Tagonto

- Tagonto Project is an attempt of nearing two far worlds
- Tag based systems
  - Almost completely unstructured and semantically empty
- Ontologies
  - Strongly structured and semantically significant
- Taking the best of both worlds

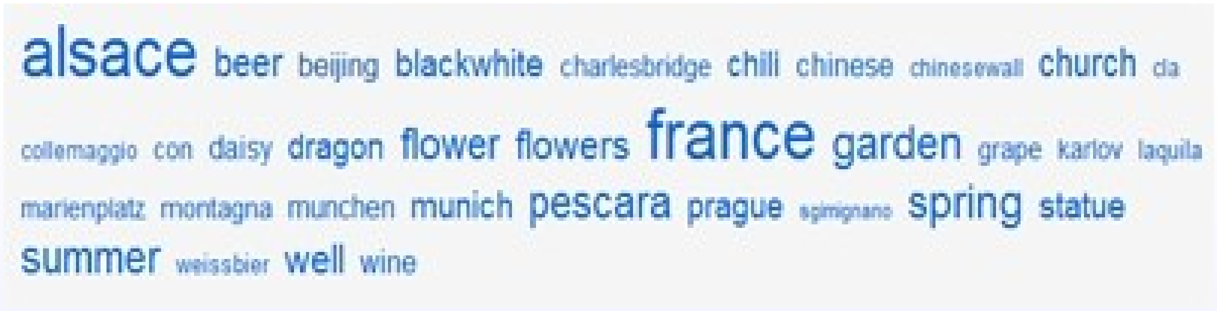
Tags and Folksonomies /1

- We all know what a TAG is...
- ...but what about a **Folksonomy**?
  - A little *more* than tagging: a user-based classification mechanism
- Sort of a *Community Powered taxonomy*
- The new age of TAGs, apart from personal use!

TAGs and Folksonomies /2

- **What are they good for?**
  - Relationships on unstructured data
  - Crossing the textual barrier
  - Merril Lynch: more than 85% of all business information is unstructured data
  - Tags lead to... more tags!

Serendipity!


alsace beer beijing blackwhite charlesbridge chili chinese chinesewall church cla
collemaggio con daisy dragon flower flowers france garden grape karlov laquila
marienplatz montagna munchen munich pescara prague sgimignano spring statue
summer weissbier well wine

A tag-cloud for a Flickr user

Ontologies

- What are they good for?
  - ...actually, a lot of stuff!
  - Narrowing down a little: what are they good for in the context of folksonomies?
  - Providing semantic and structural information where needed
- An Ontology-powered tag-based Search Engine

?!

Tagonto /2

- An *ontology-powered tag-based* **meta** search engine
- **What does Tagonto do?**
  - Associating tags to ontology concepts
    - Using algorithms or user defined mappings
  - Retrieving internet resources for a given TAG
  - *Surfing* the ontology and the associated tags

How does Tagonto work?

- Tagonto's architecture is divided into three main parts
- Tagonto LIB
  - The core of the mapping system
- Tagonto NET
  - The *meta* search engine
- Tagonto Interface
  - The mean to put them together

Tagonto NET

- A plugin based system for the interaction with tag based systems
- Written in PHP, can be used as a PHPLibrary or as a REST webservice – thus with any application and language
    - Why PHP? Because it's the *language of the net* and it's best suited to interact with Web2.0 systems
- Minimal setup and configuration
- 7 plugins are already available

Tagonto NET /2

- 2 main services offered
  - Retrieval of tagged resources given a tag
  - Retrieval of “Friends tags”, that is tag occurring often on the same resources with the given tag
- Plugins can be developed with ease
  - Copy-paste deployment
  - No strict requirement, just follow the contract and write the code

TagontoLib /1

- Stand-alone Java library
- Main task:
  - Given an ontology and a tag, finding the most significative matches between the tag and a concept of the ontology
- Developed as a part of the Tagonto project
  - Many of the non functional requirements (performance in particular) were imposed by its online-use

