Estrazione automatica di ontologie da librerie Java per documentazione machine-readable di API

Motivazioni – Caso pratico

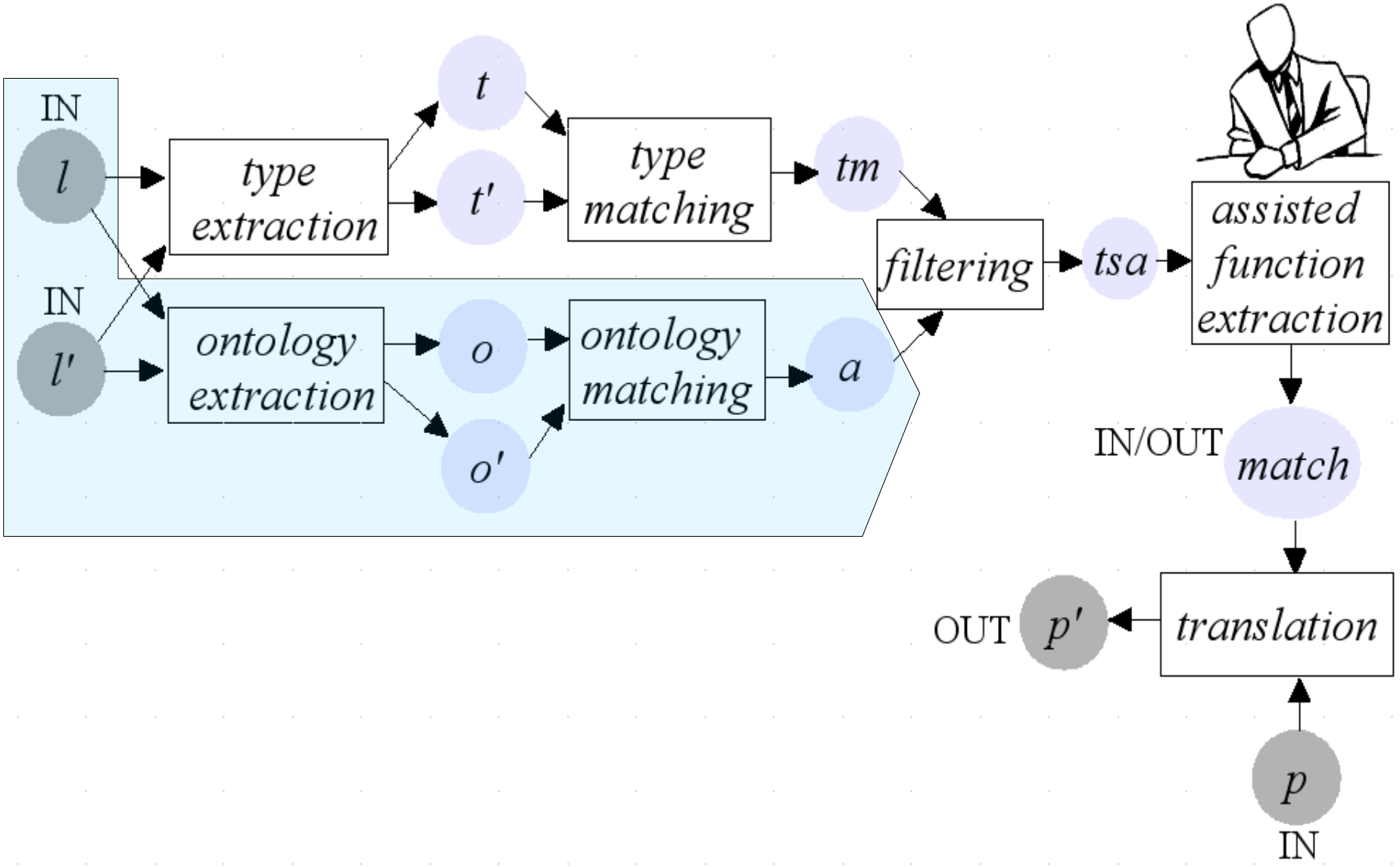
- Supporto allo sviluppatore nella migrazione di codice in modo da mantenere:
  - Correttezza dei tipi (*type safety*)
  - Corrispondenze semantiche

