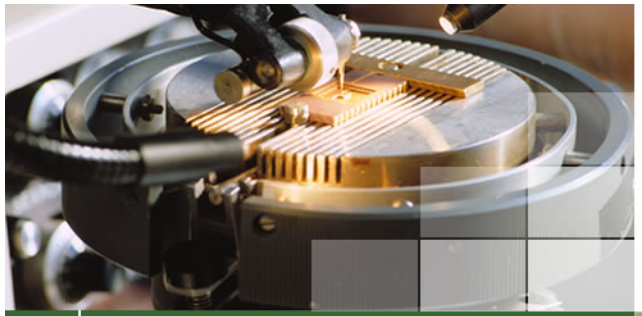


POLITECNICO DI MILANO

Dipartimento di
Elettronica e Informazione

Search Computing:

Semantic Framework and Service Registration

Silvia Quarteroni

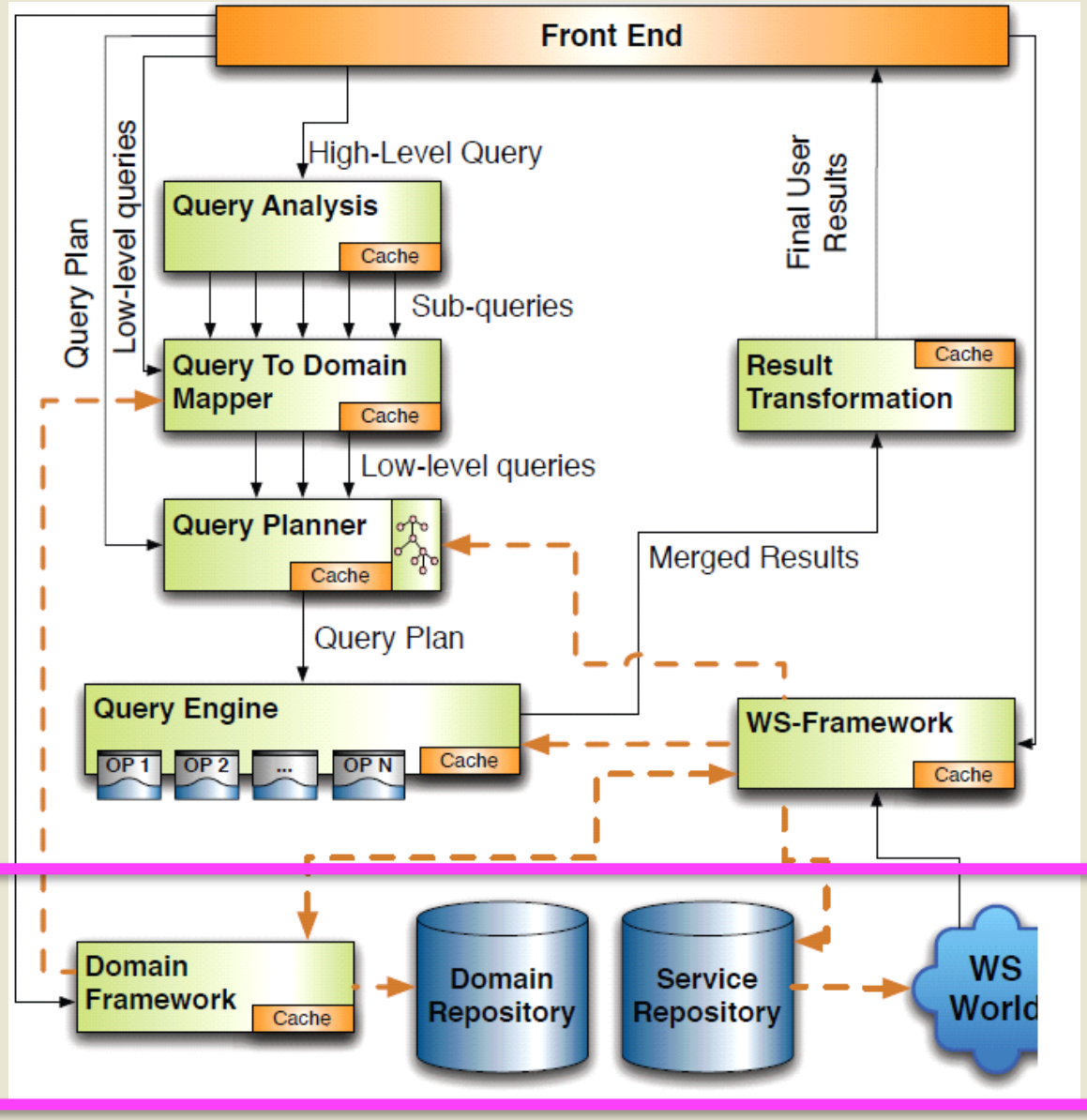
quarteroni@elet.polimi.it

Joint work with: D. Braga, M. Brambilla, S. Ceri, L. Tettamanti

Outline

- Part 1: Semantic Framework
  - How service semantics are represented
  - How to exploit service semantics to group services together or find service combinations
- Part 2: Service Registration
  - How we get to the semantic framework
  - A pragmatic approach to the mapping of service semantics into a common domain representation