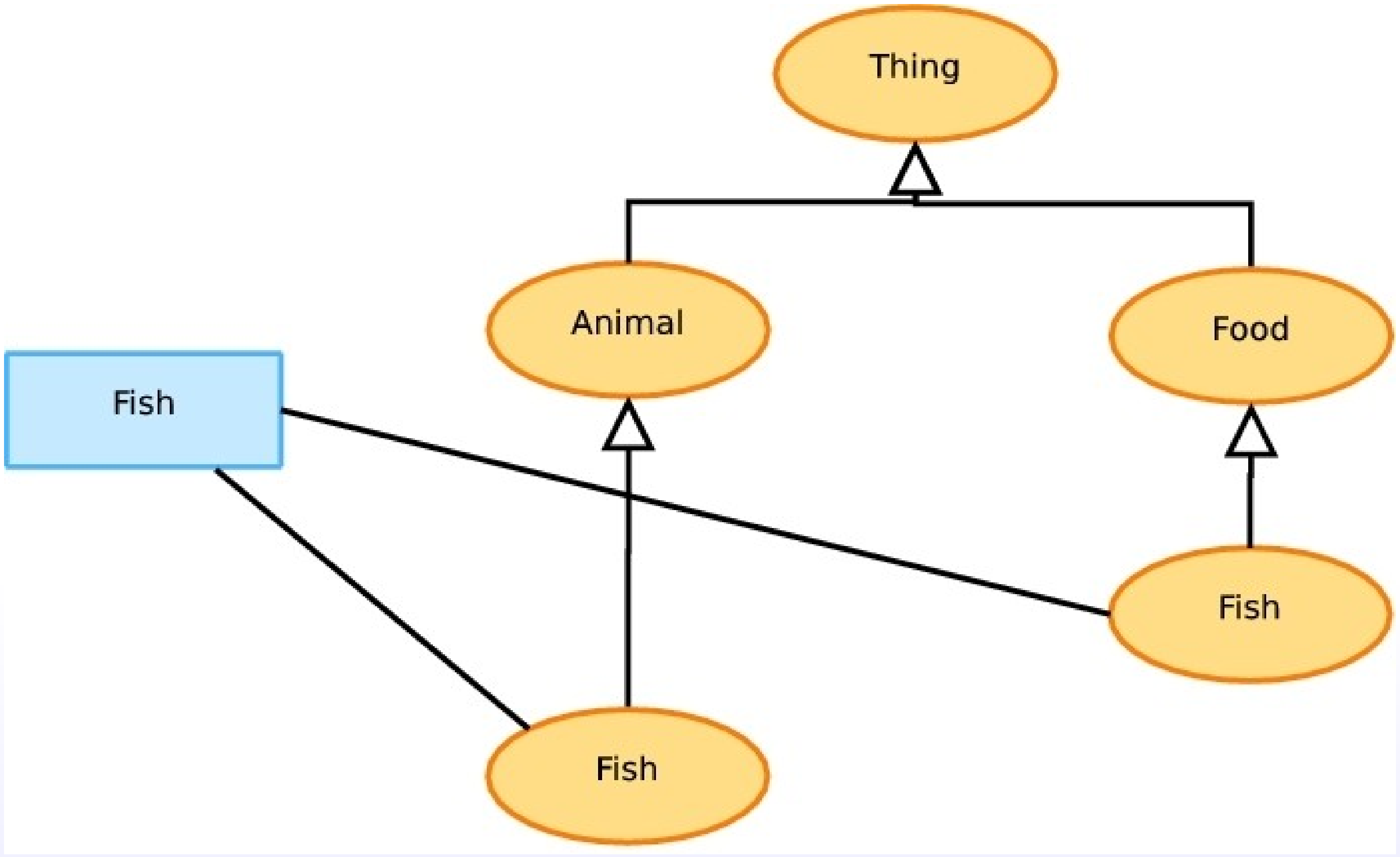
TagontoLib /2

- During development we took *inspiration* from the XSom project
- We couldn't use the Xsom project directly since it works between ontologies
  - Structural checks wouldn't have worked
  - Tagonto works between a flat structure (the tag) and a rich one (the ontology).

