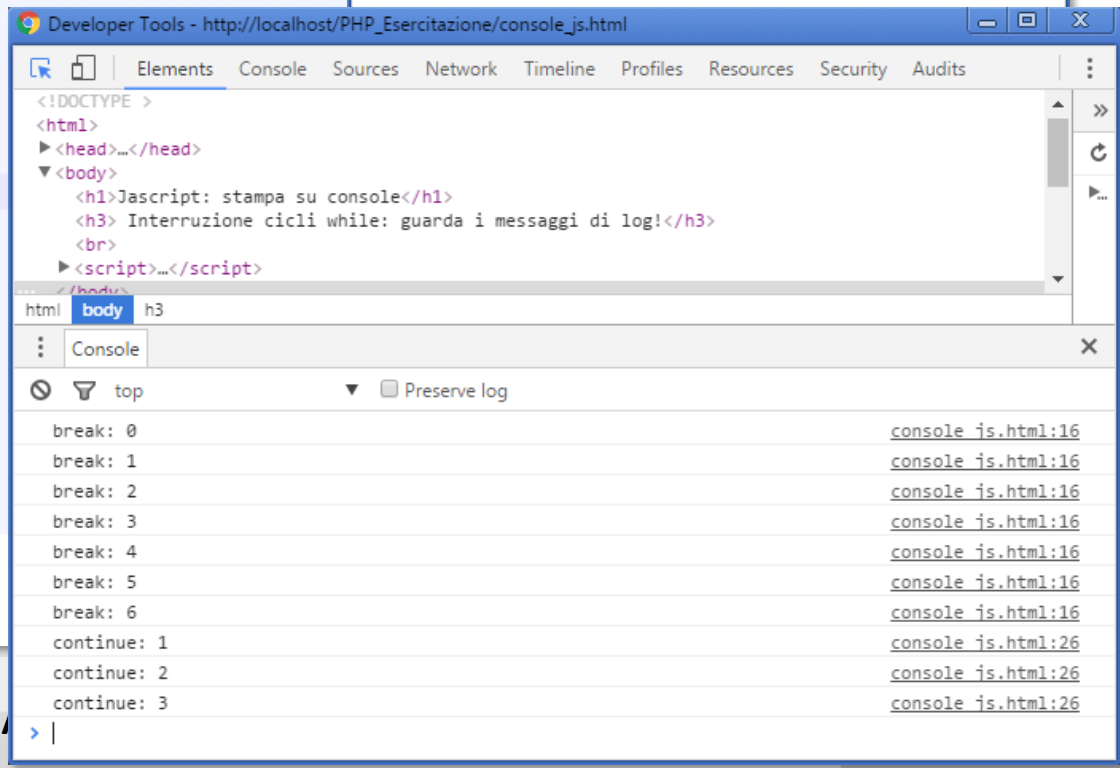
```
while{
    azioni;
    if(condizione) continue;
}
```

- **Esempio:**

```
var x= 0;
while (true) {
  console.log(x);
  // condizione di uscita
  if (x > 20) break;
  x++;
}
```

Esempio: Interruzione di ciclo e scrittura su console (Crome)

```
<!DOCTYPE>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-88
    <title>Javascript</title>
    </head>
    <body>
    <h1>Jascript: stampa su console</h1>
    <h3> Interruzione cicli while: guarda i messaggi di log!</h3>
    <script>
    var x= 0;
    while (true) {
        console.log('break: ' + x);
        // condizione di uscita
        if (x > 5) break;
        x++;
    }
    var x= 0;
    while (x < 10) {
        x++;
        if (x > 3) continue;
            // se x è maggiore di 3,
            //il log non viene più eseguito
            console.log('continue: '+ x);
    }
    </script>
  </body>
</html>
```



Switch

- Si usa se ci sono più alternative e non si vogliono inserire più *if* annidati.

```
switch (a) {
      case <espressione>:
             <lista azioni>
           ...
      default: //entra qui se nessuna condizione è verificata
             <lista azioni>
}
```

- Si usa se si deve gestire una variabile in maniera diversa, in base al valore che assume: con l'istruzione if dovremmo scrivere due if annidati, con switch ne basta uno:

```
switch(stringa){
      case 'ciao':
             alert("Ci vediamo presto");
             break;
      case 'addio':
             alert("Non torni più?");
             break;
      default:
             alert("Forse tornerai");
             break;
}
```

Try catch throw finally (1)

- try {
  <lista azioni>
  }
  catch(err) {
  <lista azioni da eseguire in caso di errore (stampa err)>
  }

  Esempio:

```
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    </head>
    <body >
    <h1>Jascript: gestione errori</h1>
    <h3 > Funziona tutto correttamente? </h3>
    <p id="demo"></p>
    <script>
        try {
            AAAAalert("Welcome guest!"); //errore voluto...
        }
        catch(err) {//scrive nel paragrafo con id='demo'
            document.getElementById("demo").innerHTML = err.message;
        }
    </script>
  </body>
</html>
```

try catch throw finally (2)

- try {

```
<lista azioni>
if(condizione) throw <eccezione>
    /*lista di condizioni che sollevano eccezioni
    personalizzate*/
}
catch(err) {
  <lista azioni da eseguire in caso di errore (stampa err)>
}
finally{
     <lista azioni da eseguire in caso>
    /* indipendentemente dagli errori sollevati */
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE
MABIDA

Esempio:
uso di try catch throw finally

```html
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    <script src="errori_javascript.js"></script>
    <style>#error {color: red;} #finally{color: green;} </style>
    </head>
    <body >
    <h1>Jascript: gestione errori</h1>
    <h3>Inserisci un numero da 1 a 10:</h3>
    <br>
    <input type ="text" id="text" />
    <button type ="submit" onclick="validazione()"> Conferma </button>
    <br>
    <p id="error"></p> <!-- per messagg
    <p id="message"></p> <!-- per messa
    <p id="finally"></p> <!-- sempre --
  </body>
</html>
```

```javascript
function validazione(){
    var x;
    x = document.getElementById("text").value;
    try {
        if(x>=1 && x<=10){//ci passa se va a buon fine
            document.getElementById("message").innerHTML =
                                    "Hai inserito correttamente: " + x;
            document.getElementById("error").innerHTML = "";
            }
        else{
            document.getElementById("message").innerHTML = "";
            if(x == "") throw "campo vuoto, inserisci un numero da 1 a 10!";
            if(isNaN(x)) throw "hai inserito una stringa, non un numero! Riprova!";
            x = Number(x);
            if(x < 1) throw "numero troppo piccolo! Riprova!";
            if(x > 10) throw "numero troppo grande! Riprova!";
        }
    }
    catch(err) {//ci passa se viene sollevato un errore
        document.getElementById("error").innerHTML = "Errore: " + err;
        //alert(err);
    }
    finally{// ci passa sempre
        document.getElementById("finally").innerHTML =
                                "Passo da finally: eseguo in ogni caso... ";
    }
}
```

Esempio: uso di try catch throw finally

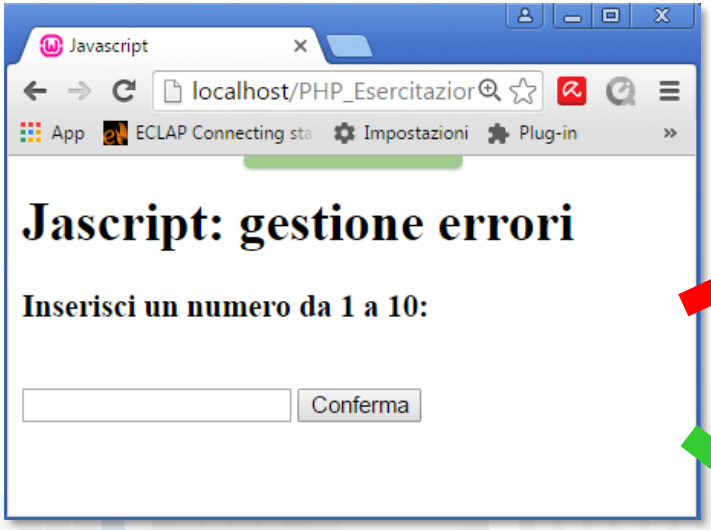
File.html

```
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Javascript</title>
    <script src="errori_javascript.js"></script>
    <style>#error {color: red;} #finally{color: green;} </style>
    </head>
    <body >
    <h1>Jascript: gestione errori</h1>
    <h3>Inserisci un numero da 1 a 10:</h3>
    <br>
    <input type ="text" id="text" />
    <button type ="submit" onclick="validazione()"> Conferma </button>
    <br>
    <p id="error"></p> <!-- per messaggi errore -->
    <p id="message"></p> <!-- per messaggi inserimento corretto -->
    <p id="finally"></p> <!-- sempre -->
  </body>
</html>
```

Esempio: uso di try catch throw finally

File.js