- Esempio: Nano XML 1.6.8 e Nano XML 2.2.3

```
public class XMLElement extends Object
{
...
public String getTagName ()
{
return this . tagName ;
}
...
public String getContents ()
{
return this . contents ;
}
...
}
```

```
public class XMLElement extends Object
{
...
public String getName ()
{
return this . tagName ;
}
...
public String getContents ()
{
return this . contents ;
}
...
public int getLineNr ()
{
return this . lineNr ;
}
```

Architettura del sistema



Strumenti usati (1): Java

Linguaggio di programmazione:

- Oggetto di studio in questo lavoro
- Usato per scrivere l'estrattore

Perchè Java?

- Molto diffuso
- Object Oriented
- Fortemente tipato
- API molto estesa
- Molte librerie a disposizione

Strumenti usati (2): Javadoc

Generazione di documentazione

- analizza il codice sorgente
- genera documentazione “guidata” dalla doclet

Perchè Javadoc?

- Rispecchia la filosofia del lavoro svolto
- Estrattore realizzato = “estensione” di Javadoc
- Pensato per chi sviluppa codice (accesso al sorgente)
- Estrazione di informazioni aggiuntive oltre agli elementi del linguaggio

Strumenti usati (3): Ontologie

Descrizioni formali delle entità di un dominio

Gruber (1993):

Un'ontologia è una **specifica esplicita** di una **concettualizzazione**

Un'ontologia "parla" del suo dominio, descrivendo:

- i suoi **concetti** essenziali (termini, vocabolario)
- la loro classificazione (tassonomia)
- le loro **proprietà (attributi)**
- creando **istanze** di concetti e proprietà

usando un linguaggio

- formale
- dichiarativo

} OWL

Strumenti usati (3): OWL

Linguaggio più diffuso per rappresentare ontologie.

- *Namespaces:* ereditati da XML
  Metodo per qualificare elementi e attributi, associandoli a delle URI, permettendo di avere identificatori sempre univoci e mantenendo l'ontologia leggibile

- *Classi*
  Una classe definisce un gruppo di individui che condividono stesse caratteristiche
- *Sottoclassi*
  Permettono di definire gerarchie tra classi
- *Proprietà*
  Definiscono relazioni tra individui (*ObjectProperty*) o mettono in relazione individui e valori (*DatatypeProperty*)