What is a Mapping /1

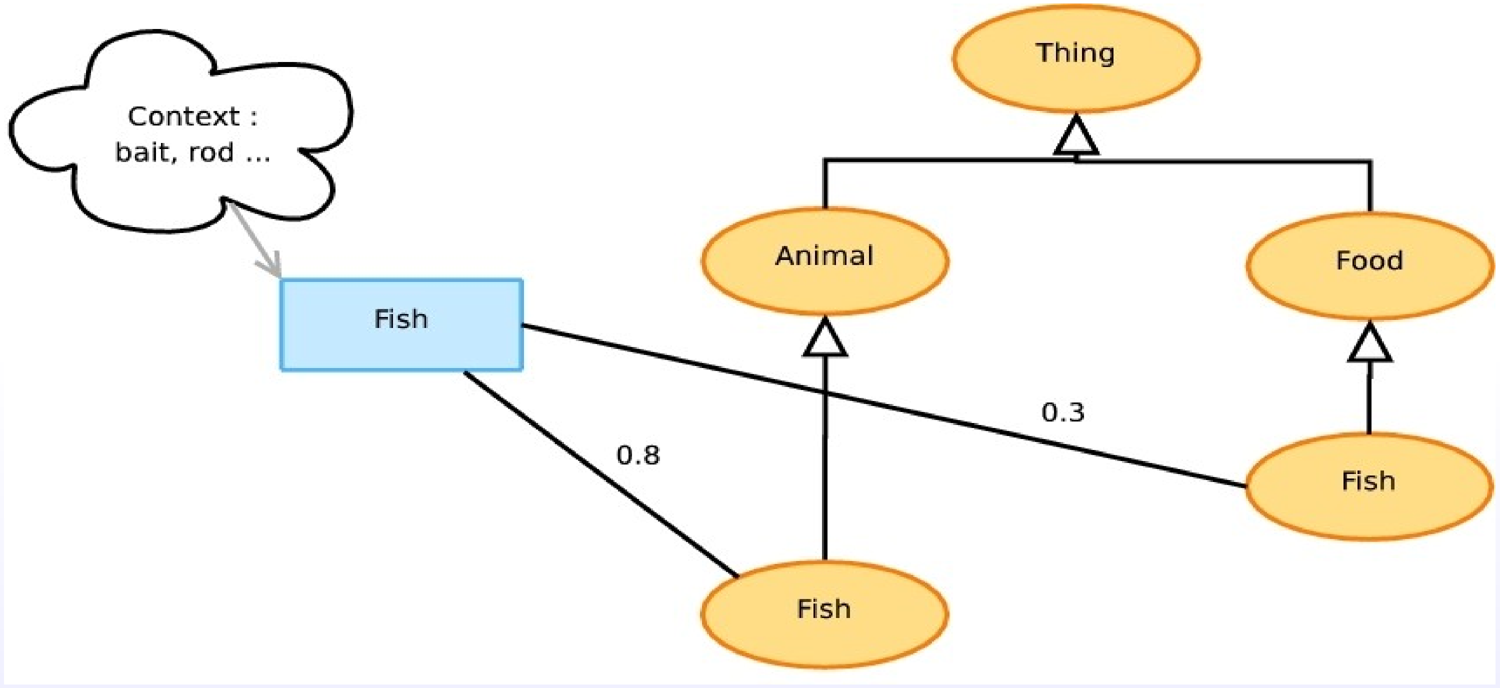
- It's a bidirectional relationship between
  - A vector of charachters (the tag)
  - A concept in a specific ontology
    - **TagontoLib considers in the analysis only named concepts**
      - Some of the mapping heuristics are based only on syntactic checks.
  - A significance value included in [0,1]

What is a Mapping 2



What is a Mapping 3

- How do we distinguish between differente mappings?
  - We can use the significance value



Mapping Heuristics : classification 1

- Mappings are generated using some heuristics
  - They can be classified in bidimensional space
    - First Dimension (Task)
      - Generative Heuristics : generating new mappings
      - Choosing Heuristics : modifying significance of pre existent mappings
    - Second Dimension (Informations Used)
      - Syntactic Heuristics : only syntactic informations
      - Semantic Heuristics : both syntactic and semantic informations

Mapping Heuristics : classification 2

	Syntactic	Semantic
Generative	Exact Match Levenshtein Match Jaccard Match Google Noise Match	Wordnet Similarity
Choosing	Max Chooser Threshold Chooser	Links Chooser Google Chooser Friends Chooser

Mapping Strategies 1

- Heuristics are put together using mapping strategies
  - They define
    - which heuristics are used
    - which is the order they are used in
    - how results from different heuristics are merged
- In other words they define the complete mapping algorithm

Mapping Strategies 2

- TagontoLib provides 3 mapping strategies by default :
  - Greedy Strategy :
    - Uses only syntactic checks and Wordnet Similarity
  - Instance Strategy
    - Uses only syntactic checks against concept instances
  - Standard Strategy
    - Uses both the greedy and the instace strategy and semantic heuristics to disambiguate mappings

Mapping Heuristics : Exact, Levenstein, Jaccard

- These heuristics generates new mapping using string comparison metrics to calculate significance
  - Exact
    - The usual string comparison avaible in java
  - Levenshtein
    - Levenshtein string similarity
  - Jaccard
    - Jaccard string similarity

Mapping Heuristics : Google Noise

- Analyzes the tag for misspellings using google
  - A search for the specified tag is issued and the result page is analyzed in search for a *maybe did you mean* suggestion
  - A noisyness measure is calculated calculating string similarity between the original tag and the suggested tag
  - If the similarity exceeds a threshold specified by the mapping strategy, new mappings are generated using a greedy strategy (to avoid performance problems).