```
function validazione(){
    var x;
    x = document.getElementById("text").value;
    try {
        if(x>=1 && x<=10){//ci passa se va a buon fine
            document.getElementById("message").innerHTML =
                                    "Hai inserito correttamente: " + x;
            document.getElementById("error").innerHTML = "";
            }
        else{
            document.getElementById("message").innerHTML = "";
            if(x == "") throw "campo vuoto, inserisci un numero da 1 a 10!";
            if(isNaN(x)) throw "hai inserito una stringa, non un numero! Riprova!";
            x = Number(x);
            if(x < 1) throw "numero troppo piccolo! Riprova!";
            if(x > 10) throw "numero troppo grande! Riprova!";
        }
    }
    catch(err) {//ci passa se viene sollevato un errore
        document.getElementById("error").innerHTML = "Errore: " + err;
        //alert(err);
    }
    finally{// ci passa sempre
        document.getElementById("finally").innerHTML =
                            "Passo da finally: eseguo in ogni caso... ";
    }
}
```



- Uno dei possibili errori gestiti

- Inserimento andato a buon fine

Approfondimento funzioni… (1)

- parseInt() – Fa il parsing di una stringa e restituisce un intero
  - var a = parseInt("24.00"); //a = 24
  - var b = parseInt("24.35"); //b = 24
  - var c = parseInt("24 34 6"); //c = 24
  - var d = parseInt(" 24 "); //d = 24
  - var e = parseInt("24 ieri"); //e = 24
  - var f = parseInt("ieri era il 24 "); //f = NaN
  - …

Approfondimento funzioni… (2)

- parseFloat() – Fa il parsing di una stringa e restituisce un float
  - var a = parseFloat("24.00"); //a = 24
  - var b = parseFloat("24.35"); //b = 24.35
  - var c = parseFloat("24 34 6"); //c = 24
  - var d = parseFloat(" 24 "); //d = 24
  - var e = parseFloat("24 ieri"); //e = 24
  - var f = parseFloat("ieri era il 24 "); //f = NaN
  - …

Approfondimento funzioni… (3)

- isNaN(val) - Restituisce true/false in base al parametro in ingresso
  - isNaN(0) //false
  - isNaN('stringa') //false
  - isNaN('12/04/2016') //true
  - isNaN('true') //false
  - isNaN('NaN') //true
  - isNaN(undefined) //true
  - isNaN(NaN) //true
- isFinite() – Controlla se un numero è finito o meno
  - var isFinite(24-3) //true
  - var isFinite('stringa') || isFinite('12/04/2016') //false

Approfondimento funzioni… (4)

- escape() – Effettua la codifica delle stringhe (codifica esadecimale di ogni carattere preceduta dal %), di tutti i caratteri tranne: * @ - _ + . /
  - var str = escape("Questa è una stringa!");
  - //str = 'Questa%20%E8%20una%20stringa%21';

- unescape() - Esegue il procedimento contrario rispetto alla escape(). Converte le codifiche esadecimali nei corrispondenti caratteri ASCII

Approfondimento funzioni… (5)

- encodeURI() – Effettua la codifica di un URI. Caratteri NON codificati: , / ? : @ & = + $ #
- decodeURI() – Esegue il procedimento contrario rispetto alla encodeUri()
- encodeURIComponent() - Effettua la codifica di un URI (usato per la codifica dei parametri)
- decodeURIComponent() – Effettua il procedimento inverso rispetto alla encodeUriComponent()

Approfondimento funzioni… (6)

- indexOf(stringa) – Rende l'indice della stringa in ingresso
  - var stringa = 'scrivo una stringa';
  - var indice = indexOf('stringa') //index = 11;

- replace() – Sostituisce una stringa con un'altra
  - var stringa = 'scrivo una stringa';
  - var res = str.replace("stringa", "riga di codice");
  - // res= 'scrivo una riga di codice';

Approfondimento funzioni… (7)

- substr(c1,c2) – Estrae una sottostringa da una stringa dal carattere c1 (compreso) al carattere c2 (compreso)
  - var str = "Buongiorno!";
  - var res = str.substr(1, 4); //res = ‘uong’;
- substring() – Estrae una sottostringa da una stringa dal carattere c1 (compreso) al carattere c2 (NON compreso)
  - var str = "Buongiorno!";
  - var res = str.substr(1, 4); //res = ‘uon’;

Approfondimento funzioni… (8)

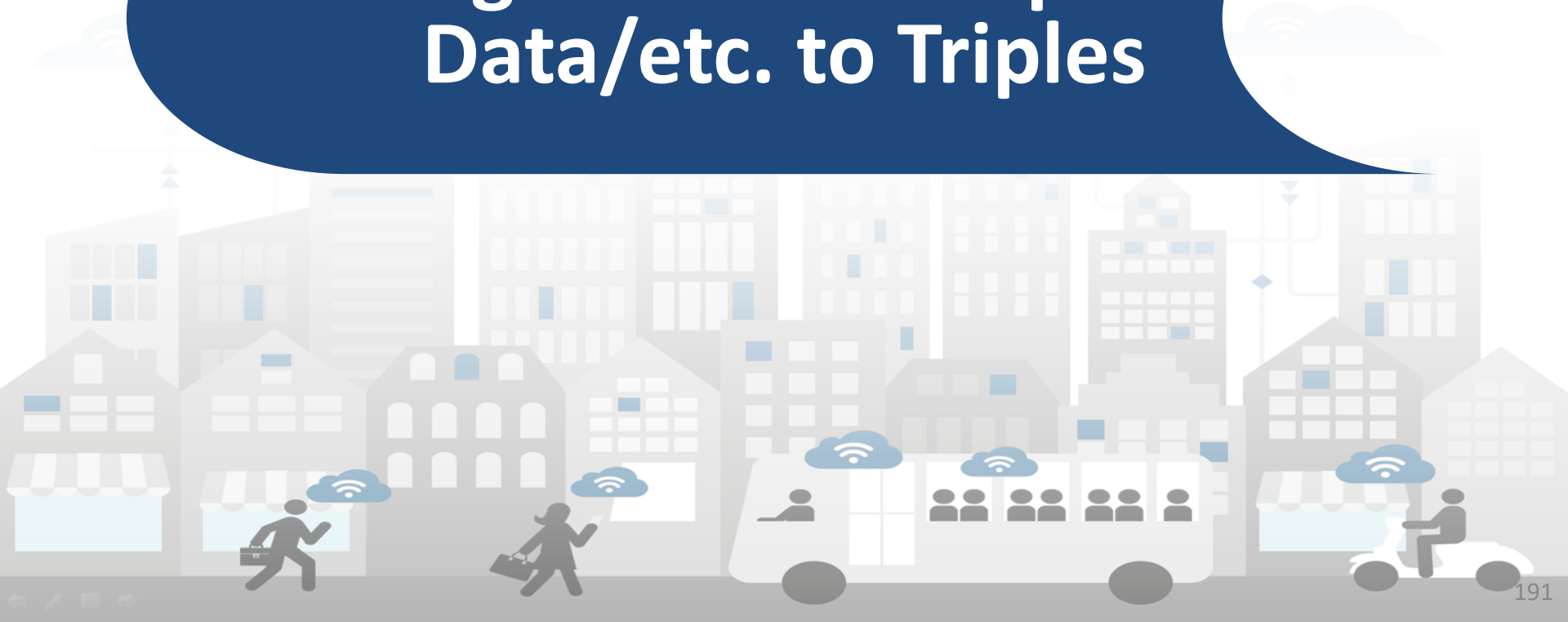
- toLowerCase(str) – Converte la stringa str in caratteri minuscoli
- toUppercase() - Converte la stringa str in caratteri maiuscoli
- trim() – Toglie gli spazi vuoti all'inizio e alla fine di una stringa
  - var email = trim(‘ nome@google.it ’);
  - //email = ‘nome@google.it’
- startsWith(str) / endsWith – restituisce true o false se una stringa inizia/finisce o meno con str
  - var stringa = ‘Benvenuto sul nostro portale!’;
  - var inizia = stringa.startWith(‘Benvenuto’); //init = true;

Serializzare gli oggetti in JavaScript

- Si parla di:
  - Serializzazione: processo di trasformazione di un oggetto/dato/informazione/… in un formato facilmente memorizzabile e/o trasmissibile
  - Deserializzazione: processo inverso

- La serializzazione in javascript avviene attraverso la la rappresentazione JSON, Javascript Object Notation

Big Data: from Open Data/etc. to Triples



Index

- Big Data: from Open Data to Triples
- ETL processes
- ETL tool: Pentaho Data Integration (PDI)
  - Features
  - Key concepts
  - Examples
- Developing ETL processes with PDI
  - Tool installation & configuration
  - SiiMobility & Snap4city Project
  - ETL processes implementation

Classic Data warehouse Architecture

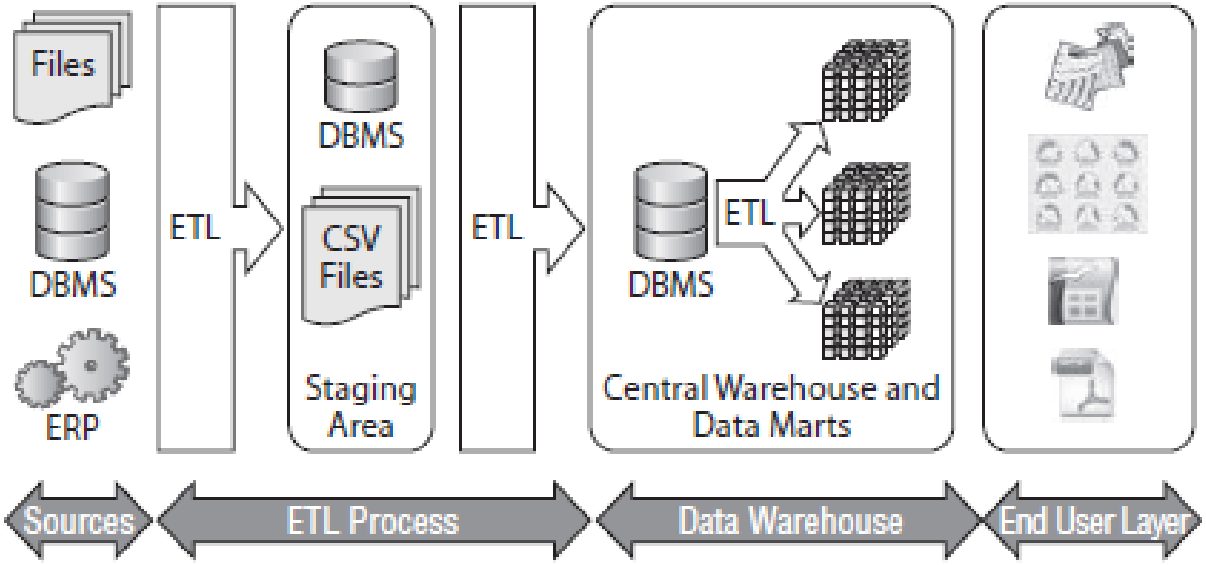
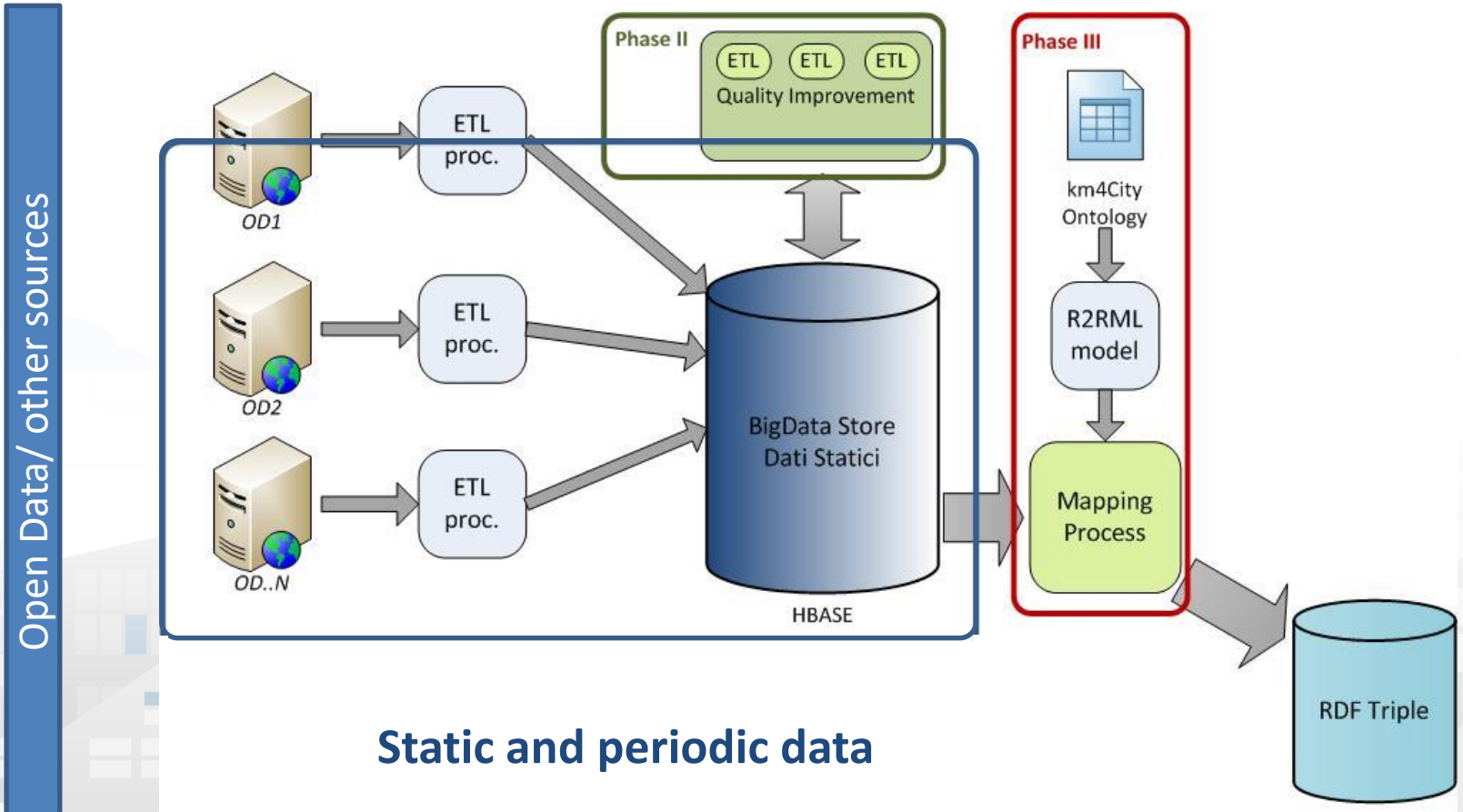


Figure 1-1: Classic data warehouse architecture

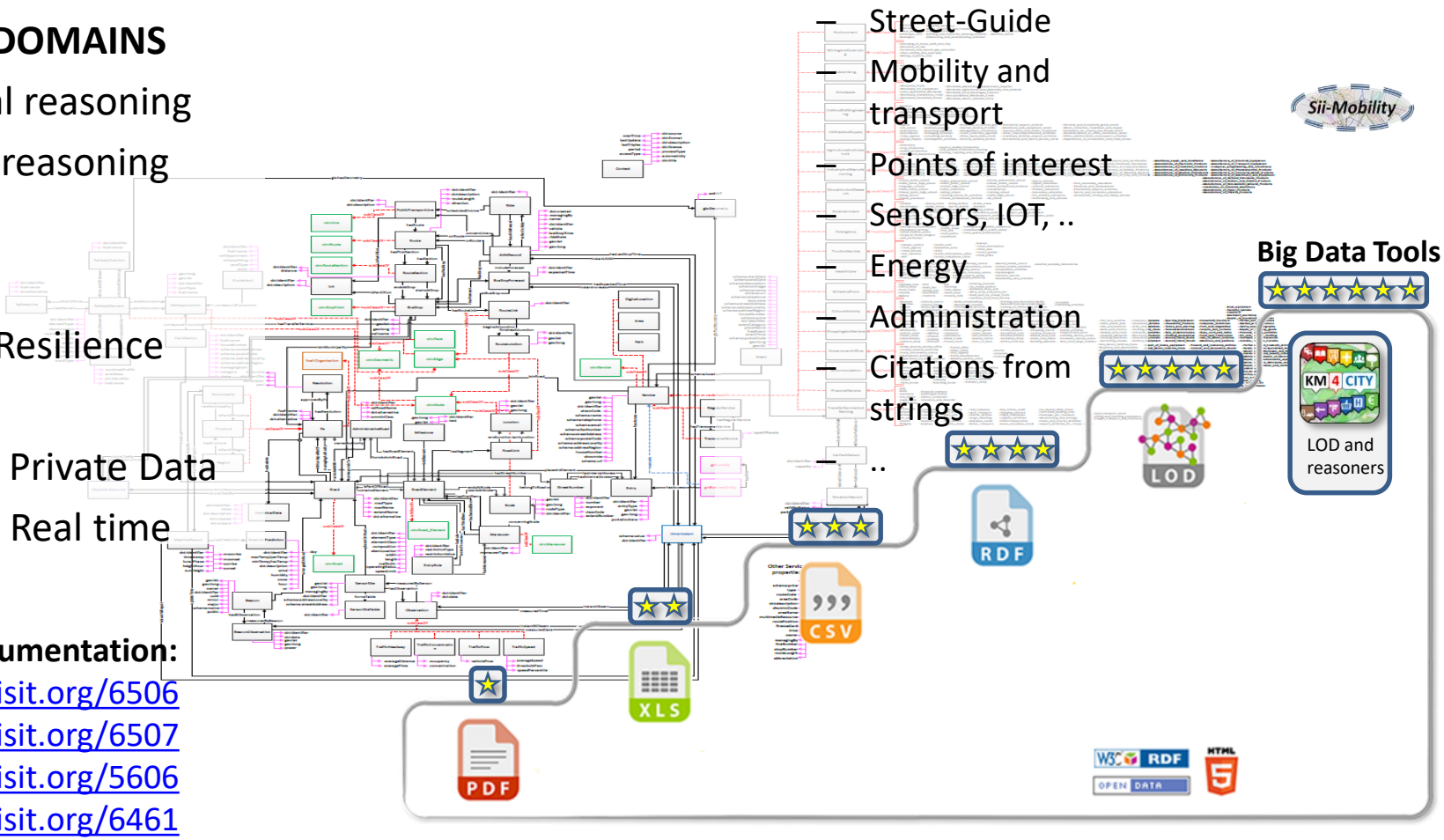
Data Engineering Architecture



Static and periodic data

Km4City: Knowledge Base

- **Multiple DOMAINS**
- Geospatial reasoning
- Temporal reasoning
- Metadata
- Statistics
- Risk and Resilience
- Licensing
- Open and Private Data
- Static and Real time



Ontology Documentation:

http://www.disit.org/6506
http://www.disit.org/6507
http://www.disit.org/5606
http://www.disit.org/6461

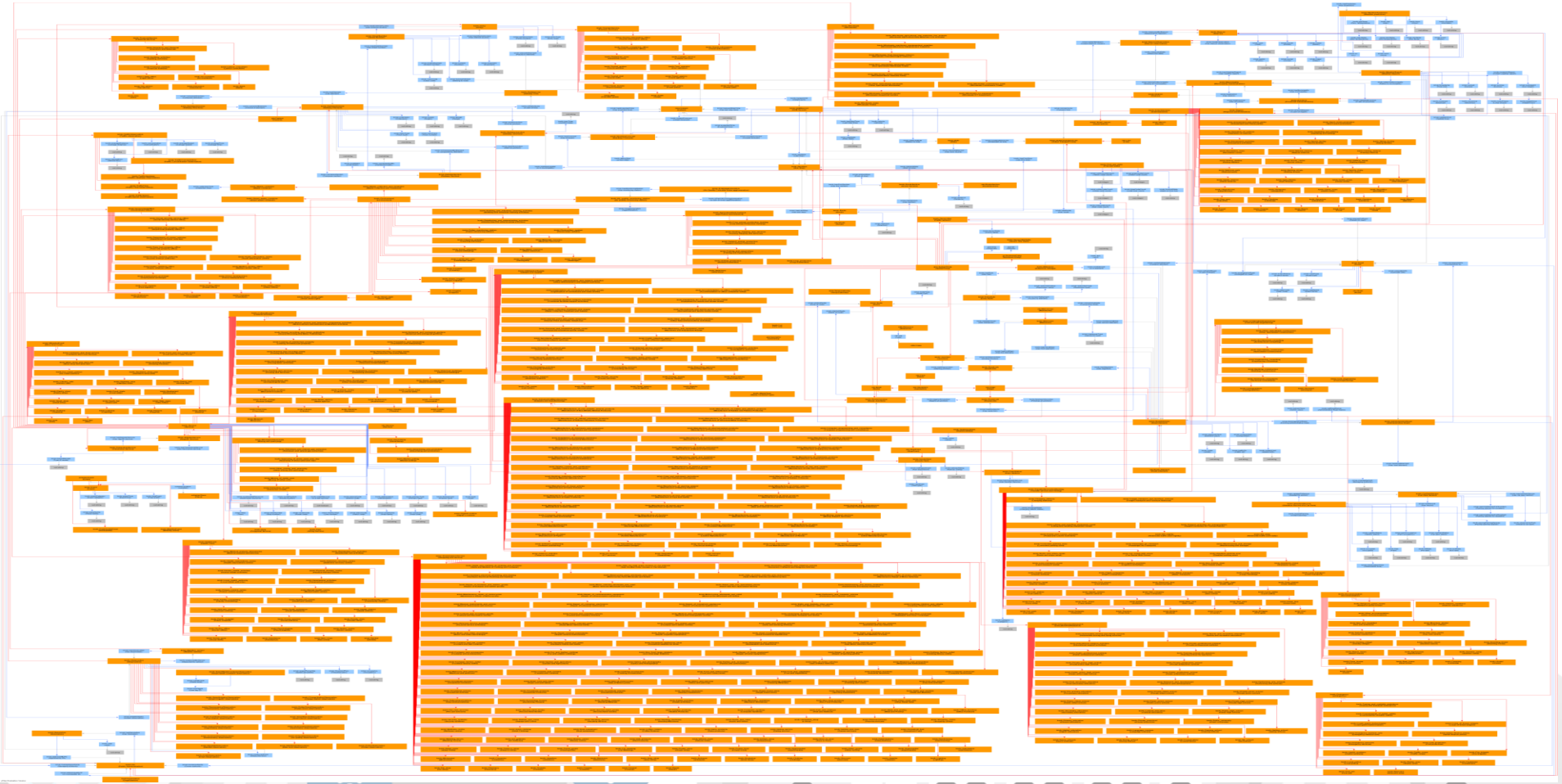
UNIVERSITÀ DEGLI STUDI FIRENZE

DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
http://www.disit.org

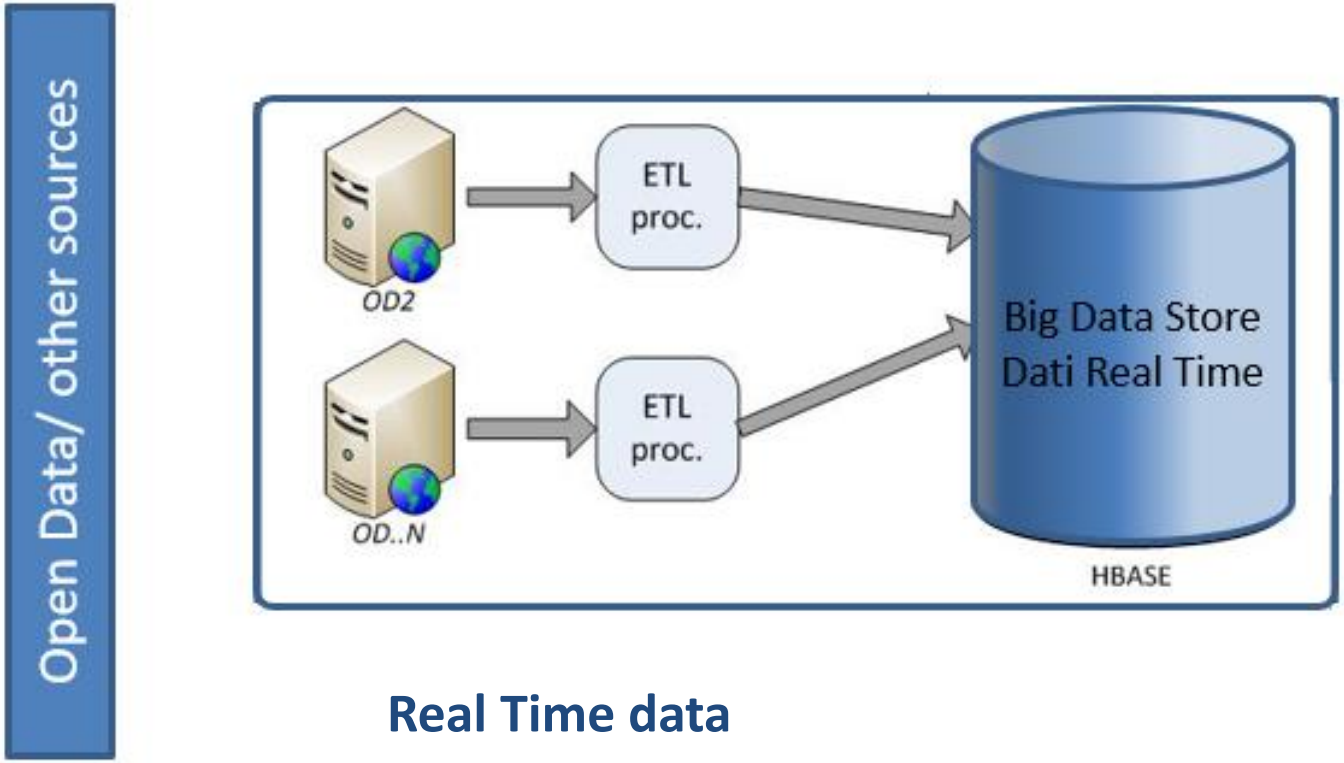
UNIVERSITÀ DEGLI STUDI FIRENZE
MABIDA

Smart-city Ontology km4city



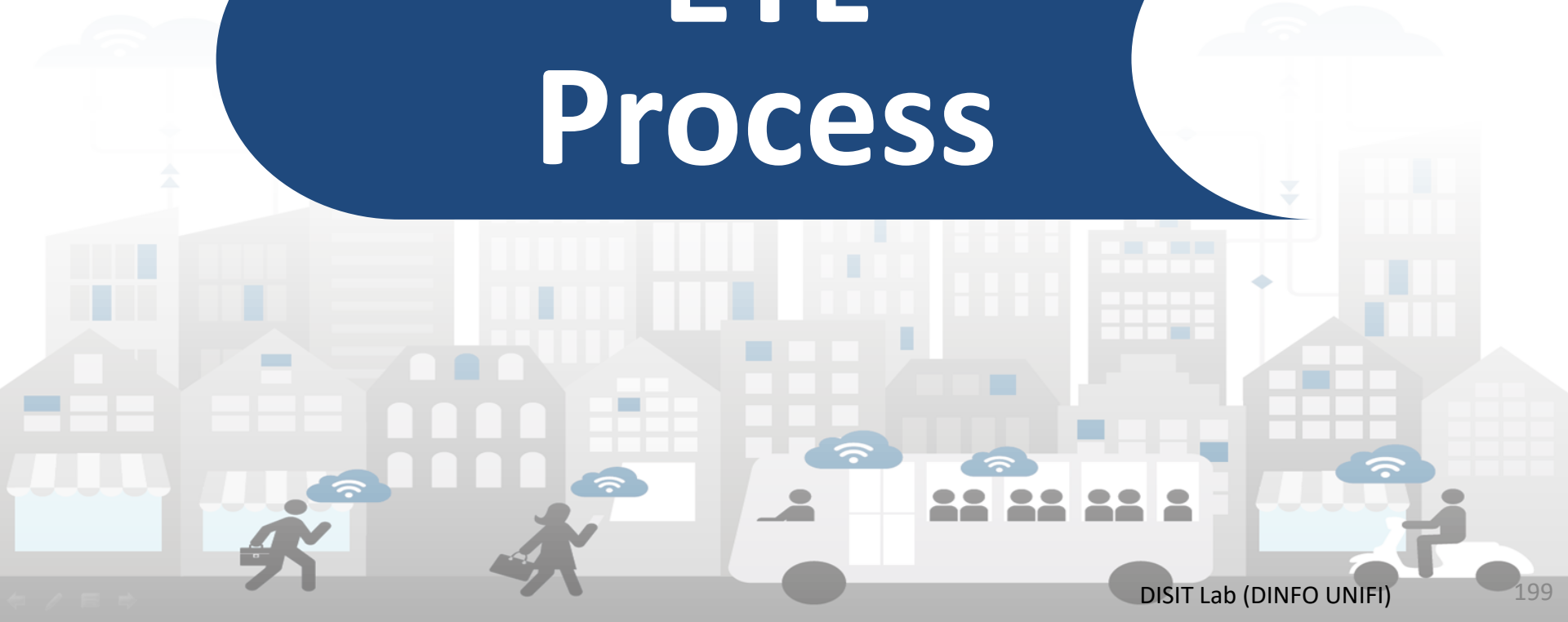
Big Data: from Open Data/etc. to NoSQL Databases

Data Engineering Architecture



Real Time data

ETL Process



Useful tools

Pre-processing data to RDF triples generation: ETL (Extract, Transform and Load)

– Process used in database and data warehousing that involves three phases.

– Useful tools to prepare data for the following data analysis phase and eventual translation into RDF.

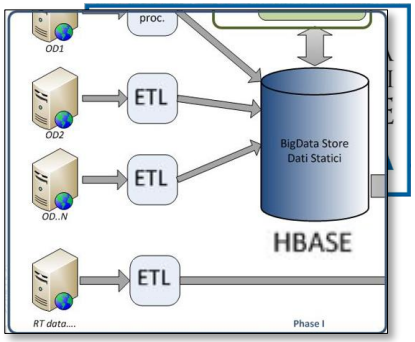
– Translation in **RDF Triples** is based on use of a specific **ontology**

- Definition of mapping models from SQL to RDF
- The triples generated are loaded on RDF store

ETL Processes

The three phases are:

- **Extracting** data from outside sources (**Ingestion** phase).
- **Transforming** data to fit operational needs which may include improvements of quality levels (**Data Quality Improvement** phase).