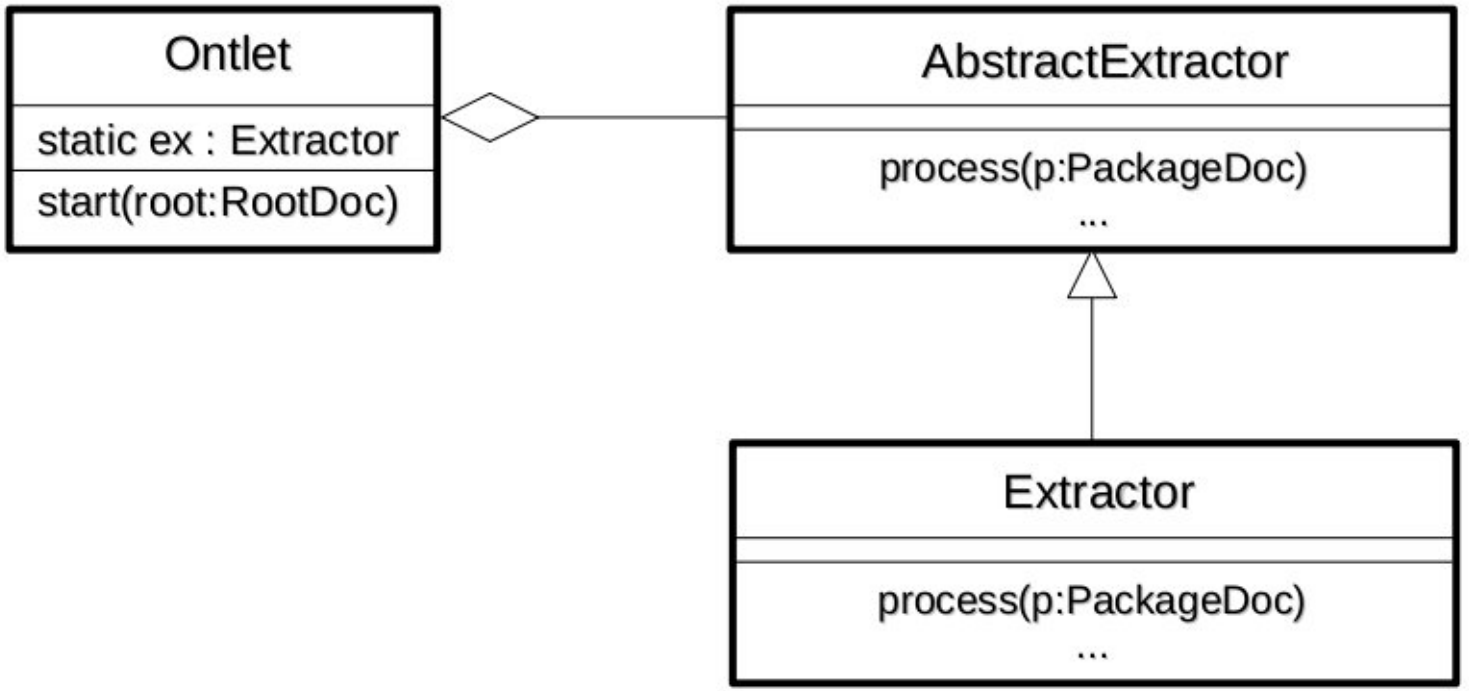
Strumenti usati (4): Jena

Framework per uso di ontologie

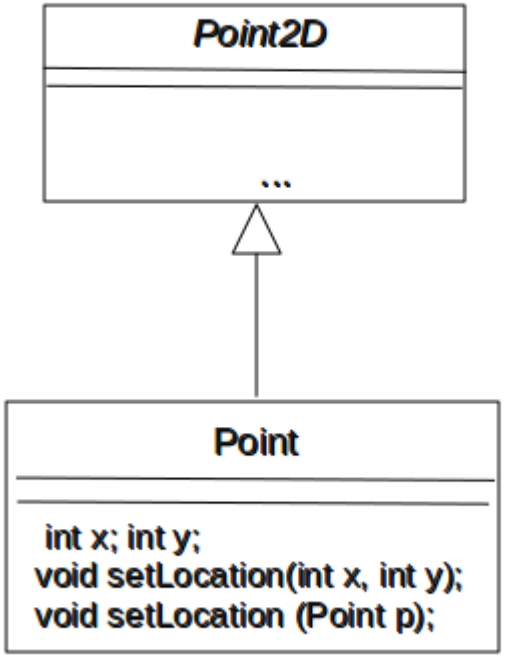
- Scritto in Java
- Fornisce API per manipolazione di ontologie (scritte anche in OWL)
- Lettura/scrittura di RDF in vari formati