Mapping Heuristics : Wordnet Similarity

- This heuristics uses a component taken directly from the Xsom project
  - For every named concept in the ontology
    - The tag and the concept name are tokenized if possible
    - Tokens are searched in the wordnet dictionary and *related words* (hyponims, hypernims and synonims) are extracted
    - Tag's *related words* and concept's *related words* are compared using string metrics to calculate similarity of mappings.

Mapping Heuristics : Max and Threshold Chooser

- These heurisics modify a list of mappings analyzing only significance.
- Their main usage is aggregating results obtained from other heuristics
  - Max chooser
    - First the maximum mapping significance in the list is found, then the list is modified removing all the mappings having lower significance
  - Threshold chooser
    - The list is modified removing all the mappings having significance lower that the threshold

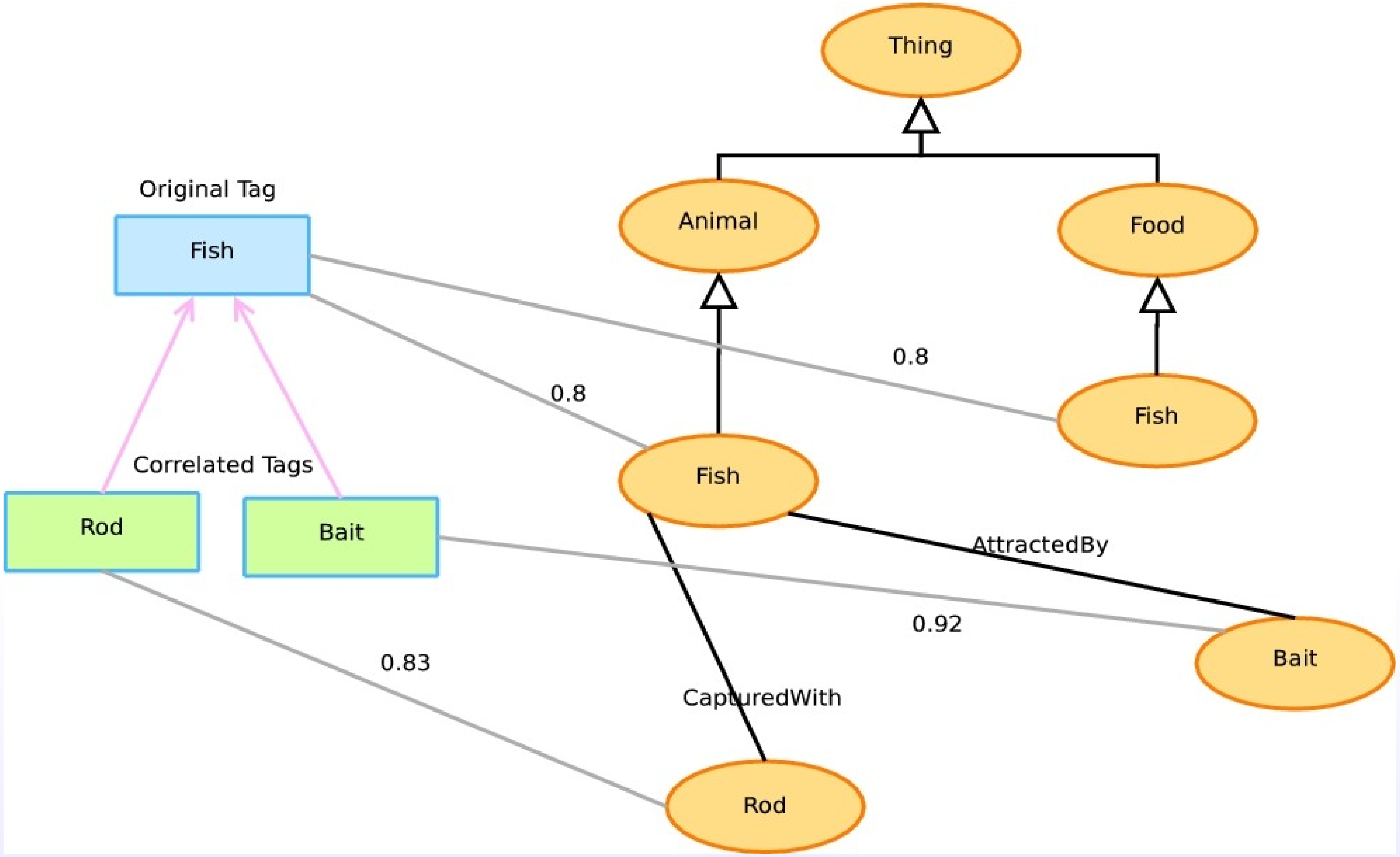
Mapping Heuristics : Link Chooser 1

- This heuristics uses the tag context and information that can be inferred from the ontology to modify the significance of pre-existing mappings.
- Basically, its task is trying to disambiguate different mappings for the same tag.

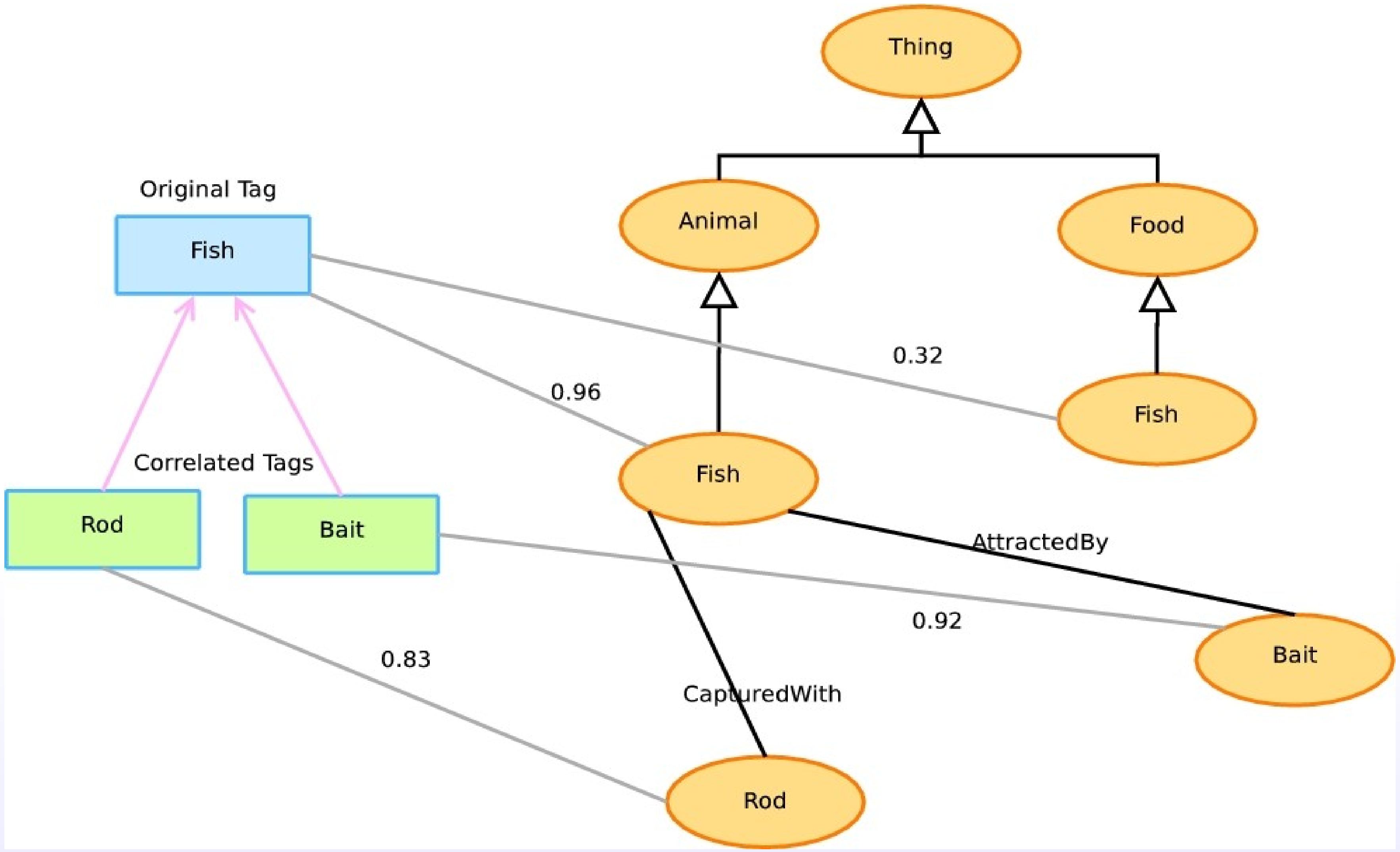
Mapping Heuristics : Link Chooser 2

- Theoretical backgroud
  - If a mapping is correct (i.e. we match the tag against it the correct concept) then the concept should be connected to concepts mapping the tag context (i.e. there should exist ontology properties having as range the *original* concept and as domain the *correlated* concepts).