Where we are in the SeCo architecture



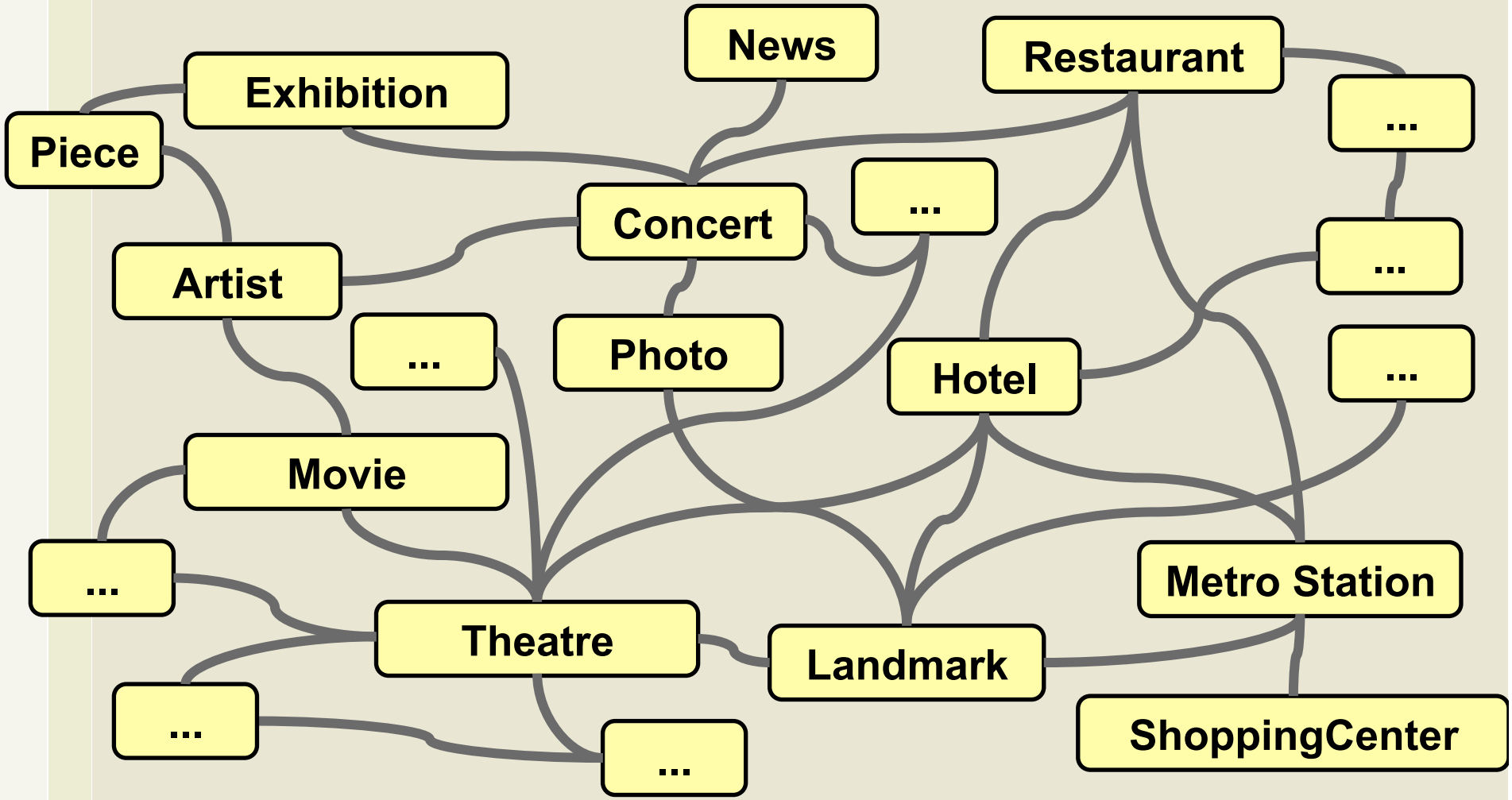
Part 1

SEMANTIC FRAMEWORK

Motivations: Data Service Integration

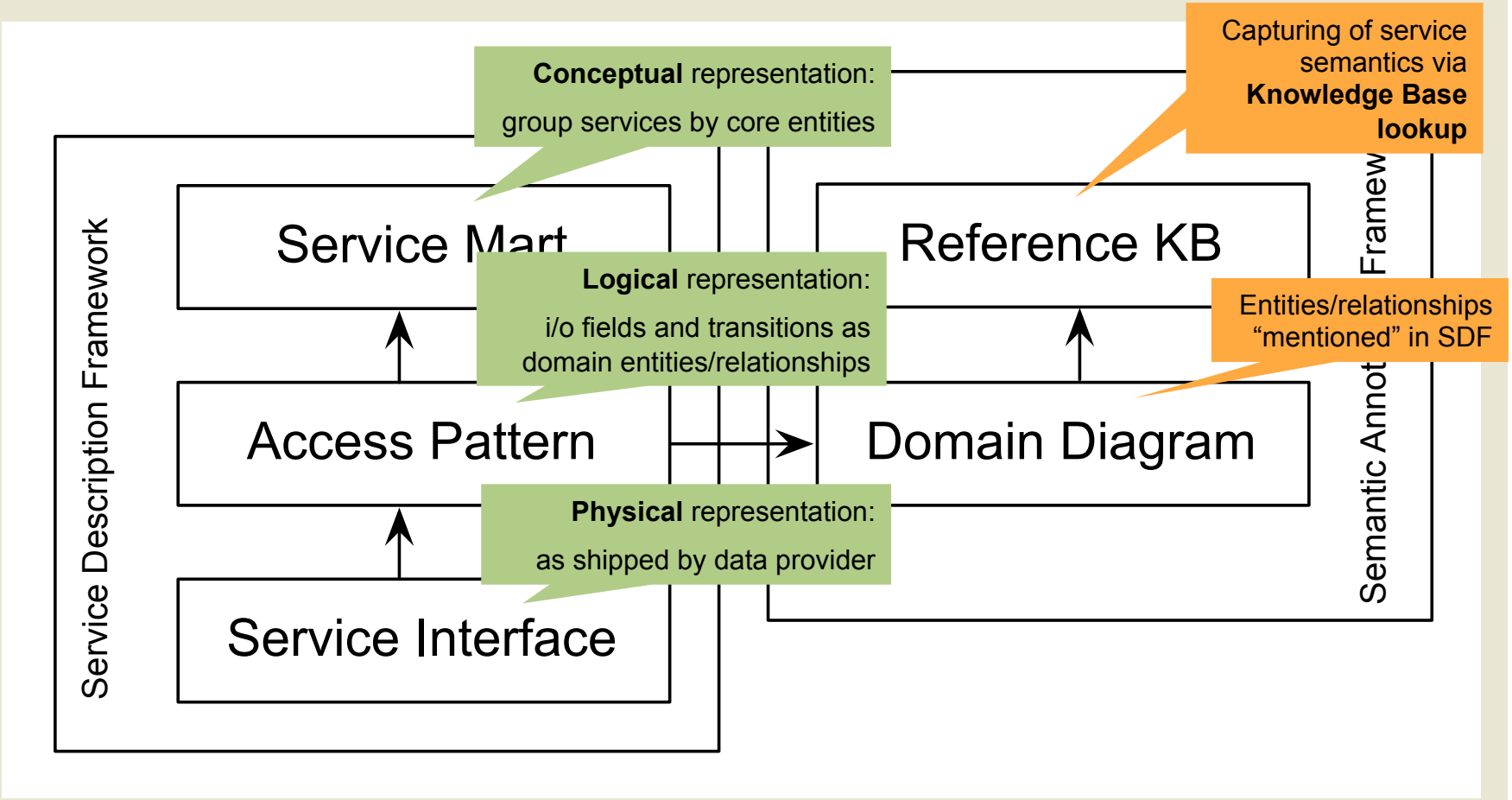
- Problem: there is no universal reference for how to express what data services do (i.e. their semantics)
  - Two services annotated with different terminology by their providers may return similar data but appear as very different
  - A “vertical” search engine per data service might not be the optimal solution!
- Idea: provide a “Semantic Framework” where concepts of the real world are mapped to entities and connected by relationships
  - Data service semantics can then be expressed in terms of such entities and relationships