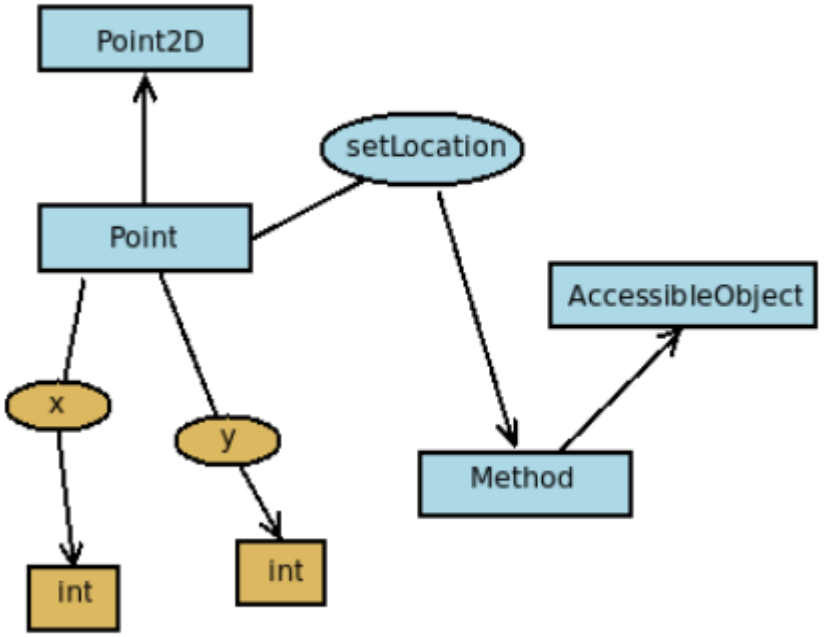
Sono state usate le API per la generazione di ontologie; il framework non è stato esteso.

Svolgimento del lavoro: progettazione



Svolgimento del lavoro: scelte progettuali





Classi

```
<owl:Class rdf:about="java.awt.geom:Point2D#"/>

<owl:Class rdf:about="java.awt:Point">
 <rdfs:comment>
   A point representing a location in (x, y)     coordinate space,
specified in integer        precision.
 </rdfs:comment>
 <rdfs:subClassOf
rdf:resource="java.awt.geom:Point2D#"/>
</owl:Class>
```

Campi

```
<owl:DatatypeProperty rdf:about="java.awt:Point#x">
 <rdfs:comment>The x coordinate.
  If no x coordinate is set it will default to 0.
 </rdfs:comment>
 <rdfs:range rdf:resource="&xsd;int"/>
 <rdfs:domain rdf:resource="java.awt:Point"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="java.awt:Point#y">
 <rdfs:comment>The y coordinate.
 If no y coordinate is set it will default to 0.</rdfs:comment>
<rdfs:range rdf:resource="&xsd;int"/>
<rdfs:domain rdf:resource="java.awt:Point"/>
</owl:DatatypeProperty>
```

Metodi

```
<owl:ObjectProperty
rdf:about="java.awt:Point#setLocation">
  <sem:paramReturnType>
    … <!-- vedere più avanti -->
  </sem:paramReturnType>

  <rdfs:domain              rdf:resource="java.awt:Point"/>
  <rdfs:range  rdf:resource="java.lang.reflect:Method"/>

</owl:ObjectProperty>
```

Elementi del linguaggio considerati

- Modificatori di accesso: public e protected
- Deprecati: tag @deprecated e Annotazioni @Deprecated rese con DeprecatedClass e DeprecatedProperty o con apposito commento
- *Namespaces:* i namespace di Java sono mantenuti nell'ontologia estratta

```
package pckg.test;
public class C
{
...
}
```

```
<rdf:RDF
<!ENTITY pckg.test = “file://.../pckg/test#”; >
<!ENTITY pckg.test.C = “file://.../pckg/test/C#”;>
 xmlns:pckg.test=“&pckg.test;"
 xmlns:pckg.test.C="&pckg.test.C;"  >
</rdf:RDF >
```

- Sono stati considerati tutti gli elementi del linguaggio riconosciuti da Javadoc tranne tipi generici, overriding, annotazioni, proprietà statiche.