Mapping Heuristics : Link Chooser 3



Mapping Heuristics : Link Chooser 4



Mapping Heuristics : Link Chooser 5

- How does it works ?
  - First we need to generate mappings for the original tag an retrieve tags specifying the context (i.e. *correlated* tags).
  - Then we map *correlated* tags using a Greedy strategy (to avoid performace problems)
  - Finally we increment the significance of the original mappins by an amount depending on the linkness of the original concept with *correlated* concepts.

Mapping Heuristics : Google Chooser

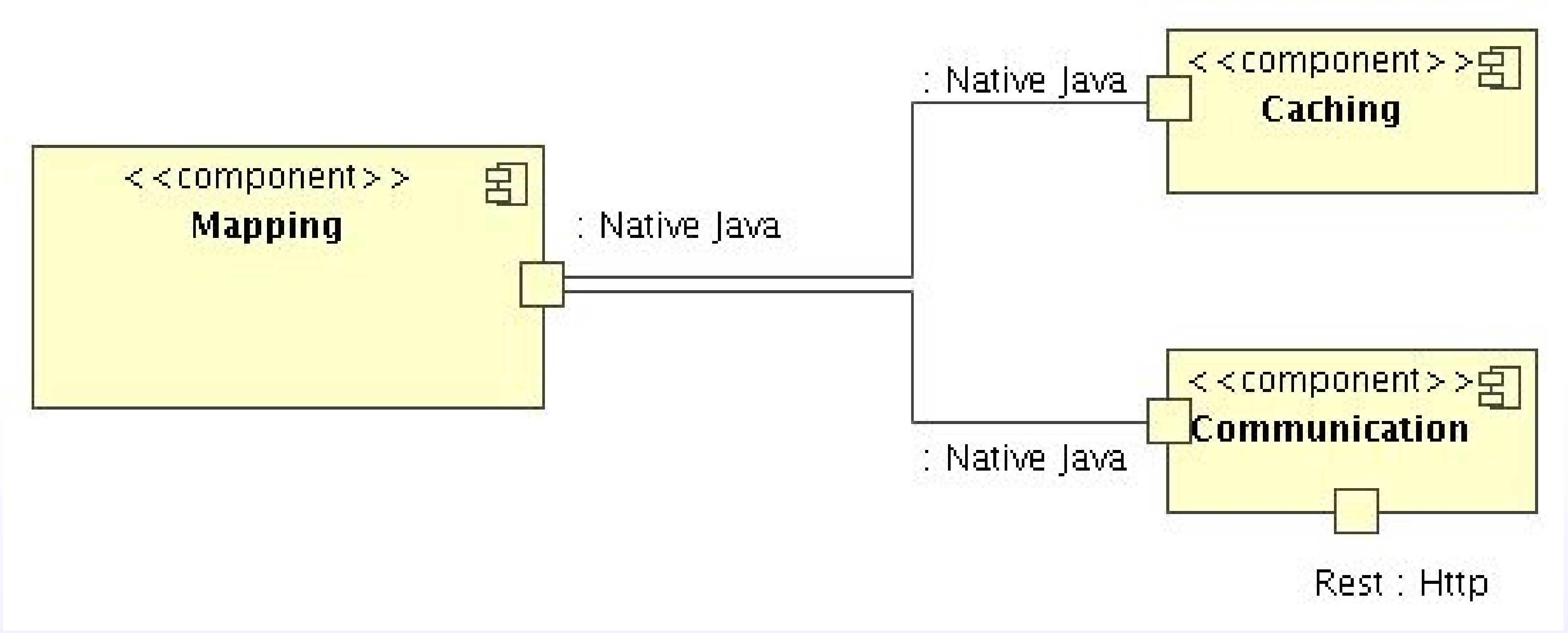
- This heuristics uses Google to retrieve the list of correlated words and then uses that list to invoke the Link Chooser
  - A search for the tag is issued and the first N results (html pages) returned by google are analyzed
    - If the meta tag keyword is present, the list of keyword is used as the list of correlated tags
    - Otherwise html tags are stripped off and correlated tags are extracted using text mining techniques (stemming, vector of counts...)