- **Loading** data into the end target (database, operational data store, data warehouse, data mart, etc.). So the data can be translated in **RDF triples using a specific ontology (Static/periodic datasets) or on NoSQL Databases (Dynamic datasets)**



Phase I: Data Ingestion

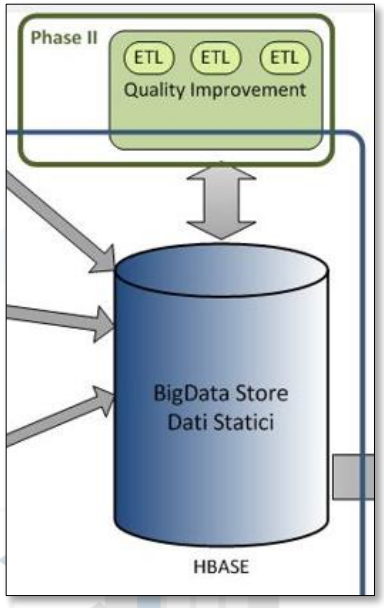
- **Purpose is to store data in HBase**

(Big Data NoSQL database).

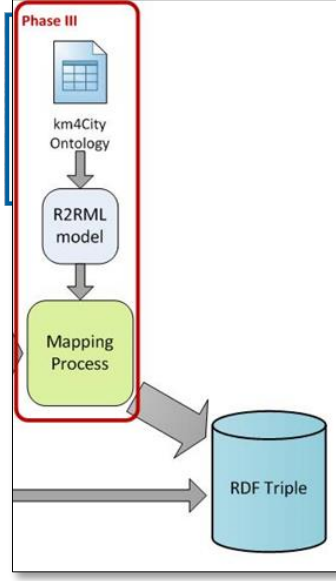
- **Acquisition of wide range of OD/PD**: open and private data, static, quasi static and/or dynamic real time data.
- **Static and semi-static data** include: points of interests, geo-referenced services, maps, accidents statistics, etc.
  - files in several formats (SHP, KML, CVS, ZIP, XML, JSON, etc.)
- **Dynamic data** mainly data coming from sensors
  - parking, weather conditions, pollution measures, bus position, …
  - using Web Services
- Using **Pentaho - Kettle** for data integration (Open Source tool)
  - using specific **ETL** Kettle transformation processes (one or more for each data source)

Phase II: Data Quality Improvement

- **Purpose: add more information as possible and normalize data from ingestion**
- **Problems kinds:**
  - Inconsistencies, incompleteness, typos, lack of standards, multiple standards, …
- **Problems on:**
  - Place-name code
  - Street names (e.g., dividing names from numbers, normalize when possible)
  - Dates and Time: normalizing
  - Telephone numbers: normalizing
  - Web links and emails: normalizing



Phase III: Data mapping



- **Purpose is to translate data from QI in RDF triples**
- **We use triples to do inference on data.**

- Using **Karma Data Integration tool, a mapping model** from SQL to RDF on the basis of the ontology was created.
  - Data to be mapped first temporary passed from HBase to MySQL and then mapped using Karma (in batch mode)

- The mapped data in triples have to be uploaded (and indexed) to the **RDF Store** (Virtuoso).
- Triples are composed by a subject, a predicate and an object.

ETL tool: Pentaho Data Integration (PDI)

Pentaho Data Integration (PDI)

- **Pentaho** is a framework that contains several packages integrated to allow complete management:
  - ***Business Intelligence problems;***
  - ***Data Warehouse problems;***
  - ***Big Data problems.***

- **Kettle** is the ETL component Pentaho for data transfer and processing.

Pentaho Data Integration (Kettle) (1)

- Free, **open** ETL tool
  - It is available also in **enterprise version**
- **Developed in Java**, therefore is guaranteed the compatibility and portability with the major operating systems (Windows, Linux, OS X, …)
- **Powerful** Extraction, Transformation and Loading (ETL) capabilities

Pentaho Data Integration (Kettle) (2)

- **Scalable**, standards-based architecture.
- Opportunity to interfacing with the main NoSQL Databases (HBase, Cassandra, MongoDB, CouchDB, …).
- It uses an innovative, **metadata-driven** approach.
- Graphical, **drag and drop** design environment.

Pentaho Data Integration (Kettle) (3)

Main strengths:

- Collect data from a **variety of sources** (extraction);
- Move and modify data (transport and transform) while cleansing, denormalizing, aggregating and enriching it in the process;
- Frequently (daily) store data (loading) in the final target destination, usually a **large** dimensionally modeled **database** (or **data warehouse**).

Pentaho Data Integration (Kettle)(4)

Main weakness:

- Kettle is not able to transform data into RDF triples, therefore it is necessary use other tools at a later stage (**Karma**).

Kettle's 4 main programs

- **Spoon:** graphically oriented end-user tool to model the **flow of data** from input through transformation to output (**transformation**)
- **Pan** is a **command line tool** that executes transformations modeled with Spoon
- **Chef**: a graphically oriented **end-user tool** used to model **jobs** (transformations, FTP downloads etc. placed in a flow of control)
- **Kitchen** is a **command line tool** to execute jobs created with Chef.

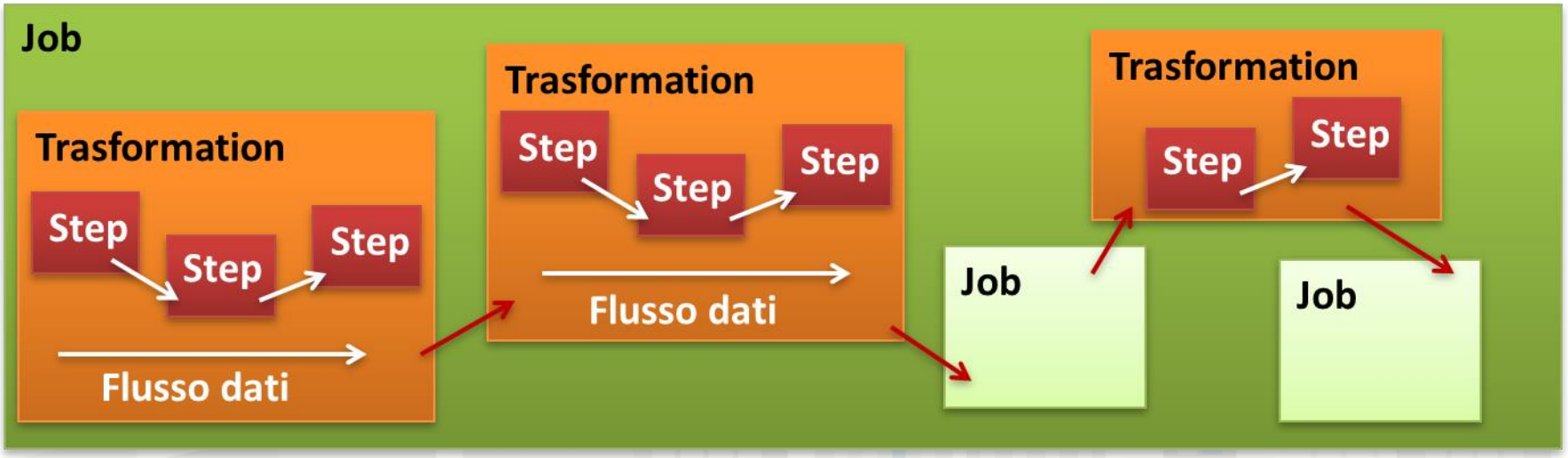
Kettle: Concepts

- Kettle is based on two key concepts (from operating point of view):
  - **Job** (with extension ".kjb");
  - **Transformation** (with extension ".ktr"), composed of several **steps**.

- Kettle's key components are:
  - **Spoon** for ETL process modeling;
  - **Pan** to execute the transformations from command line;
  - **Kitchen** to execute the Job from command line.

Kettle: Operational structure

Kettle operating components are organized as follows:



- The steps are executed in parallel on separated threads and there is no necessarily a beginning or end point of the transformation.
- A job manages the sequential execution of lower-level entities: transformations or other jobs.

Spoon Concepts: Steps and hoops

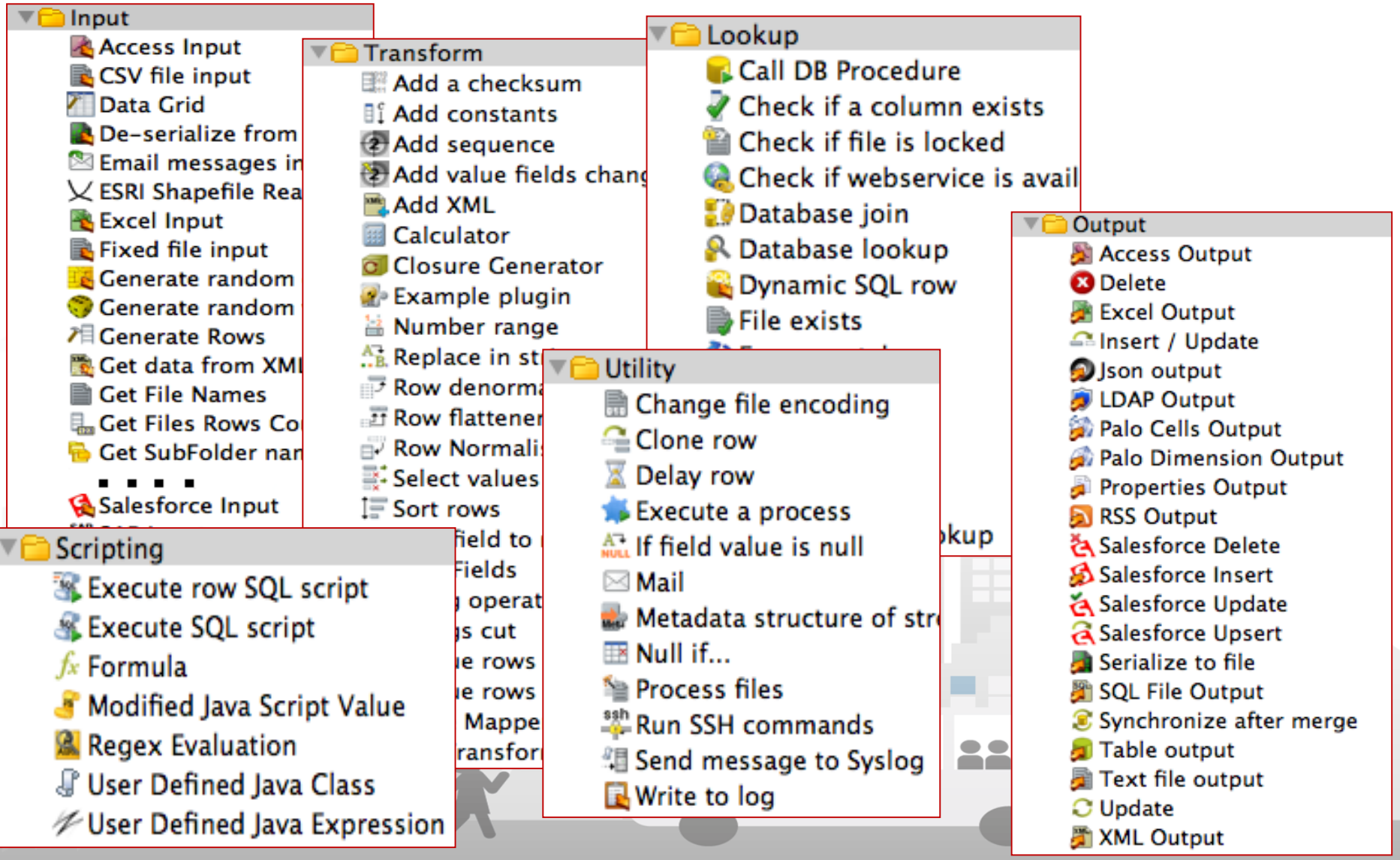
- One **step** denotes a particular kind of **action** that is performed **on data**.
- **Hops** are links to connect steps together and allow data to pass from one step to another.
- Steps can be easily "created" by **dragging** the icon from the treeview **and dropping** them on the graphical model view.
- Kettle provides a lot of different step types and can be **extended with plugin**.

Type of Steps in Spoon

Three different kinds of steps:

- **Input:** process some kind of 'raw' resource (file, database query or system variables) and create an output stream of records from it.
- **Output:** (the reverse of input steps): accept records, and store them in some external resource (file, database table, etc.).
- **Transforming:** process input streams and perform particular actions on them (adding new fields/new records); these actions produce one or more output streams.
- **NOTE: 'Main.kjb'** is usually the primary job.
- REF: http://wiki.pentaho.com/display/EAI/Pentaho+Data+Integration+Steps

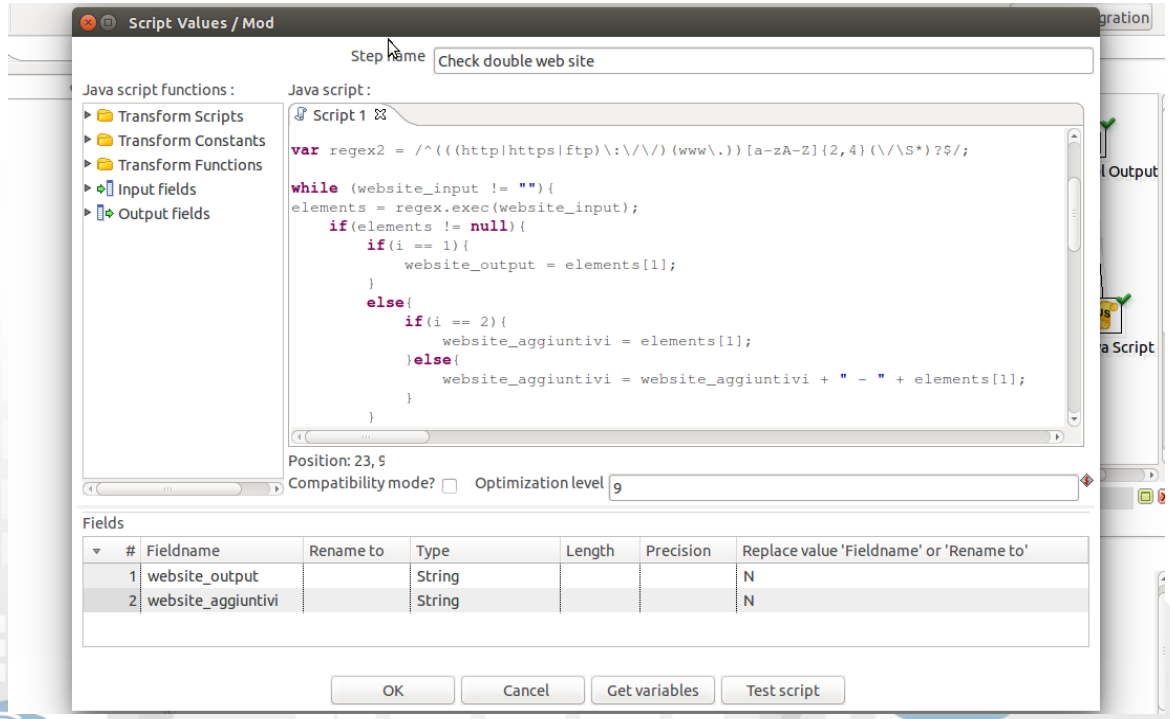
Type of Transformations in Spoon



Kettle: Transformations

Kettle offers many types of steps in order to execute various data operations, also it offers:

- Possibility of add some JavaScript code
- Possibility of use regular expressions.

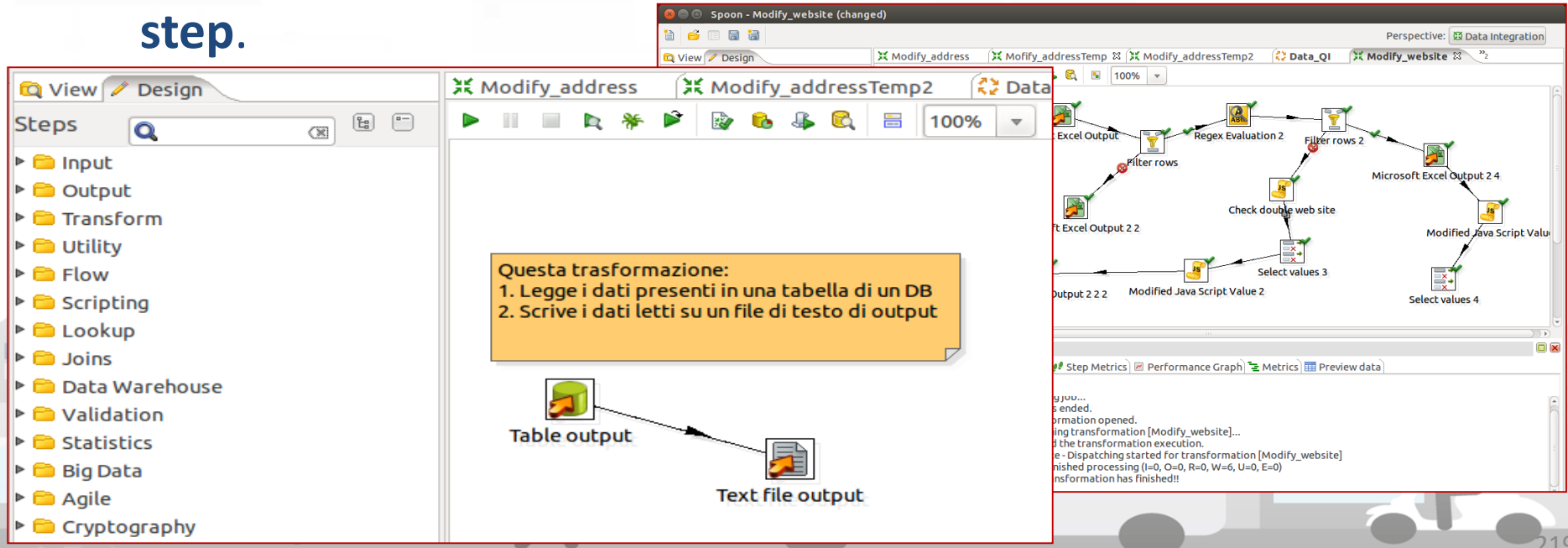


Spoon Concepts: Steps and hoops

- One **step** denotes a particular kind of **action** that is performed **on data**.
- **Hops** are links to connect steps together and allow data to pass from one step to another.
- Steps can be easily "created" by **dragging** the icon from the treeview **and dropping** them on the graphical model view.
- Kettle provides a lot of different step types and can be **extended with plugin**.

Kettle: Transformations

- Transformations define how the data must be collected, processed and reloaded.
- Consist of a series of steps connected by links called Hop.
- Typically a transformation has one **input step**, one or multiple **transformation steps** and one or more **output step**.



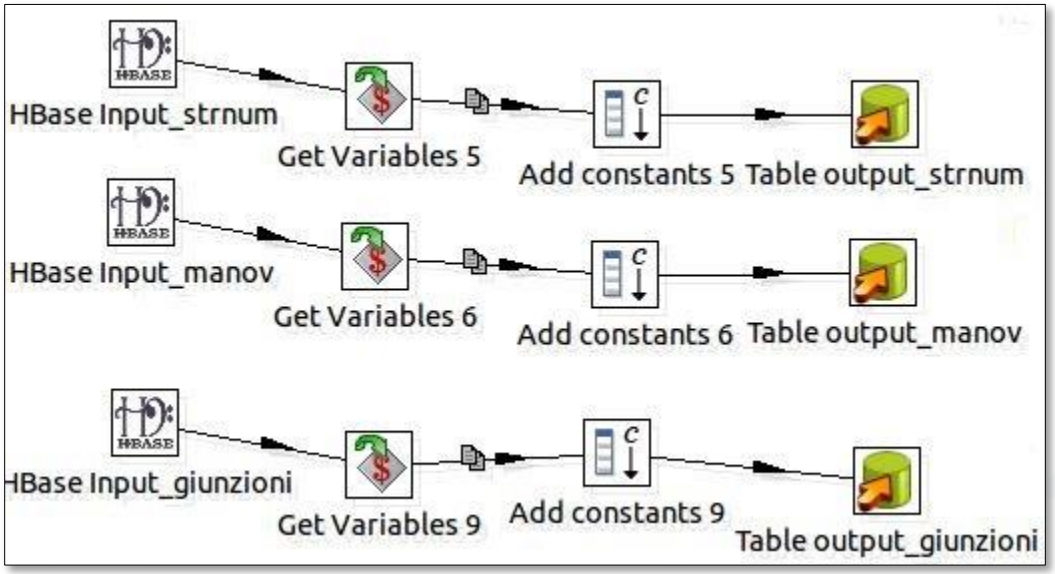
Sequential Execution

These steps (transformations) are executed sequentially (there is a single flow execution).