Osservazioni su alcuni elementi considerati

- **Array**: owl:ObjectProperty con range in classe corrispondente a java.lang.reflect.Array
- owl:Annotation (arrayTypeAndDim) che raccoglie le informazioni sul tipo degli elementi e dimensione dell'array.
- Si mantiene la semplicità dell'ontologia estratta e non si perdono informazioni importanti

Osservazioni su alcuni elementi considerati

- **Metodi**: owl:ObjectProperty con range in classe corrispondente a java.lang.reflect.Method

  owl:Annotation (paramReturnType) che raccoglie le informazioni sul metodo, sotto forma di una lista di liste contenenti:
  - *is-deprecated/is-not-deprecated*,
  - *tipo_rit*,
  - *tipo_p_1*,*tipo_p_2*,...,*tipo_p_n*
- Si mantiene la semplicità della traduzione: non si distingue tra metodi astratti, statici; gli overloading sono mantenuti nell'annotazione, in quanto informazioni importanti sfruttabili da ulteriori componenti specifici

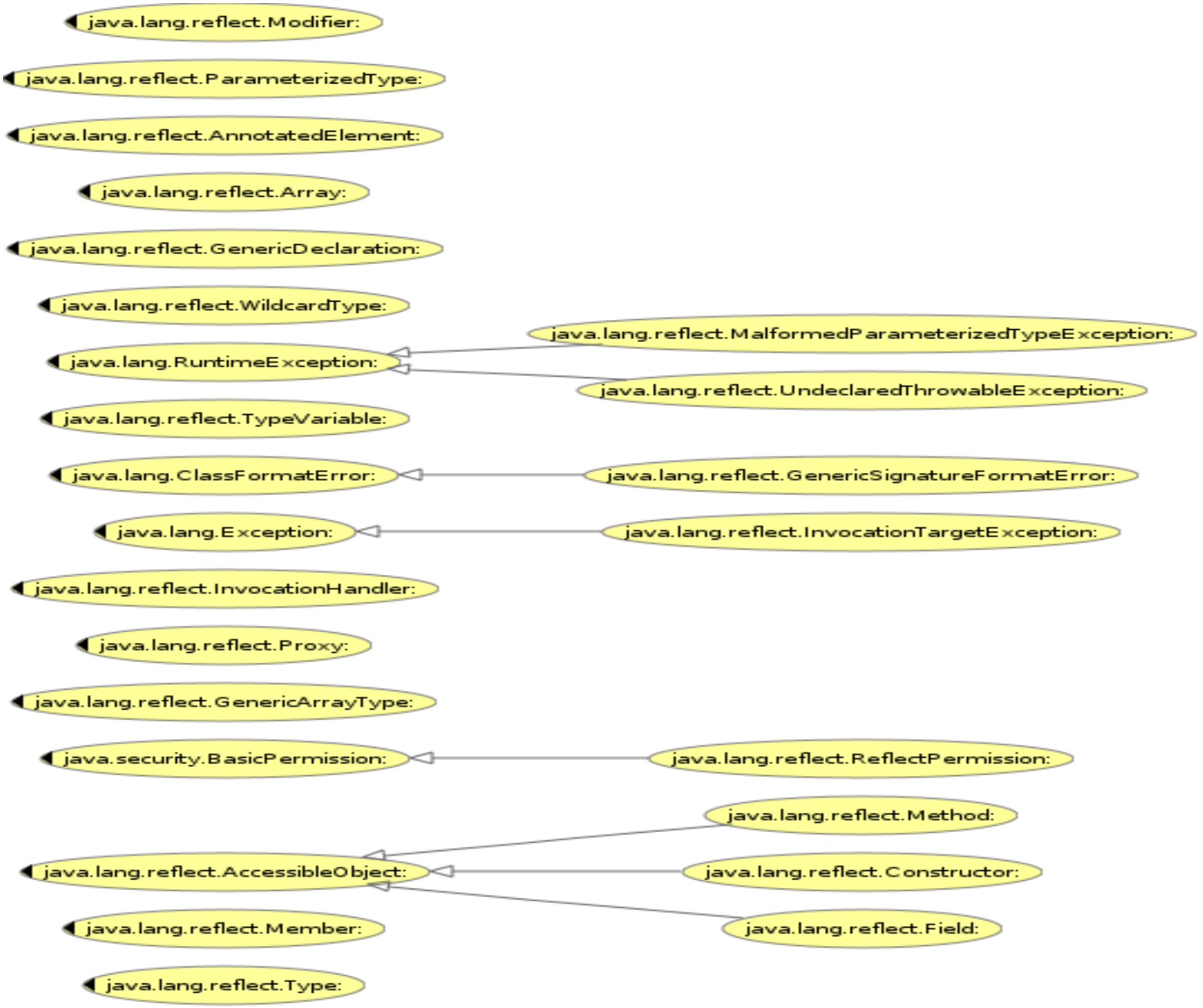
Processo di estrazione

- Invocazione di Ontlet da riga di comando, con vari parametri
- Istanziazione di Extractor e invocazione del metodo execute(), con parametro di tipo RootDoc
- Istanziazione di OntModel, creazione di classi generali dell'ontologia
- Per ogni package passato a Javadoc, vengono processate tutte le classi
- Per ciascuna classe vengono processati i suoi campi e metodi
- Scrittura su file dell'OntModel costruito e man mano popolato

Esperimenti: generazione ontologia per Java API

- Sorgenti di Java SE 1.6
- Package java e suoi sottopackage
- Dimensione file creato: 2,2 MB
- Tempo di elaborazione su macchina Linux Ubuntu 9.10, processore Intel Core 2 1,73GHz: 46 secondi circa

Ontologia per java.lang



Esperimenti: esempio

```
[...]
 <!ENTITY java.util.jar.JarFile "http://dummy/ombre/it/code/java/util/jar/JarFile#">
...
xmlns:java.util.jar.JarFile="&java.util.jar.JarFile;"
[...]
 <owl:Class rdf:about="&java.util.jar.JarFile;">
   <rdfs:subClassOf>
     <owl:Class rdf:about="&java.util.zip.ZipFile;"/>
       </rdfs:subClassOf>
   </owl:Class>