Rationale of Service Representation
High-Level View: Domain Diagram with connections



Service Framework in SeCo

- A Service Description Framework coupled with a Semantic Annotation Framework



Service Description Framework: Service Interfaces

- **Service Interfaces** (SIs) are "physical" representations of data services
  - Directly come from provider with either very light or no wrapping
- Service Interface specification:
  - Name
  - A set of *input* and *output* fields
  - Additional parameters: selectors (when to select this SI), template calls to data service, whether data service returns ranked results, etc.
- Examples:
  - SI1: YQL_Film [iYear, oActor(first,last), oCategory, oTitle]
  - SI2: Google_Theaters [iLocation, oLocation, oTitle, oShowTimes]
  - SI3: imdb_movies [iYear, oTitle, oActors(firstname,lastname), oGenre]

Portion of a Service Interface description (JSON)


```
"__type__" : "RDBServiceInterface",
"__id__" : "imdb_movies",
"chunked" : "true",
"description" : "This service provides access to a materialized version of the IMDB database by genre and year.",
"inputAttributes" : [ {
  "__id__" : "iGenre",
  "dataType" : "STRING",
  "name" : "iGenre"
}, {
  "__id__" : "iYear",
  "dataType" : "INTEGER",
  "name" : "iYear"
} ],
"modifiers" : {
  "cacheEnabled" : "true",
  "cacheTimeToLive" : "3600000",
  "chunkSize" : "100",
  "scoreFunction" : {
    "__type__" : "WeightedScoreFunction",
    "weights" : [ {
      "best" : "10.0",
      "fieldName" : "score",
      "weight" : "1.0",
      "worst" : "0.0"
    } ]
  }
},
"name" : "IMDB Movies by Genre",
"outputSignature" : [ {
  "__type__" : "ServiceInterfaceAttribute",
  "__id__" : "movieId",
  "dataType" : "INTEGER",
  "name" : "movieId"
}, {
  "__type__" : "ServiceInterfaceAttribute",
  "__id__" : "title",
  "dataType" : "STRING",
  "name" : "title"
}, {
```

Service Description Framework: Access Patterns

- **Access Patterns** (APs) express the I/O fields of SIs in terms of attributes of domain entities
- An AP I/O field is characterized by
  - A semantic type, i.e. the domain item it refers to (**Location.city**)
  - A label representing its *role* in the AP (*departureCity*, *arrivalCity*)
- An AP is fully specified by:
  - A set of *input* and *output* fields
  - A *focus*: the main domain entity mentioned in its output
  - A functional *name* referring to its focus and I/O fields
- Examples:
  - AP1: MovieActorByYear (fits SI1 and SI3) > focus: **Movie**
  - AP2: TheaterByMovieAndLocation (fits SI2) > focus: **Theater**