Mapping Heuristics : Friends Chooser

- This heuristics uses a TagontoNet service to retrieve tags correlated to the original tag, then uses this list to invoke the List Chooser.

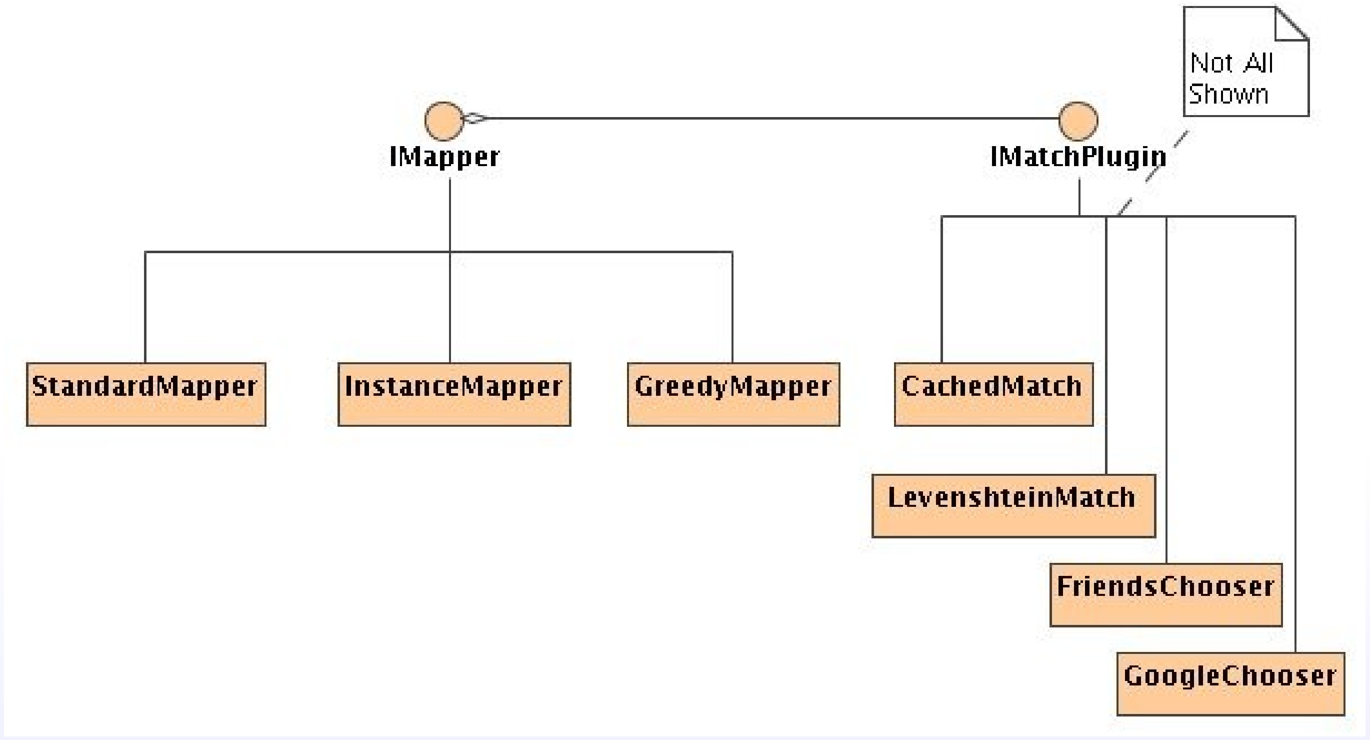
TagontoLib : Architecture

- TagontoLib uses 3 main component to accomplish its task



The mapping Component 1

- Implements the mapping heuristics and strategies exposed before



The Mapping Component : Greedy Mapper

- The Greedy Mapper has been developed to efficiently obtain a list of mappings for a tag
  - Extensive use of cached informations
  - Only syntactic heuristics and Wordnet Similarity
  - Mapping may not be precise, especially in presence of similar tags and concept names.