  <owl:ObjectProperty rdf:about="&java.util.jar.JarFile;getManifest">
    <sem:paramReturnType rdf:parseType="Collection">
      <rdf:Description>
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:resource="&rdf;nil"/>
          <rdf:first>java.util.jar.Manifest</rdf:first>
        </rdf:rest>
        <rdf:first>is-not-deprecated</rdf:first>
      </rdf:Description>
    </sem:paramReturnType>
    <rdfs:comment>Returns the jar file manifest, or null if none.</rdfs:comment>
    <rdfs:range rdf:resource="&java.lang.reflect.Method;"/>
    <rdfs:domain rdf:resource="&java.util.jar.JarFile;"/>
  </owl:ObjectProperty>
```

Esperimenti: allineamenti - Def

Allineamento (ontology matching): processo di individuazione di corrispondenze tra concetti di ontologie diverse.

Per ***corrispondenza*** tra E1 di ontologia 1 ed E2 di ontologia 2 si intende una quintupla

<Id, E1, E2, Rel, Conf>

Dove:

- Id: identificatore univoco della corrispondenza
- E1 ed E2 sono le entità delle ontologie
- Rel è una relazione che vale tra E1 ed E2 (*equivalenza*, più generale, disgiunzione...)
- Conf è una misura di confidenza che vale per la corrispondenza tra E1 e E2

Esperimenti: allineamenti - metodi

- Abbiamo usato metodi di allineamento “**name-based**”
  - *String-based:* misurano la somiglianza tra coppie di entità guardando le stringhe che li etichettano
    - *Substring distance*: misura il rapporto della sottostringa più lunga comune alle due stringhe rispetto alla loro lunghezza
    - *N-gram distance* due stringhe sono più simili quanti più n-grammi hanno in comune
    - SMOA somiglianza tra due stringhe è una funzione delle loro comunanze e differenze
  - *Language-based* usano algoritmi di elaborazione del linguaggio naturale. La somiglianza è vista come pezzi di testo significativi. Molti metodi usano WordNet

Esperimenti: allineamenti

- 4 coppie di librerie
  - Nano XML 1.6.8 e 2.2.3
  - java.net 1.5 e 1.6
  - java.security 1.5 e 1.6
  - java.awt 1.5 e 1.6

da cui si sono estratte 8 ontologie

- Su ciascuna coppia sono stati eseguiti 5 algoritmi di allineamento di ontologie