Portion of an Access Pattern description (JSON)

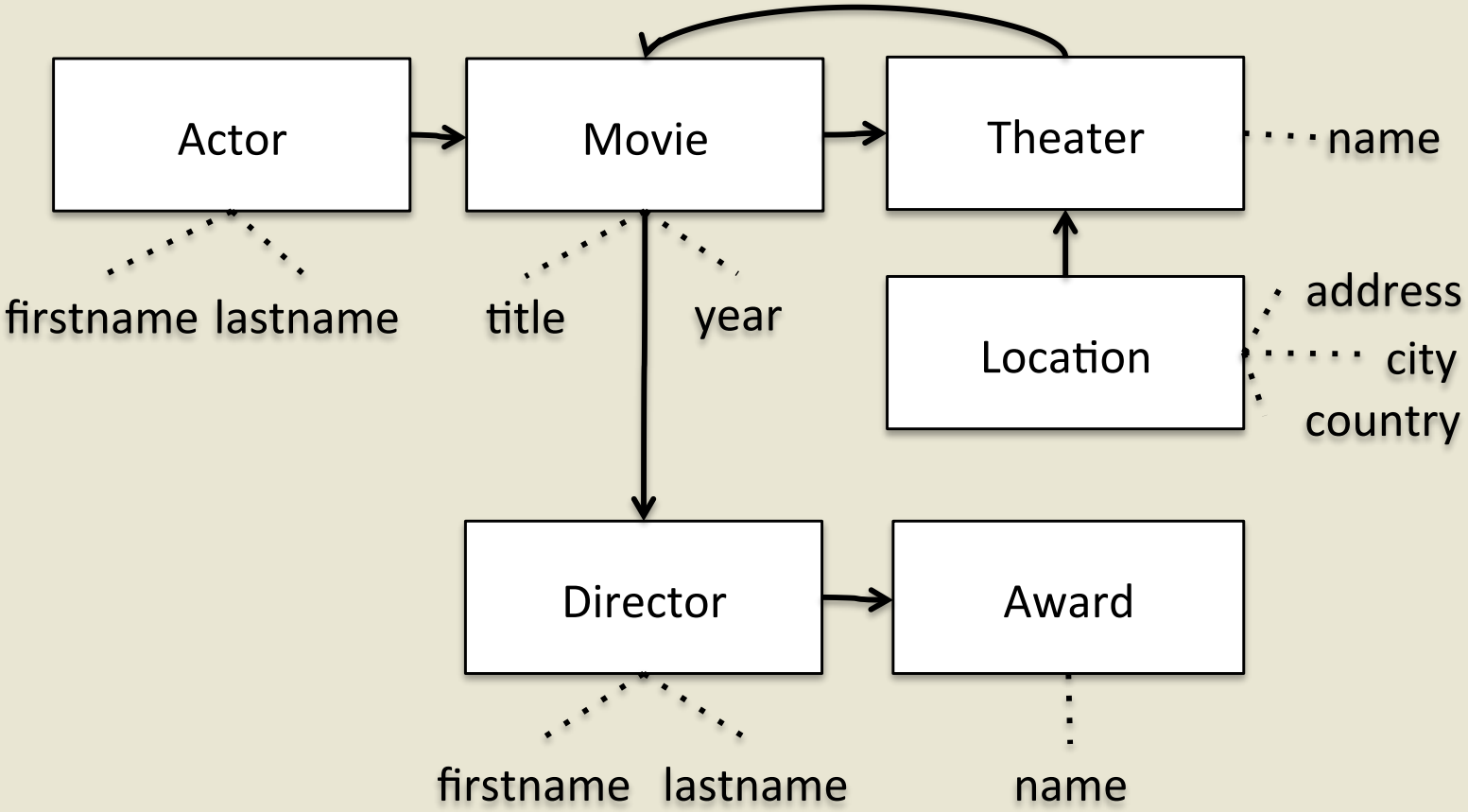

```
"__type__" : "AccessPattern",
"__id__" : "GET movie BY movie.title",
"focusEntityName" : "movie",
"inputAttributes" : [ {
  "__id__" : "title",
  "DDMapping" : {
    "domainAttributeName" : "title",
    "domainEntityName" : "movie",
    "path" : "movie.title"
  },
  "dataType" : "STRING",
  "description" : "",
  "name" : "title",
  "role" : "iTitle",
  "selector" : "false"
} ],
"name" : "GET movie BY movie.title",
"outputSignature" : [ {
  "__type__" : "PatternAttribute",
  "__id__" : "movie",
  "DDMapping" : {
    "domainAttributeName" : "movie",
    "domainEntityName" : "movie",
    "path" : "movie.movie"
  },
  "dataType" : "INTEGER",
  "description" : "",
  "name" : "movie",
  "role" : "movieId",
  "selector" : "false"
}, {
  "__type__" : "PatternAttribute",
  "__id__" : "title",
  "DDMapping" : {
    "domainAttributeName" : "title",
    "domainEntityName" : "movie",
    "path" : "movie.title"
  },
```