```
void main(){
        int a;
        f1(a);
        f2(a+2);
}
```

The statements are executed sequentially.

Parallel Execution



- Unlike before there are multiple streams of execution that are executed in parallel (simultaneously).
- Like in a multi-threading programming, multiple thread (portions of the running program) can virtually run independently and in parallel.

Kettle: Pan e Kitchen

- The Transformations made with Spoon can be executed with Pan from command line (similarly Kitchen for the Job).

```
/usr/local/pdi/pan.sh -file /home/pentaho/repos/LetturaDati.ktr
```

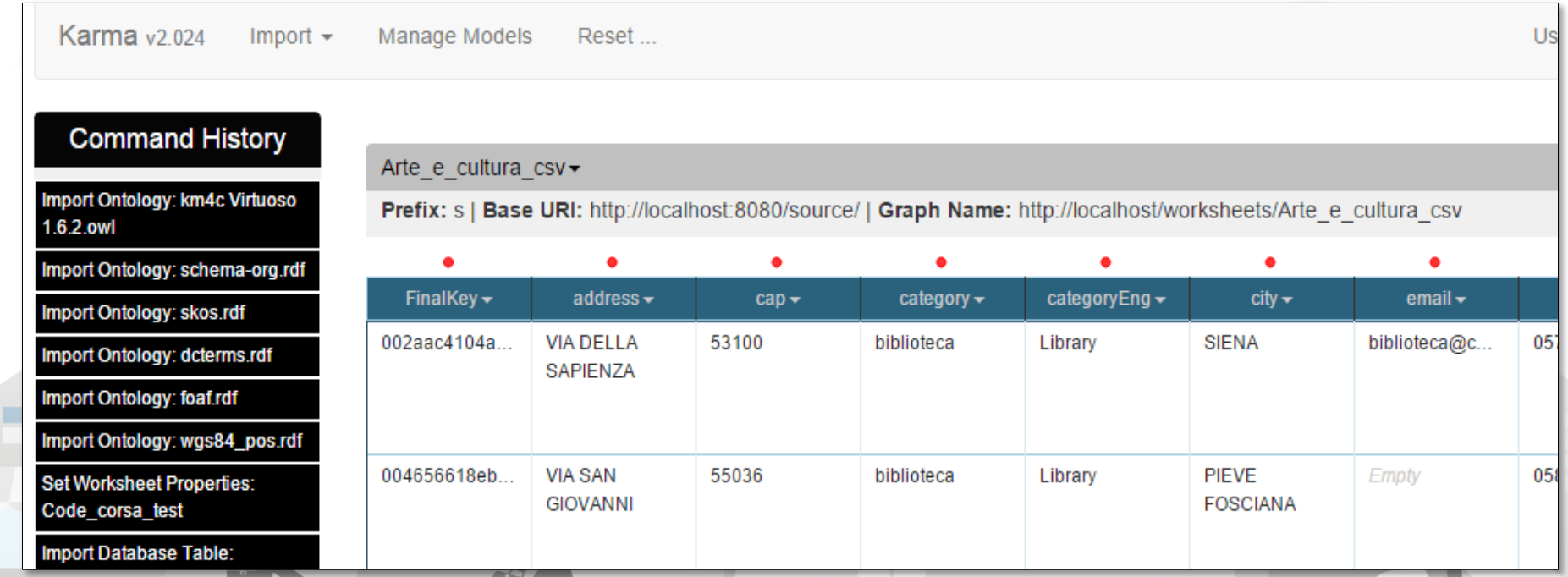
```
# Lancia il job ogni sabato alle sei di mattina...
6 6 * * 6 /usr/local/pdi/kitchen.sh -file /home/pentaho/repo/Aggiorna1.kjb >> /tmp/cron1.log 2>&1
```

- The output is typically recorded on a log which can be analyzed in case of errors.