Allineamenti: considerazioni

Gli esperimenti sono stati condotti seguendo due assunzioni differenti:

- Solo le corrispondenze tra elementi deprecati della vecchia lib e i rimpiazzi suggeriti sono considerate corrette
- Sono considerate corrette le corrispondenze di cui sopra così come quelle “identiche”

Sono stati costruiti degli allineamenti di riferimento per misurare la bontà dei risultati ottenuti dagli algoritmi di allineamento

Allineamenti: considerazioni

Per ogni coppia di librerie considerata, *per ciascuna delle due assunzioni* esposte prima, *per ciascun algoritmo di allineamento* eseguito si considerano tre misure

- **Precision**: *il numero di corrispondenze corrette* (rispetto all'allineamento di riferimento) trovate *diviso* per *il numero totale* di corrispondenze trovate
- **Recall**: *il numero di corrispondenze corrette diviso per il numero di corrispondenze attese*
- **F-measure**: la media armonica di precision e recall

Allineamenti: risultati

		Matching algorithm			
Experiment	**Indicator**	**Substr**	**N-gram**	**SMOA**	**WordNet**
java.awt with deprecated	Precision	0.61	0.63	0.59	0.61
	Recall	0.54	0.51	0.51	0.53
	F-measure	0.57	0.56	0.55	0.57
java.awt without deprecated	Precision	0.61	0.63	0.59	0.60
	Recall	0.54	0.51	0.52	0.54
	F-measure	0.57	0.56	0.55	0.57
java.net with deprecated	Precision	0.67	0.70	0.67	0.67
	Recall	0.60	0.59	0.60	0.60
	F-measure	0.63	0.64	0.63	0.63
java.net without deprecated	Precision	0.66	0.70	0.67	0.66
	Recall	0.60	0.59	0.60	0.60
	F-measure	0.63	0.64	0.63	0.63
java.security with deprecated	Precision	0.50	0.57	0.50	0.50
	Recall	0.44	0.43	0.44	0.44
	F-measure	0.47	0.49	0.47	0.47
java.security without deprecated	Precision	0.47	0.54	0.47	0.47
	Recall	0.44	0.42	0.44	0.44
	F-measure	0.46	0.47	0.45	0.45
Nano XML with deprecated	Precision	0.20	0.22	0.20	0.20
	Recall	0.15	0.15	0.15	0.15
	F-measure	0.17	0.18	0.17	0.17
Nano XML without deprecated	Precision	0.23	0.27	0.23	0.23
	Recall	0.23	0.23	0.23	0.23
	F-measure	0.23	0.25	0.23	0.23

Lavori futuri

- Ulteriori esperimenti
  - con altri matcher
  - con altre coppie di librerie

  .... che porteranno a decidere

- eventuali ulteriori arricchimenti dell'ontologia estratta
- Implementazione degli altri componenti del sistema

Lavori correlati

Anno	Autori	Met	Cosa modella	Da	A	Scopo
1991	Devanbu, Brachman, Selfridge	M	Dominio applicativo	Linguaggio naturale, codice	KANDOR	Supporto a documentazione codice
2004	Zimmer, Rauschmayer (TUNA)	M	Strutture codice sorgente + istanze	Linguaggio Java-like	UML, Topic Map	Supporto a sviluppo del codice, fondendo MDA e XP
2004	Eberhart, Agarwal (SmartAPI)	SA	Concetti dell'API + istanze	Java	OWL	Supporto nella stesura del codice
2005	Dietrich, Elgar	SA	Concetti dei Design Pattern	Java	OWL	Supporto manutenzione codice (individuaz. Design Pattern)
2007	Witte, Li, Zhang, Rilling	SA	Cooncetti del dominio, Concetti software	Java	OWL	Supporto a mantenimento codice
2008	Yu, Zhan, Yi, Li, Wang	A	Specifica del linguaggio Java, pattern di errore	Java	OWL	Analisi statica degli errori