The Mapping Component : Instance Strategy

- The Instance Mapper has been developed to include in the analysis also instances of concept.
  - This strategy does only syntactic checks between the name of an individual of a concept and the tag.
  - If matches with high significance are found, new mappings of the tag on the class of the individual are generated

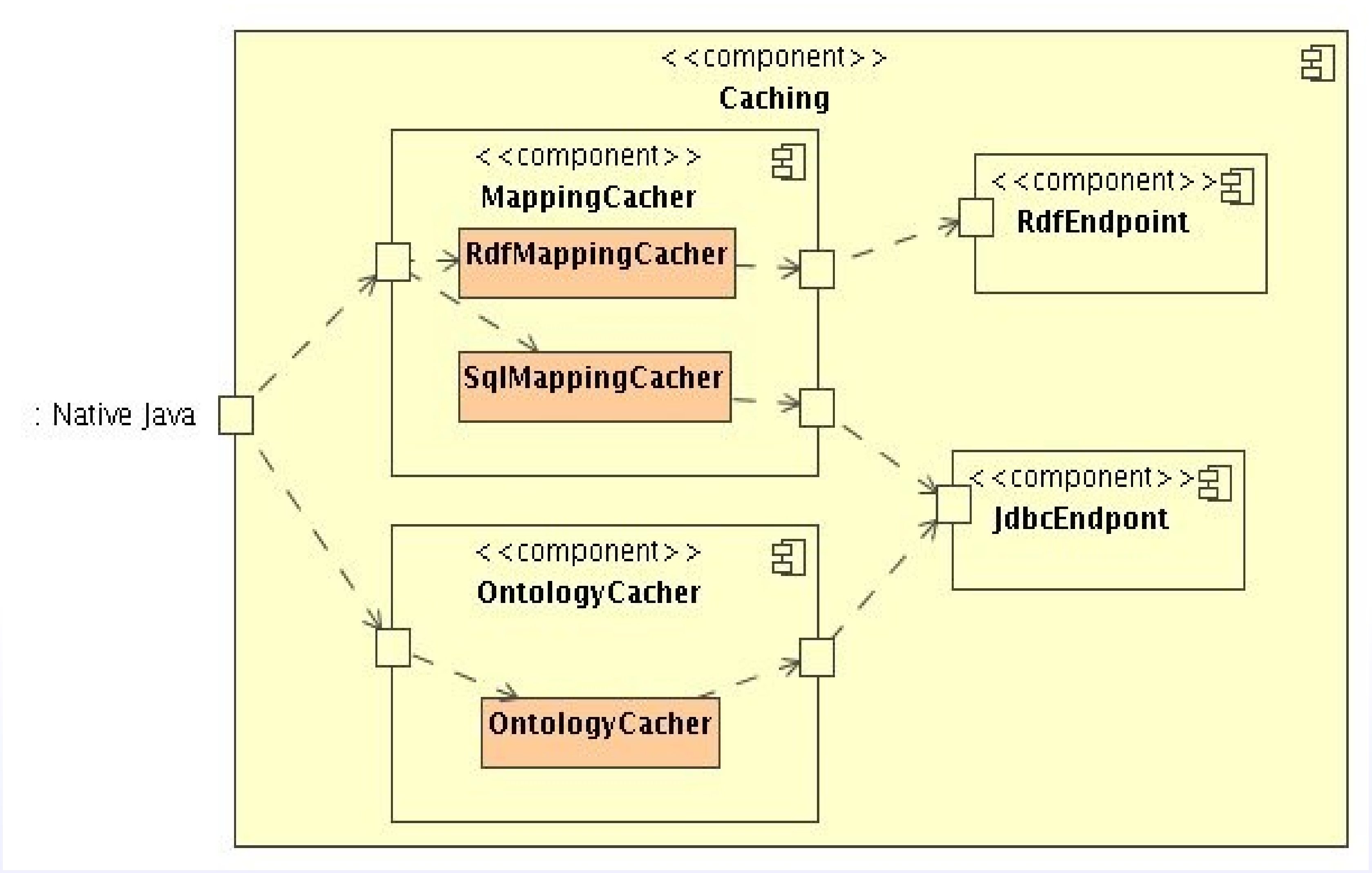
The Mapping Component : Standard Mapper

- The Standard Mapper implements the most complete strategy avaible.
  - It uses the Greedy Mapper to obtain an intial set of mappings
  - Then uses the Instance Mapper to enlarge the initial set
  - Finally uses the Google Chooser and the Friends Chooser to accomplish disambiguation of mappings.

The Caching Component 1

- This component was developed to satisfy the strict performance requirements imposed by an online use of TagontoLib.
- Since plugins implementing heuristics do an extensive use of knowledge inferred from the ontology and reasoning can be pretty slow, this component caches also knowledge inferred from the ontology with the use of a reasoner.

The Caching Component 2



The Communication Component

- This component was developed to provide a facility to communicate with the lib without using java.
- It implements an HTTP REST interface (running on an embedded lightweight web server) to invoke the main functionalities of the library.