Service Description Framework: Service Marts

- **Service Marts** (SMs) generalize over all APs sharing the same focus
- Specified by
  - A domain entity, e.g. **Movie**
  - The set of all APs whose focus is the entity (e.g. AP1)
- E.g. all APs whose focus is the **Movie** entity fall under the ***Movie*** Service Mart
  - Movie by actor
  - Movie by theater location
  - Movie by year
  - …

Semantic Annotation Framework: Domain Diagram

- A **Domain Diagram** (DD) represents the entities, attributes and relationships defined by the services
  - A simple Entity-Relationship diagram



Semantic Annotation Framework: Reference Knowledge Base

- A reference **Knowledge Base** (KB) represents entities, relationships and instances in an arbitrarily large domain
  - Many KBs exist on the Web (YAGO, datasets from LinkedData, etc.)
- The DD refers to one or more KBs in such a way that
  - Each DD *entity* (e.g. **Movie**) corresponds to exactly one KB concept
  - Each *attribute* of a DD entity (e.g. ***evaluation***) corresponds to exactly one KB concept
  - However, DD *relationships* need not have a counterpart in the KB and viceversa

YAGO as a reference Knowledge Base

- KB in SeCo: YAGO (mpi-inf.mpg.de/**yago**-naga/**yago**/)
  - A huge repository of over 3 million *facts*, e.g. “Albert_Einstein is-a scientist”
  - Obtained semi-automatically from the integration of Wikipedia and the WordNet lexical database (wordnet.princeton.edu)

- SeCo – YAGO mapping:
  1. DD entities are grounded with their YAGO counterparts
     - SeCo’s **Hotel, evaluation** correspond to YAGO’s **Hotel**, **Evaluation**
  2. This gives meaning to shared relationships
     - **Director** *hasWon* **Award, Suburb** *isPartOf* **City**
  3. YAGO is useful to learn about concept instances
     - “inn” is a synonym of **Hostel** (YAGO *means* relationship)
     - “Paris”, “Rome” are instances of **City** (YAGO *type* relationship)
- Benefits of SeCo – YAGO mapping:
  1. No need to invent yet another ontology, we can reuse YAGO terms
  2. Many information extraction resources are expressed in a subset of YAGO terms; this is beneficial if we want to interpret natural language queries over services