```
INFO  07-06 06:10:02,486 - Using "/tmp/vfs_cache" as temporary files store.
INFO  07-06 06:10:02,712 - Pan - Start of run.
INFO  07-06 06:10:02,902 - Lettura dati per DWH - Dispatching started for transformation [Lettura dati per DWH]
INFO  07-06 06:10:02,929 - Lettura dati per DWH - This transformation can be replayed with replay date: 2011/06/07 06:10:02
INFO  07-06 06:10:03,233 - DB DWH - Connected to database [Self DB] (commit=100)
INFO  07-06 06:10:03,599 - DB AS_UTIL - Finished reading query, closing connection.
INFO  07-06 06:10:03,614 - DB AS_UTIL - Finished processing (I=27, O=0, R=0, W=27, U=0, E=0)
INFO  07-06 06:10:03,625 - DB DWH - Finished processing (I=0, O=27, R=27, W=27, U=0, E=0)
INFO  07-06 06:10:03,626 - Pan - Finished!
INFO  07-06 06:10:03,627 - Pan - Start=2011/06/07 06:10:02.713, Stop=2011/06/07 06:10:03.126
INFO  07-06 06:10:03,627 - Pan - Processing ended after 0 seconds.
INFO  07-06 06:10:03,627 - Lettura dati per DWH -
INFO  07-06 06:10:03,627 - Lettura dati per DWH - Step DB AS_UTIL.0 ended successfully, processed 27 lines. ( - lines/s)
INFO  07-06 06:10:03,628 - Lettura dati per DWH - Step DB DWH.0 ended successfully, processed 27 lines. ( - lines/s)
```

Karma

- Karma is a mapping model based on ontology (**km4city**) from MySQL tables to RDF.
- Triples are uploaded to **Virtuoso**, an RDF Store.
- It can import MySQL tables but no HBase ones.



ETL Examples

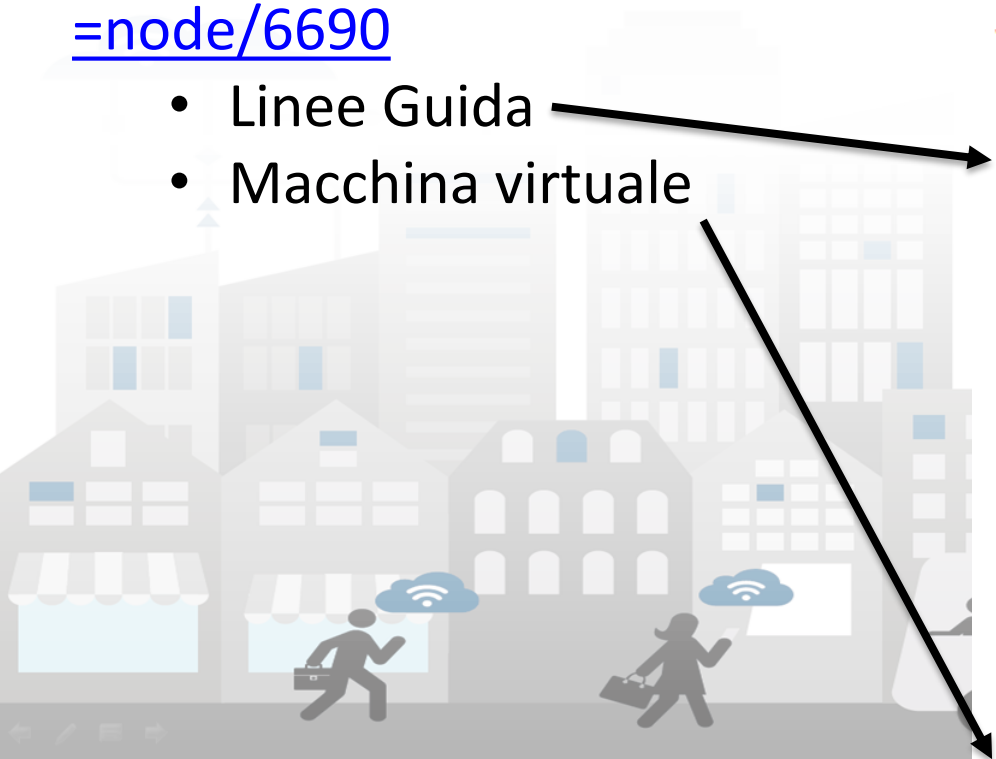
UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB http://www.disit.org

UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

GUIDA e strumenti: Programmazione ETL per Data Warehouse

http://www.disit.org/drupal/?q=node/6690

- Linee Guida
- Macchina virtuale



Distributed Systems and Internet Technologies Lab
Distributed Data Intelligence and Technologies Lab
Department of Information Engineering (DINFO)
University of Florence

http://www.disit.dinfo.unifi.it

HOME | ABOUT | RESEARCH | INNOVATION | EDUCATION AND COURSES | HOWTO | EVENTS

GUIDA E STRUMENTI: PROGRAMMAZIONE ETL PER DATA WAREHOUSE

DOCUMENTAZIONE

- Sii-Mobility: DE4.2a-Sistema di acquisizione e Aggregazione dati, dal concetto al dato, dal dato al database con ETL, e dal database al modello ontologico (ITA, ENG)
- GUIDA alla programmazione: Programmazione ETL per Data Warehouse (ITA)
- manuale utente per la creazione di ETL per dati statici e dinamici
- SLIDE: Km4City Sii-Mobility: Data Ingestion Tutorial, Overview, Parte 1
- VIDEO: Km4City Sii-Mobility: Data Ingestion Tutorial, Overview, Parte 1
- SLIDE esercitazioni produzione ETL: Km4City Sii-Mobility: Data Ingestion Tutorial, Parte 2: Teoria ed esercitazioni, vedi anche video
- VIDEO Parte 2a, teoria: Data Ingestion Tutorial
- VIDEO Parte 2b, Esercitazione su ETL, data ingestion Tutorial
- Slide 2014-2015 Programmazione ETL per DataWarehouse (Parte 8): from open data to triples, OD 2 RDF, OD and PD, static and Dynamic OD, Problemi architetturali, programmazione ETL, esempi concreti, massive data mining and crawling, quality improvement, geolocalization, triplification, reasoning and rendering, example of km4city data ingestion.
- esempi di processi formalizzati in ETL per il DataWarehouse
- **Testi consigliati**
  - Pentaho Data Integration 4 Cookbook - PACKT Publishing (A. S. Pulvirenti, M. C. Roldàn)
  - Pentaho Kettle Solutions - Wiley (M. Casters, R. Bouman, J. van Dongen)
- pagina web Km4City: http://www.km4city.org
- pagina Open Source di DISIT Org: http://www.disit.org/6763
- Scarica il Flyer di Km4City: http://www.km4city.org/km4city-booklet-v02-21x21-md1.pdf
- P. Bellini, M. Benigni, R. Billero, P. Nesi and N. Rauch, "Km4City Ontology Building vs Data Harvesting and Cleaning for Smart-city Services", International Journal of Visual Language and Computing, Elsevier, http://dx.doi.org/10.1016/j.jvlc.2014.10.023 http://www.disit.org/6573

MACCHINA VIRTUALE, VMSDETL, GIA' PRONTA

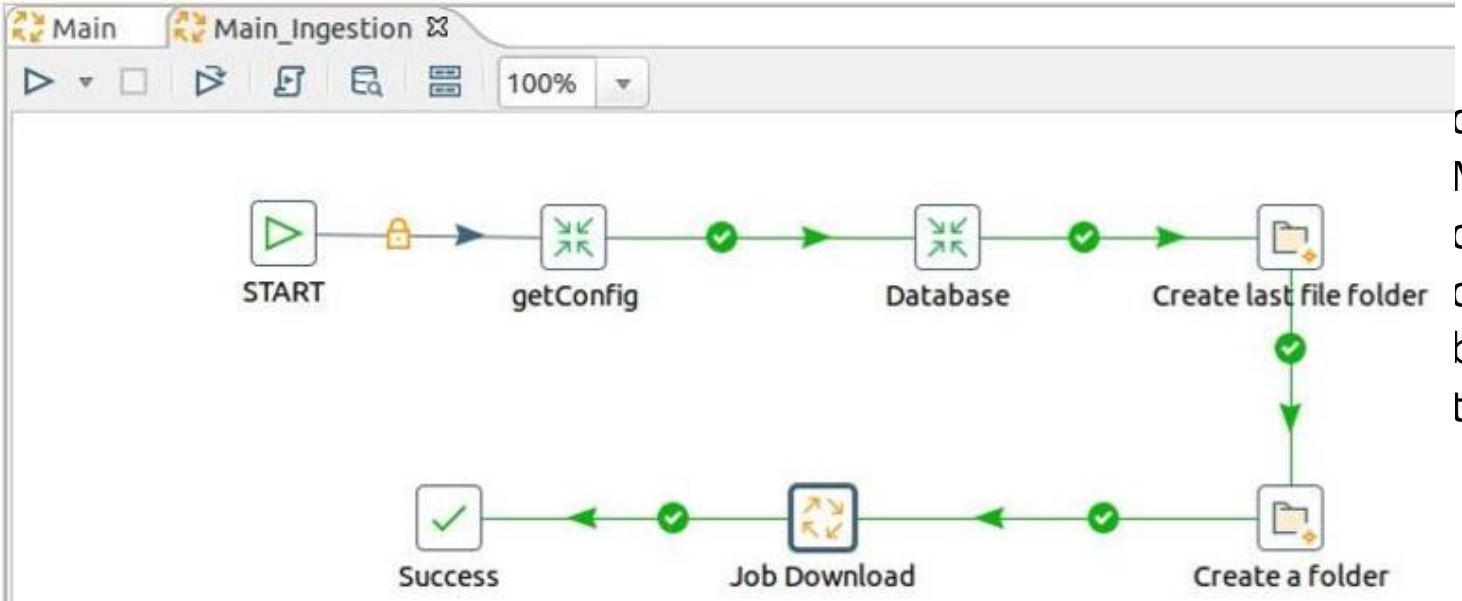
VMSDETL, con Linux Ubuntu 14.04 (root: ubuntu, password: ubuntu)

- **questo è il LINK alla macchina virtuale (versione 0.7, 28-02-2017), da scaricare e decomprimere in una directory, include Karma**
  - Versione del 2017/2018 0.8 con Phoenix
  - Versione del 2017/2018 0.8 con Phoenix per Virtualbox

Ingestion

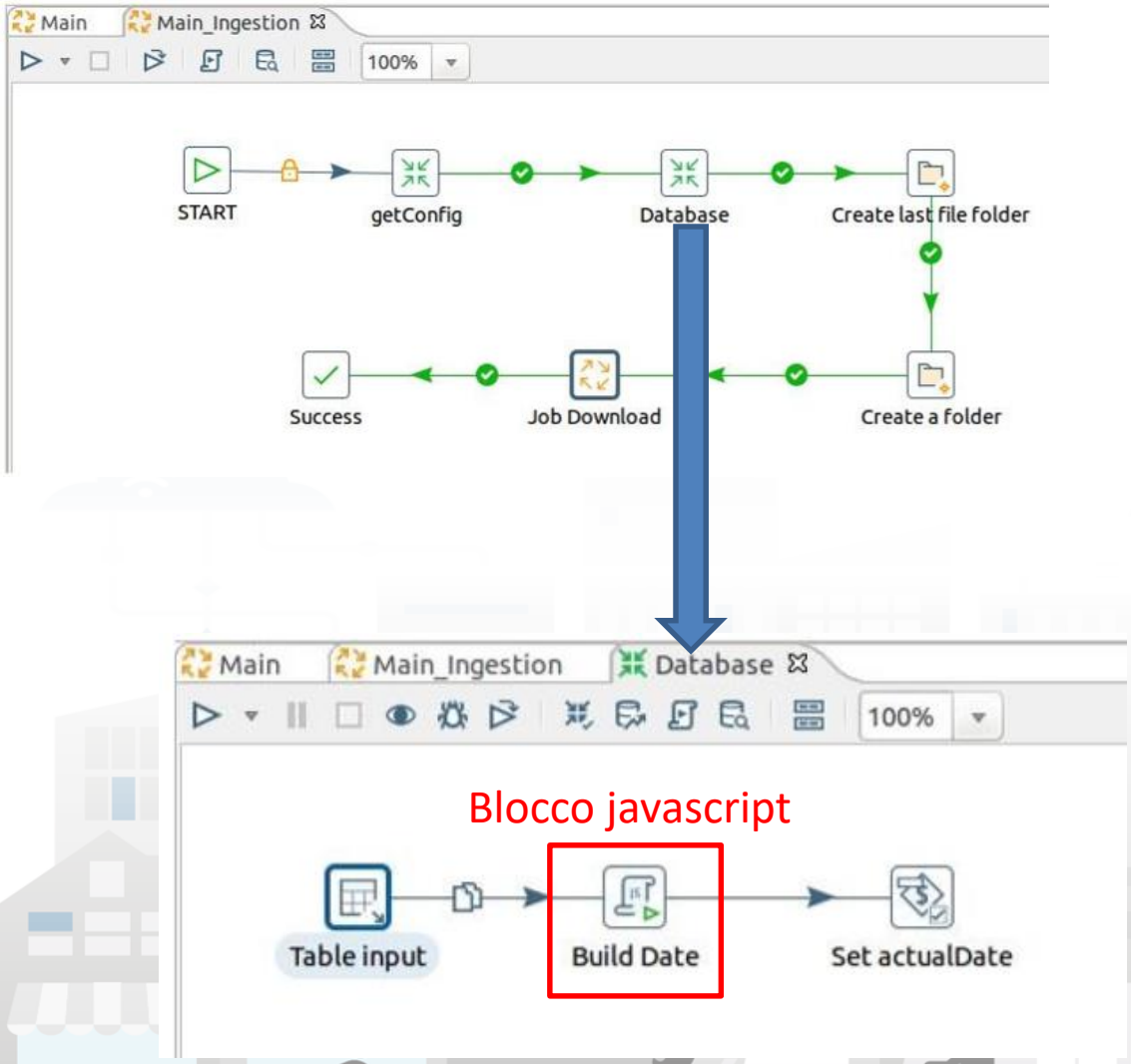
In questa trasformazione si procede ne modo seguente:

i) acquisizione del dataset dal provider (ad esempio scaricando un file dal web, tramite ftp, facendo crawling, etc.) e copia di esso in in file che viene memorizzato nel file system;
ii) lettura dei dati dal file appena memorizzato;
iii) estrazione dei dati di rilievo;
iv) memorizzazione dei dati estratti su una tabella HBase.



In Figura, la pipeline del job Main_Ingestion.kjb, che a sua volta contiene altri step (o blocchi), job, trasformazioni.

Ingestion (2)

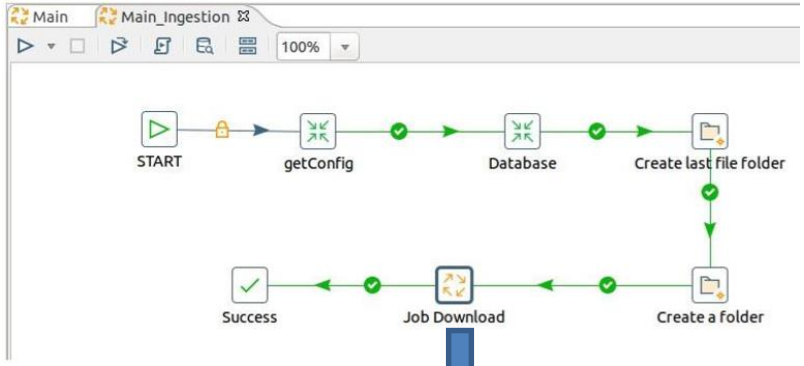


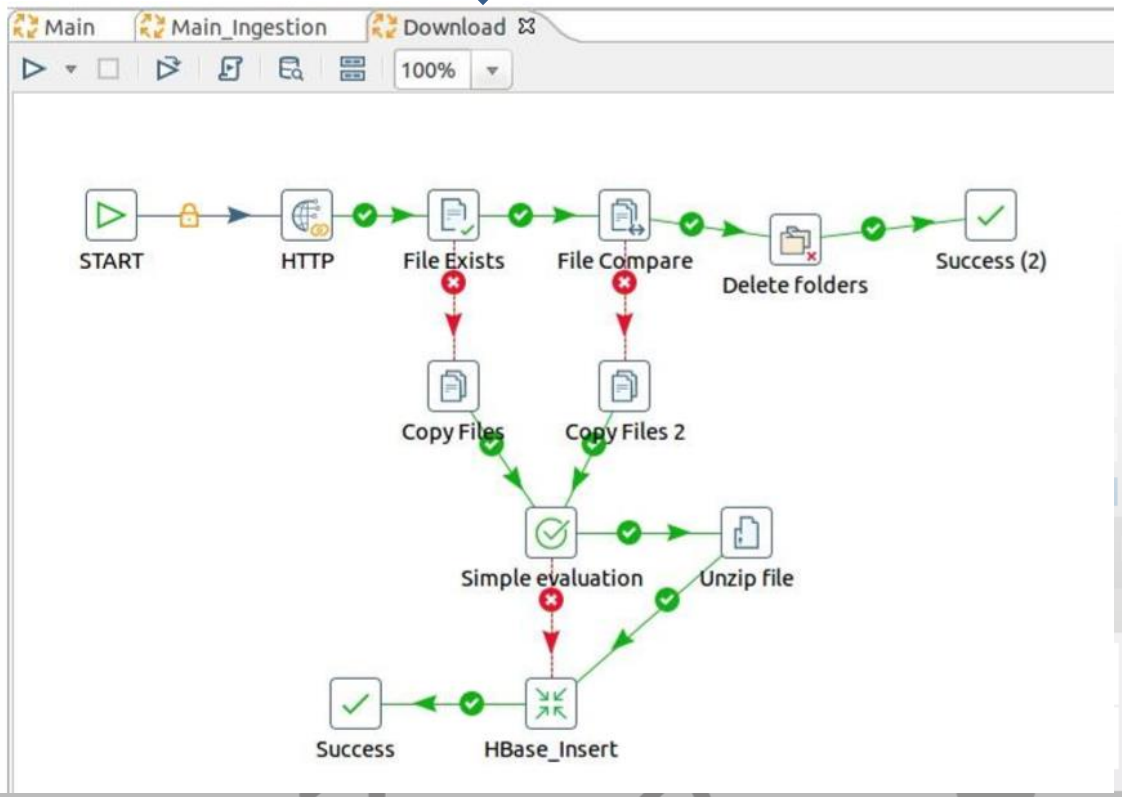
- Lo step getConfig si limita a caricare un file di configurazione e a settare delle variabili globali
- Lo step Database è di tipo Transformation Executor e richiama la trasformazione Database.ktr che ha la struttura riportata nella figura sottostante

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
http://www.disit.org

Ingestion (2)

UNIVERSITÀ DEGLI STUDI FIRENZE
MABIDA



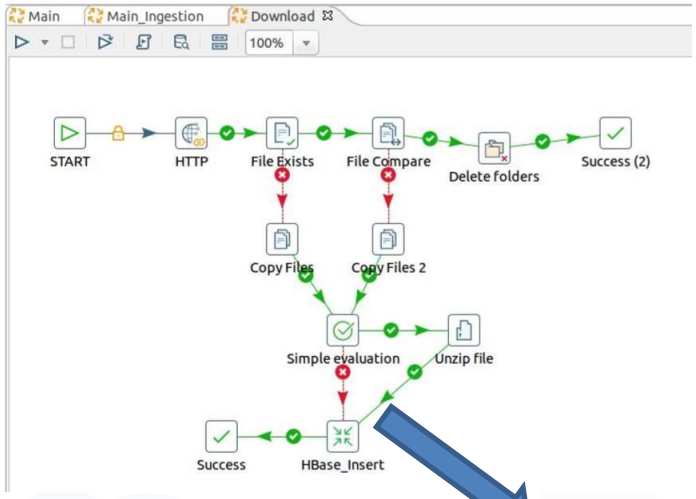


- Job Download:
  Nello step HTTP vengono impostati i campi:
  - 'URL', come indirizzo da cui scaricare il file di interesse;
  - 'Target', come percorso dove salvare tale file (ovviamente i percorsi sono relativi, ovvero vengono utilizzate le variabili settate in precedenza

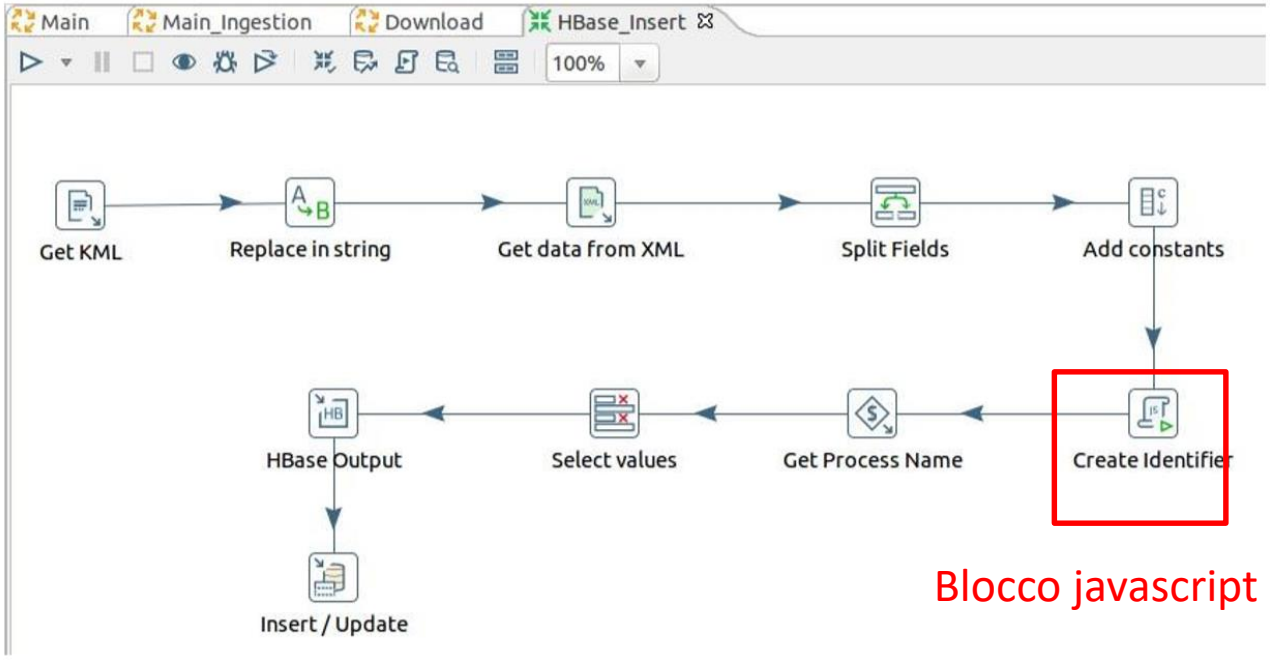
UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
http://www.disit.org

Ingestion (3)

UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA



- Nella figura sottostante si osserva la pipeline della trasformazione Hbase_insert:
  - Il dato in questo caso viene letto da un file.kml
  - viene "ripulito"
  - infine è inserito in HBase tramite lo step HBase Output



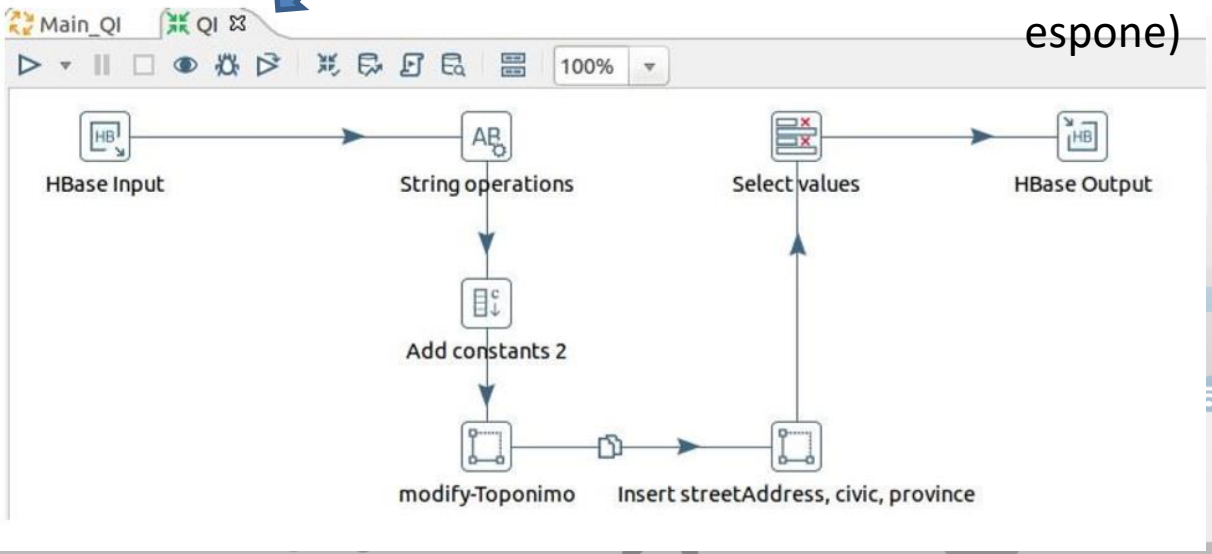
Blocco javascript

Quality Improvement

- La fase di Quality Improvement ha come obiettivo quello di migliorare la qualità dei dati acquisiti nella fase di ingestion, operando opportune modifiche sullo specifico dato in esame e memorizzandolo in una nuova tabella Hbase
- La struttura del Job Main_QI è riportata in Figura

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
http://www.disit.org
UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

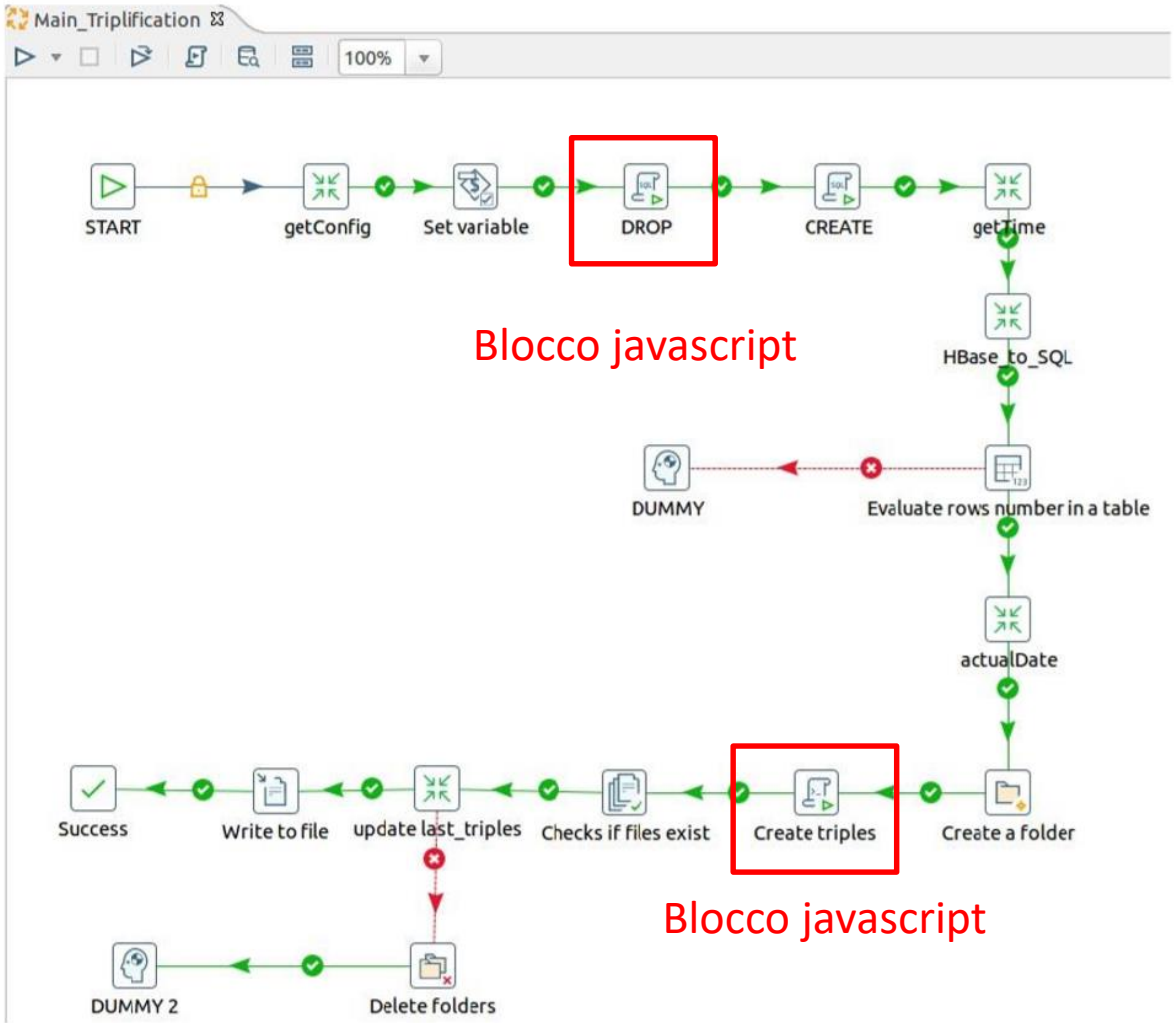
Quality Improvement (2)



- La trasformazione QI.ktr ha la responsabilità di aggiungere fondamentali proprietà al dato
- Tramite una query SPARQL vengono estratte informazioni come: coordinate (latitudine e longitudine), via, numero civico, città, cap, ma soprattutto il toponimo
- In pratica il dato viene arricchito con le informazioni mancanti (che il provider non espone)

Triplification

- La fase di Triplification ha l'obiettivo di generare un file.n3 contenente un insieme di triple RDF partendo dai dati acquisiti e processati nelle due fasi precedenti. Ciascuna informazione collezionata viene collegata alla ontologia KM4City, tramite il tool Karma.



Karma prevede:

- Creazione di un modello che faccia il mapping dei tipi di dati analizzati su una o più ontologie (nel nostro caso nella multi-ontologia KM4City)
- il mapping si effettua incrociando le ontologie con i dati contenuti in una tabella di un database MySQL (quindi per generare il modello Karma è necessario trasferire temporaneamente i dati da HBase a MySQL).
- L'applicazione del modello ai dati tramite il tool kitchen

Real Time

Scheduler

- For **Real Time data** (car parks, road sensors, etc.), the ingestion and triple generation processes should be performed periodically (no for **static data**).

- A scheduler is used to manage the periodic execution of ingestion and triple generation processes;
  - this tool throws the processes with a predefined interval determined in phase of configuration.

Scheduler: DISCES

Smart Cloud Engine
DISIT - Distributed Systems and Internet Technology Lab

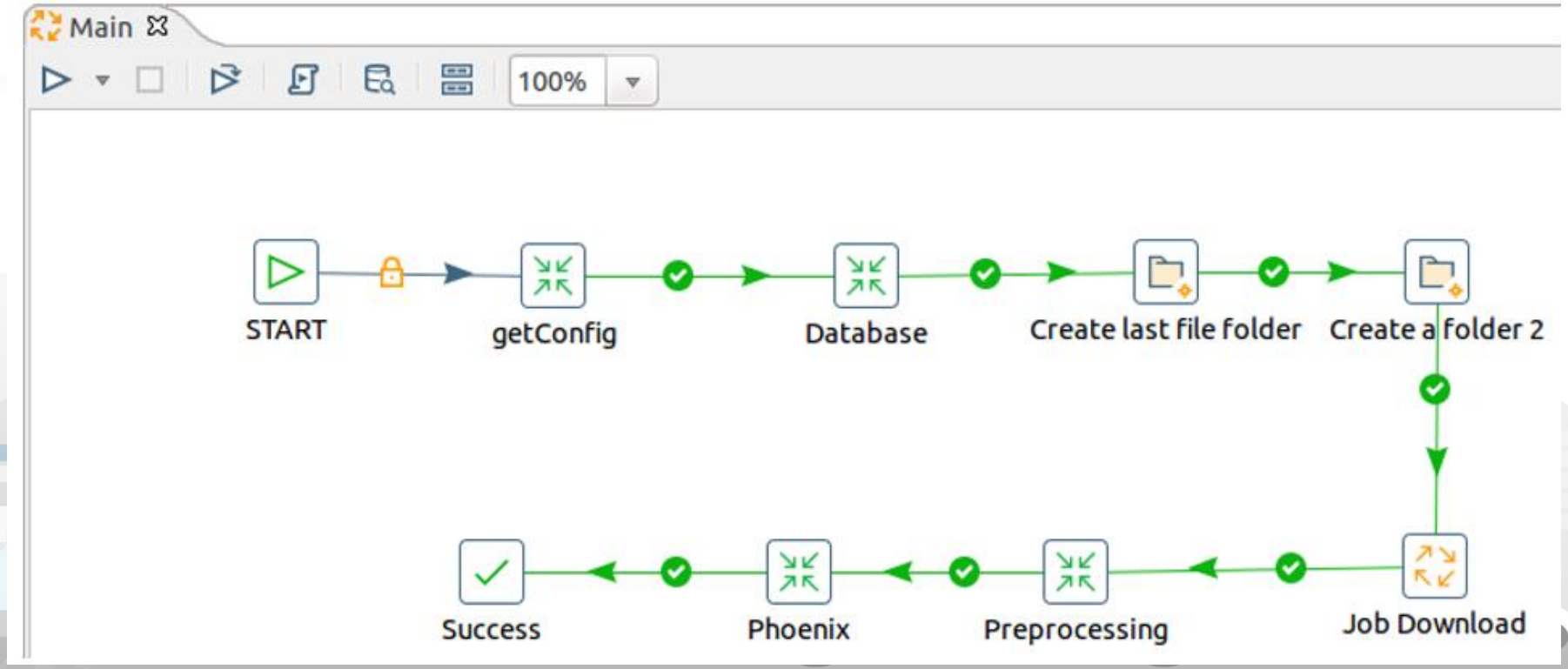
SCHEDULER NAME	ID	FIRE INSTANCE ID	DATE	JOB NAME	JOB GROUP	JOB DATA	STATUS	PROGRESS	TRIGGER NAME	TRIGGER GROUP	PREV FIRE TIME	NEXT FIRE
SCE	183652	hadoopnodet0715265676204441526567620531	2018-05-17 19:52:04	sensore_Demidoff_periferia	sensori_traffico_veicolare	#processParameters=[{"processPath":"/u	SUCCESS	100%	sensore_Demidoff_periferia_trigger	sensori_traffico_veicolare_trigger	2018-05-17 19:52:00	2018-05-17
SCE	183651	hadoopnodet0415265553660711526555366283	2018-05-17 19:50:02	Sensori_MeteoSIR_Phoenix	Sensori_Phoenix	#processParameters=[{"processPath":"//	FAILED		Sensori_MeteoSIR_Phoenix_trigger	Sensori_Phoenix	2018-05-17 19:50:00	2018-05-17
SCE	183650	hadoopnodet0715265676204441526567620530	2018-05-17 19:50:32	CarPark_MIIC_160_Phoenix	CarPark_Phoenix	#processParameters=[{"processPath":"/h	SUCCESS	100%	CarPark_MIIC_160_Phoenix_trigger	CarPark_Phoenix_trigger	2018-05-17 19:50:00	2018-05-17
SCE	183649	hadoopnodet04c1526555081135152655508 1132	2018-05-17 19:49:19	Bike_Siena_Phoenix	Bike_Phoenix	#processParameters=[{"processPath":"/h	SUCCESS	100%	Bike_Siena_Phoenix_trigger	Bike_Phoenix_trigger	2018-05-17 19:49:00	2018-05-17
SCE	183648	hadoopnodet0415265553660711526555366281	2018-05-17 19:49:38	smart_waste_FI_phoenix	smart_waste_phoenix	#processParameters=[{"processPath":"/h	SUCCESS	100%	8b095cf2-d5b3-4c93-890a-b6544d950542	137efde2-25b6-4b7c-95ef-1aaff2f8ea51	2018-05-17 19:49:00	2018-05-17
SCE	183647	hadoopnodet0715265676204441526567620528	2018-05-17 19:49:36	Bike_Pisa_Phoenix	Bike_Phoenix	#processParameters=[{"processPath":"/h	SUCCESS	100%	Bike_Pisa_Phoenix_trigger	Bike_Phoenix_trigger	2018-05-17 19:49:00	2018-05-17
SCE	183646	hadoopnodet0715265676204441526567620527	2018-05-17 19:49:05	Sensore_Senese_periferia	sensori_traffico_veicolare	#processParameters=[{"processPath":"/u	SUCCESS	100%	Sensore_Senese_periferia_trigger	sensori_traffico_veicolare_trigger	2018-05-17 19:49:00	2018-05-17
SCE	183645	hadoopnodet04c1526555081135152655508 1131	2018-05-17 19:48:06	Sensore_Senese_Centro	sensori_traffico_veicolare	#processParameters=[{"processPath":"/u	SUCCESS	100%	Sensore_Senese_Centro_trigger	sensori_traffico_veicolare_trigger	2018-05-17 19:48:00	2018-05-17
SCE	183644	hadoopnodet0415265553660711526555366280	2018-05-17 19:49:38	charging_stations_FI_phoenix	charging_stations_phoenix	#processParameters=[{"processPath":"/h	SUCCESS	100%	charging_stations_FI_phoenix_trigger	charging_stations_phoenix_trigger	2018-05-17 19:47:00	2018-05-17
SCE	183643	hadoopnodet0715265676204441526567620526	2018-05-17 19:48:10	weather_sensor_SantaMarta_FI_phoenix	weather_sensor_phoenix	#processParameters=[{"processPath":"/h	SUCCESS	100%	weather_sensor_SantaMarta_FI_phoenix_trigger	weather_sensor_phoenix_trigger	2018-05-17 19:46:00	2018-05-17
SCE	183642	hadoopnodet0415265553660711526555366278	2018-05-17 19:46:58	CarPark_SienaParcheggi_Phoenix	CarPark_Phoenix	#processParameters=[{"processPath":"/h	SUCCESS	100%	CarPark_SienaParcheggi_Phoenix_trigger	CarPark_Phoenix_trigger	2018-05-17 19:45:00	2018-05-17
SCE	183641	hadoopnodet0715265676204441526567620520	2018-05-17 19:47:58	CarPark_MIIC_164_Phoenix	CarPark_Phoenix	#processParameters=[{"processPath":"/h	SUCCESS	100%	CarPark_MIIC_164_Phoenix_trigger	CarPark_Phoenix_trigger	2018-05-17 19:45:00	2018-05-17

Ingestion RT

- Il processo di acquisizione dei dati nel caso real time, comprende una unica fase durante la quale il dato viene:
  - scaricato dal provider
  - Processato
  - inserito in una tabella Hbase

In Figura è possibile osservare la struttura interna del file Main.kjb

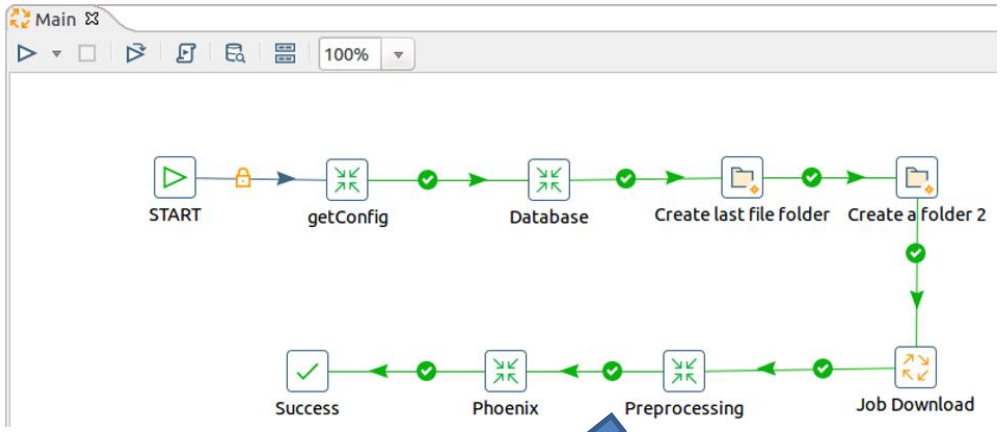
- Lo st4ep download è come quello visto per il caso statico



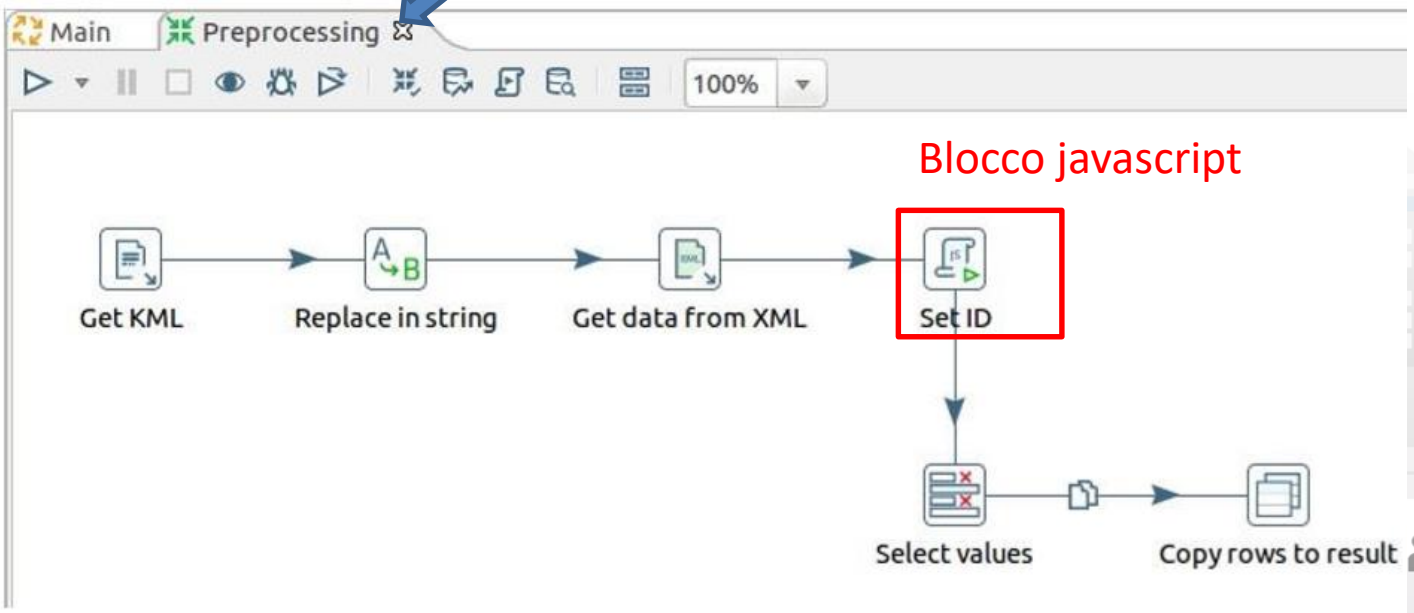
UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
http://www.disit.org

Ingestion RT (2)

UNIVERSITÀ DEGLI STUDI FIRENZE
MABIDA



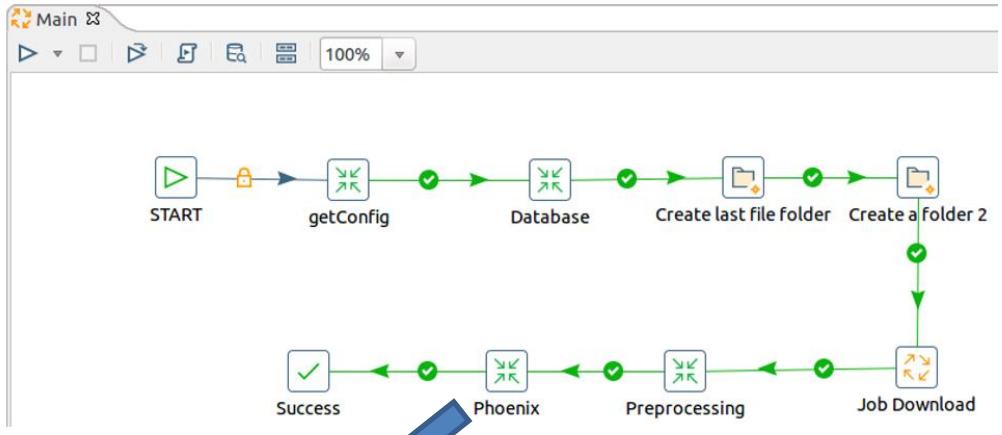
- Lo step Preprocessing in questo esempio ha soltanto il compito di selezionare l'informazione RealTime e di settare un identificativo per ogni riga del dataset
- Tale identificativo è identico a quello utilizzato durante l'acquisizione della parte statica
- Questo campo è fondamentale perché collega il dato statico e quello RealTime

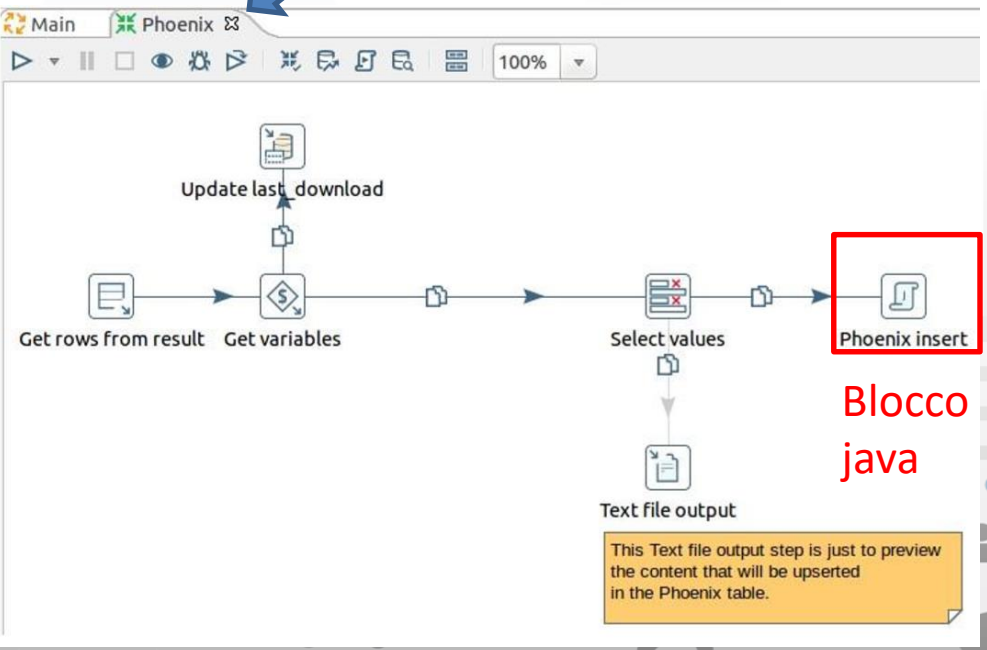


UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
http://www.disit.org

Ingestion RT (3)

UNIVERSITÀ DEGLI STUDI FIRENZE
MABIDA

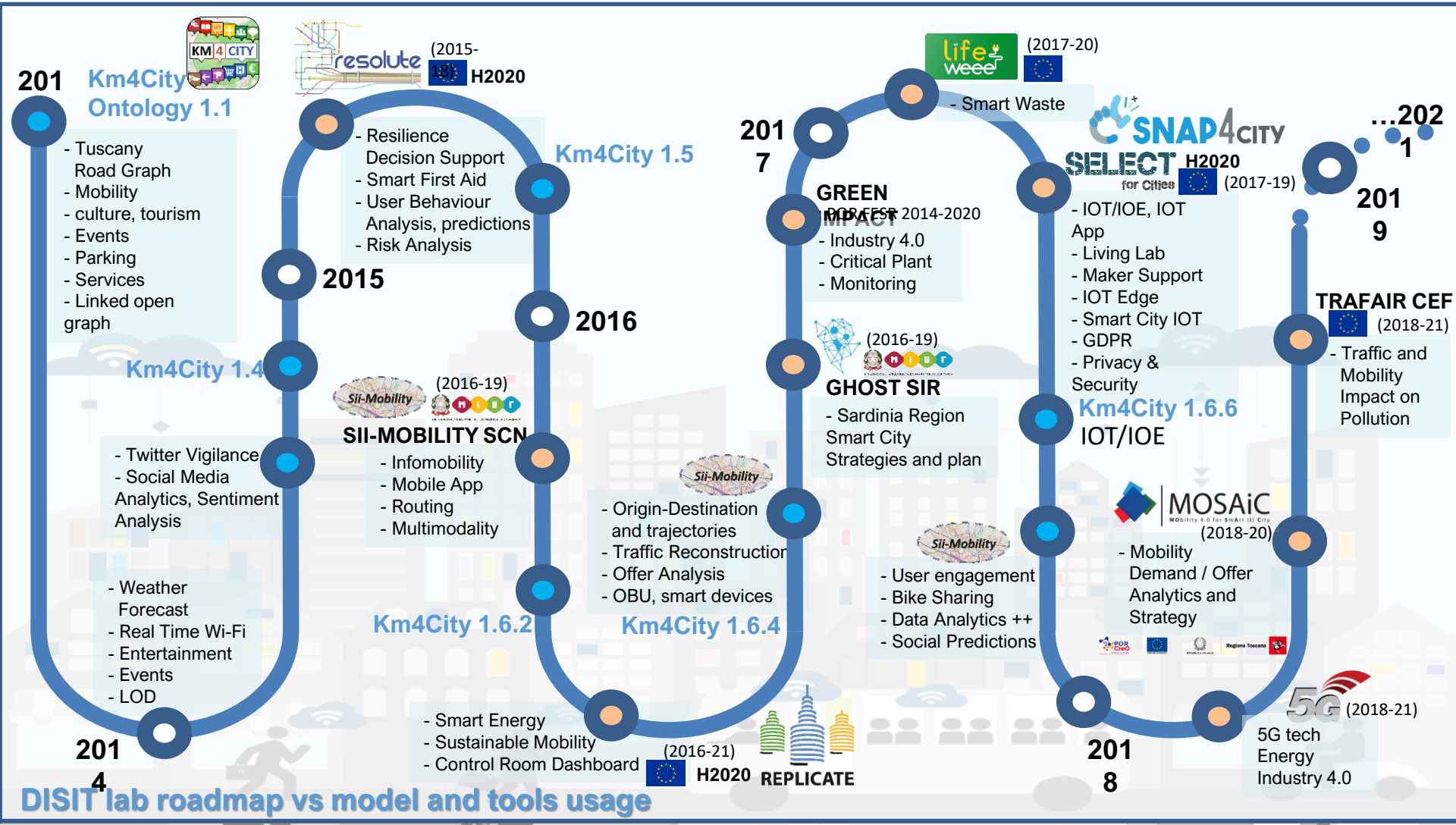




Blocco java

- dallo stream il dato viene letto
- attraverso lo step Phoenix insert viene inserito (mediante la funzione upsert) in una tabella Hbase
- Tale tabella deve essere creata da riga di comando

Services/ Projects



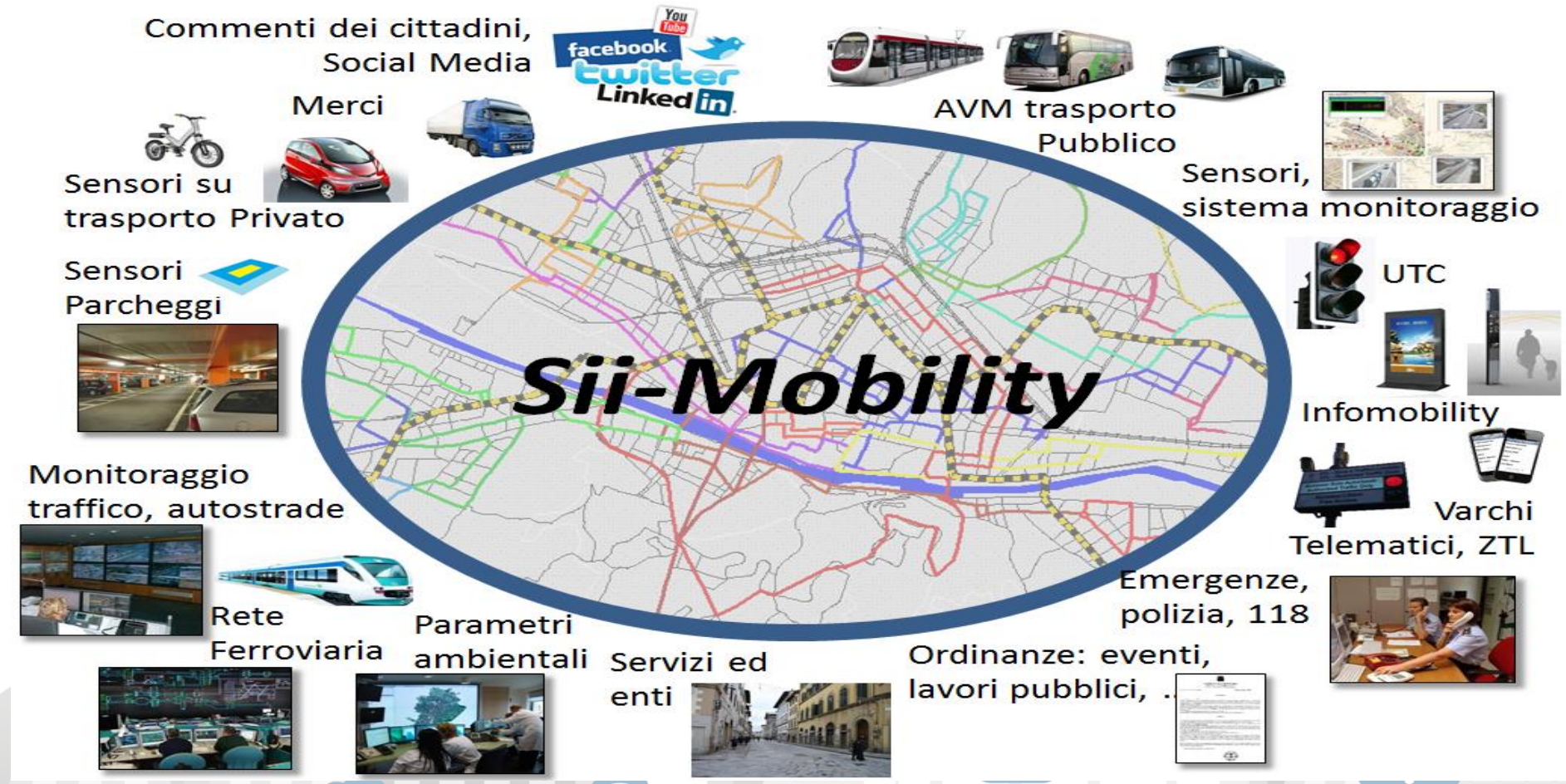
DISIT lab roadmap vs model and tools usage

Sii-Mobility

KM 4 CITY

MIUR MINISTERO DELL'ISTRUZIONE DELL'UNIVERSITÀ E DELLA RICERCA

http://www.Sii-Mobility.org

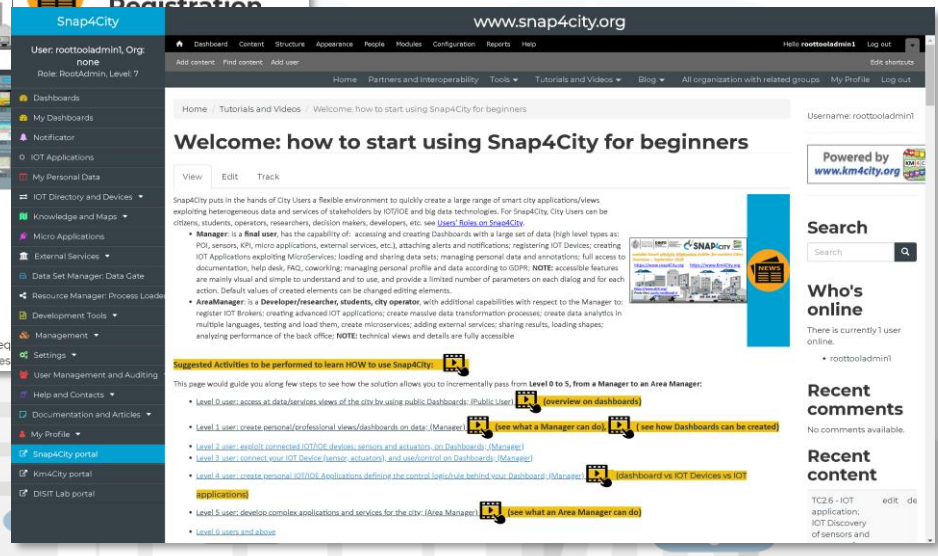


UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB http://www.disit.org

SNAP4CITY

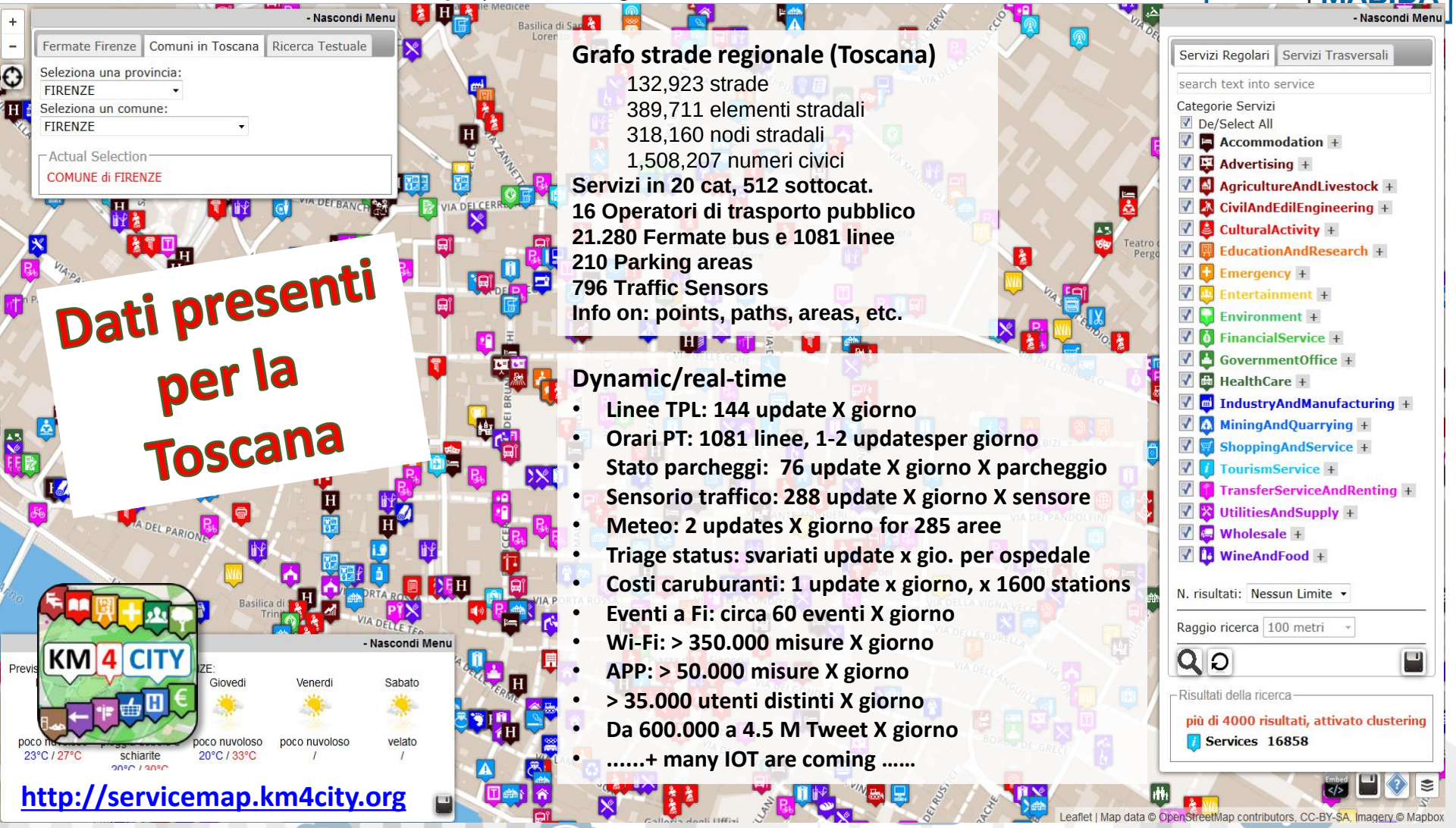
UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

Https://www.snap4city.org



Https://www.km4city.org

http://servicemap.disit.org



Dati presenti per la Toscana

Grafo strade regionale (Toscana)

- 132,923 strade
- 389,711 elementi stradali
- 318,160 nodi stradali
- 1,508,207 numeri civici

Servizi in 20 cat, 512 sottocat.
16 Operatori di trasporto pubblico
21.280 Fermate bus e 1081 linee
210 Parking areas
796 Traffic Sensors
Info on: points, paths, areas, etc.

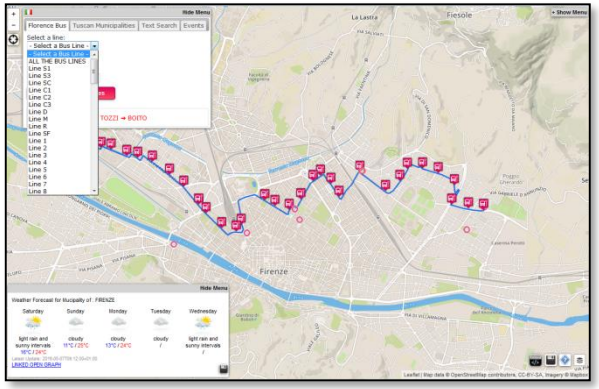
Dynamic/real-time

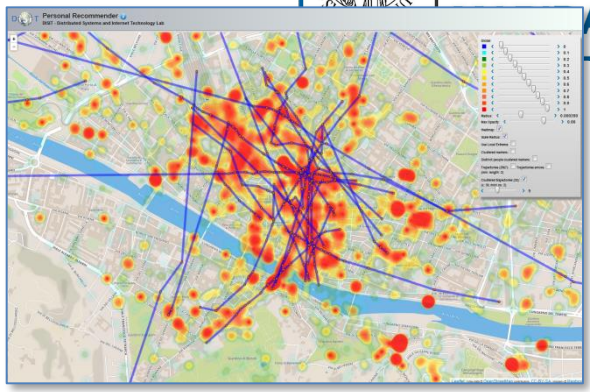
- **Linee TPL: 144 update X giorno**
- **Orari PT: 1081 linee, 1-2 updatesper giorno**
- **Stato parcheggi: 76 update X giorno X parcheggio**
- **Sensorio traffico: 288 update X giorno X sensore**
- **Meteo: 2 updates X giorno for 285 aree**
- **Triage status: svariati update x gio. per ospedale**
- **Costi caruburanti: 1 update x giorno, x 1600 stations**
- **Eventi a Fi: circa 60 eventi X giorno**
- **Wi-Fi: > 350.000 misure X giorno**
- **APP: > 50.000 misure X giorno**
- **> 35.000 utenti distinti X giorno**
- **Da 600.000 a 4.5 M Tweet X giorno**
- **.....+ many IOT are coming .....**

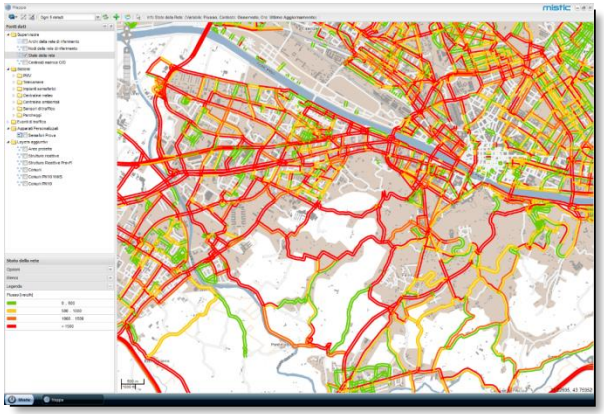
http://servicemap.km4city.org

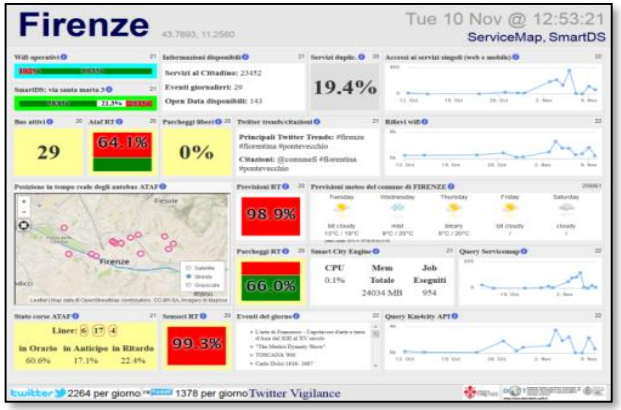
Smart City Dashboard

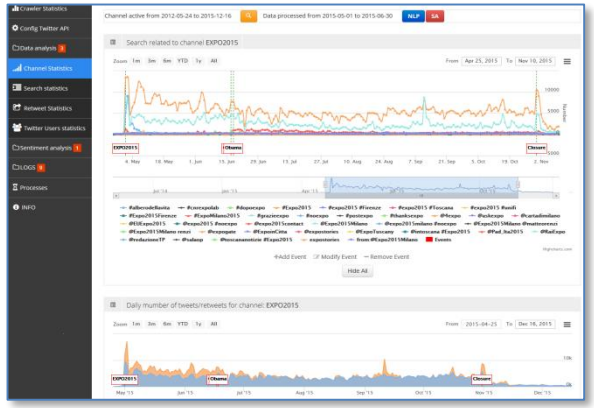
UNIVERSITÀ DEGLI STUDI FIRENZE

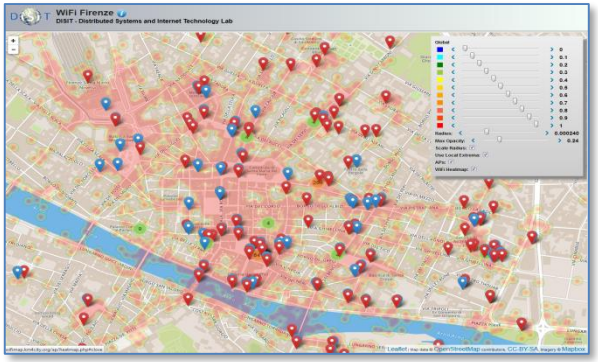


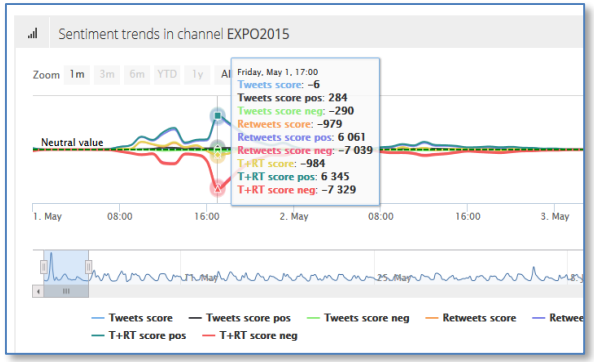








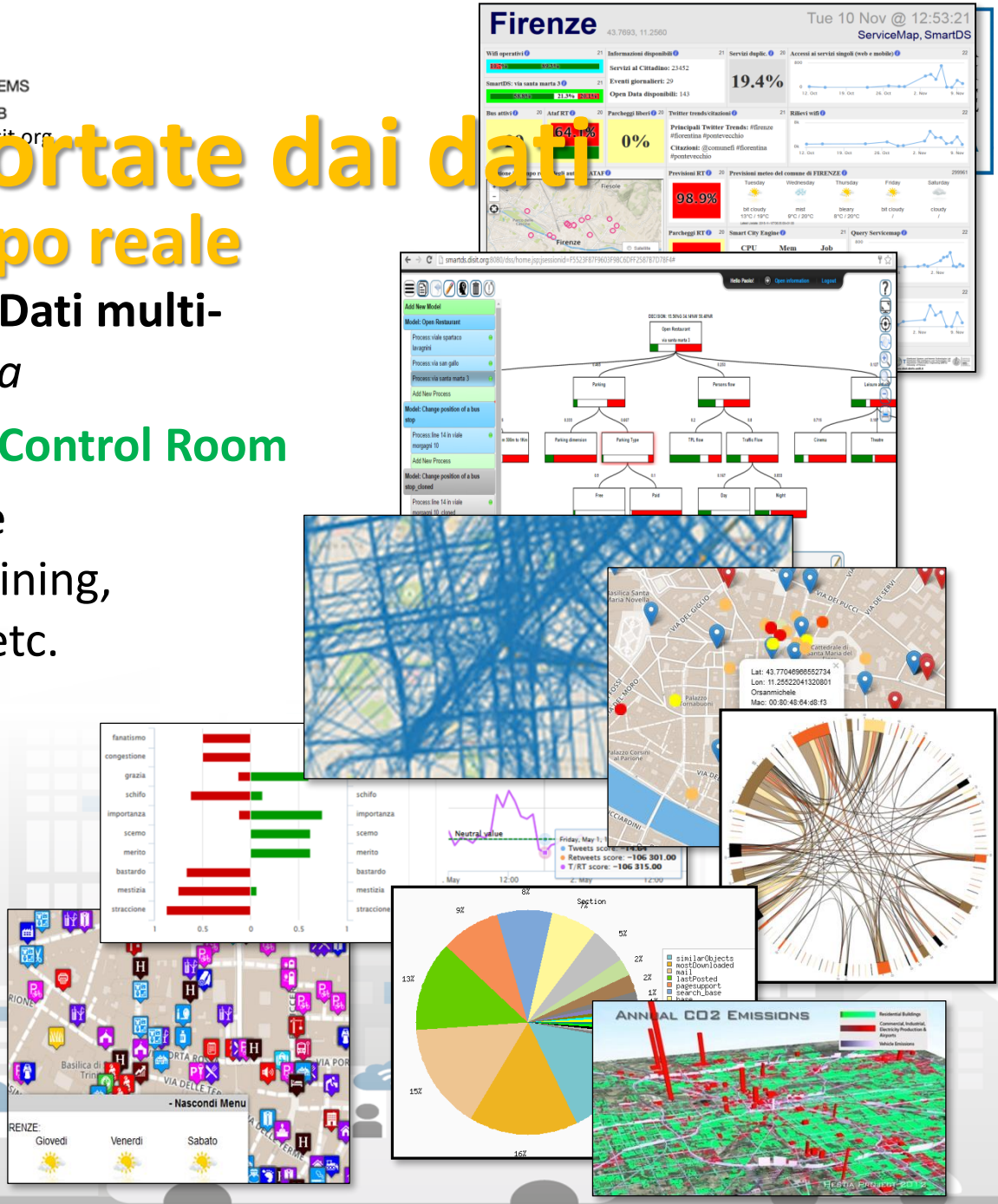




UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
http://www.disit.org

Decisioni supportate dai dati periodiche ed in tempo reale

- **Condivisione e Integrazione Dati multi-dominio:** *semantica e bigdata*
- **Dati →Smart City Engine → Control Room**
- **analisi**: monitoraggio, flussi e comportamenti, sondaggi, mining, correlazioni, cause – effetti, etc.
  - Per il miglioramento di servizi correnti
  - Per reagire ad eventi, incremento della resilienza,
  - Per la creazione servizi innovativi
  - …



SMART CITIES for a better world
FIRENZE. Venerdì 26 febbraio 2016

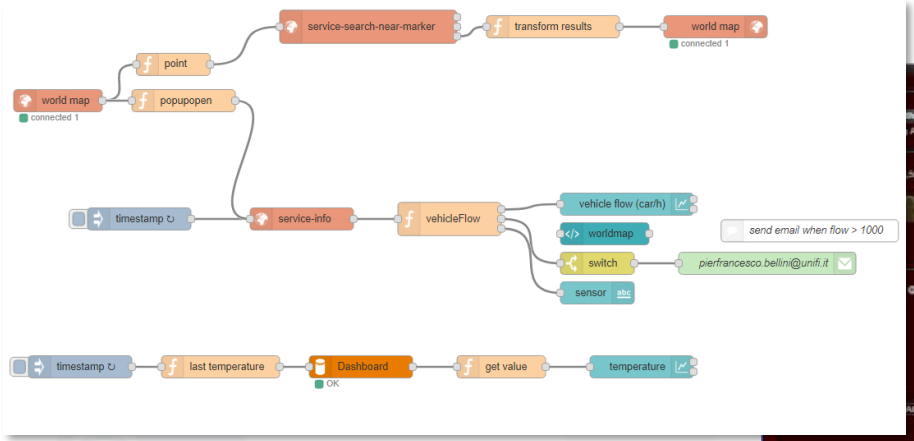
UNIVERSITÀ DEGLI STUDI FIRENZE

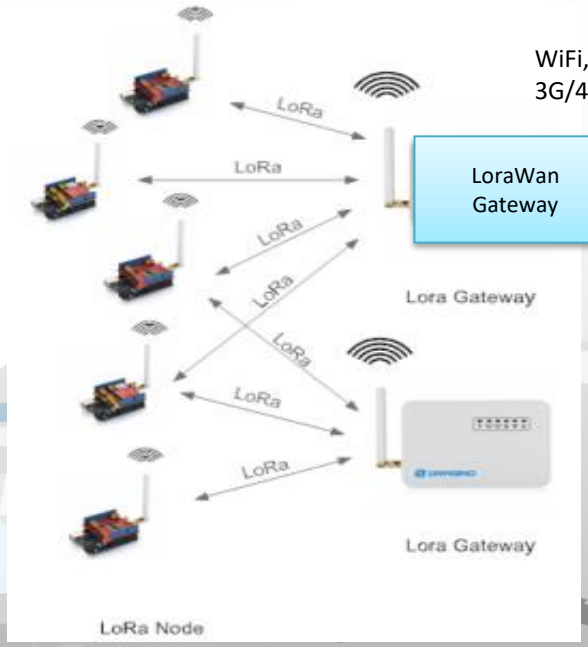
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
http://www.disit.org

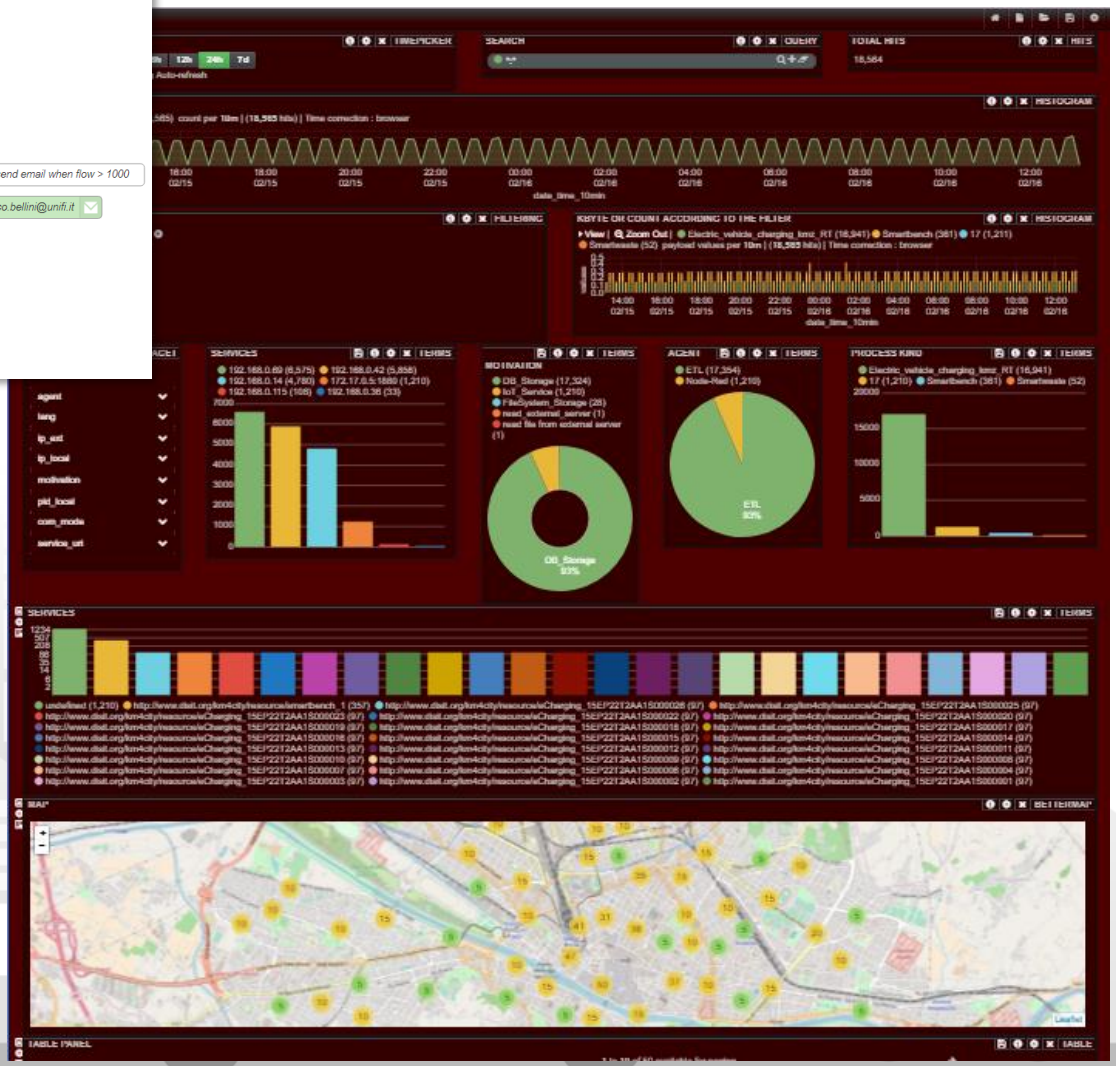
UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

Internet of Things





WiFi, RJ45
3G/4G



Bibliography

- http://www.pentaho.com
- km4city, the DISIT Knowledge Model for City and Mobility
- Pentaho Kettle solutions – Building Open Source ETL Solutions with Penthao Data Integration. Matt Casters, Roland Bouman, Jos Van Dongen. Wiley.

UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB http://www.disit.org

UNIVERSITÀ DEGLI STUDI FIRENZE MABIDA

Big Data

Michela Paolucci

University of Florence, Department of Information Engineering,

DISIT Lab, http://www.disit.org, http://www.sii-mobility.org,

paolo.nesi@unifi.it, michela.paolucci@unifi.it,

MASTER BIG DATA, ANALYTICS AND TECHNOLOGIES FOR MANAGEMENT