YAGO as a reference Knowledge Base

Browse YAGO2

Entity: [ ] case insensitive

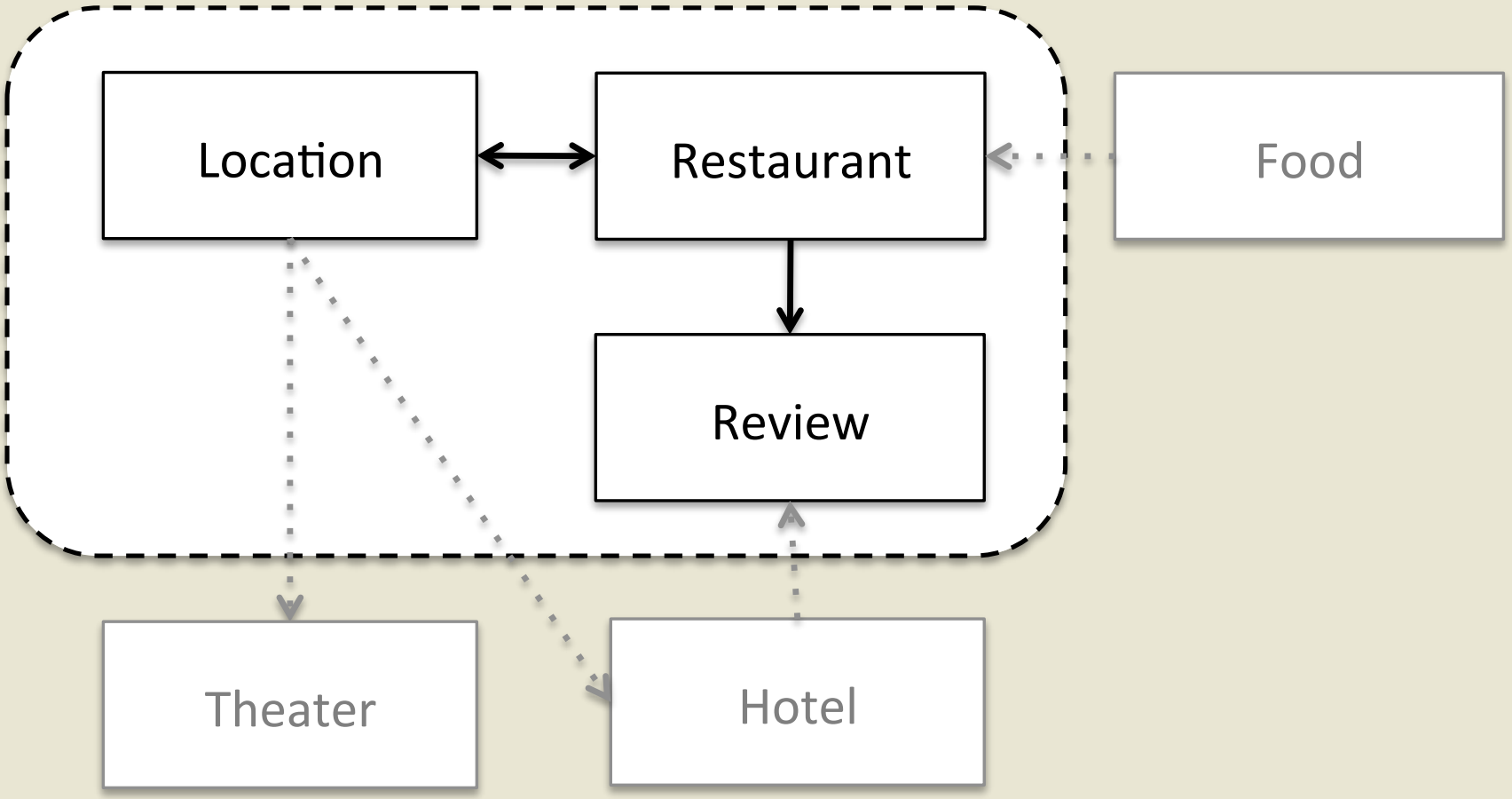
wordnet_city_108524735

Show transitive facts

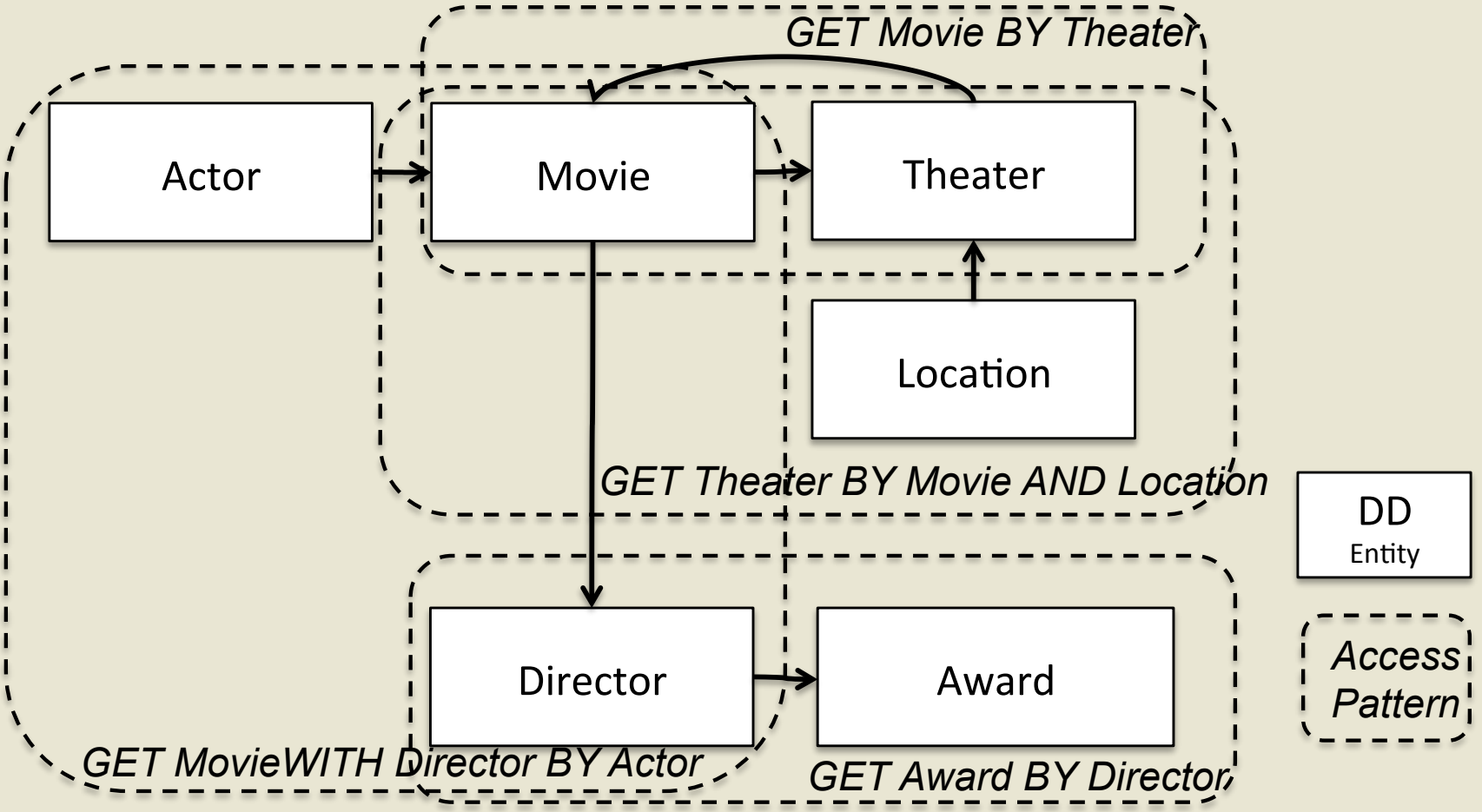
← city ← metropolis ← urban center	means	
← Achaemenid cities ← Aegean Sea port cities and towns in Turkey ← Akkadian cities ← Aksumite cities ← American cities in fiction ← Amorite cities ← Ancient Assyrian cities ← Ancient Chinese cities ← Ancient cities ← Ancient cities in Cyprus ← Ancient cities in Italy ← Ancient cities in Northern Cyprus ← Ancient cities in Serbia ← Ancient Greek cities ← Ancient Greek cities in Italy ← Ancient Indian cities ← Aramaean cities ← Australian capital cities ← Black Sea port cities and towns in Turkey ← Canaanite cities ...	subclassOf	
← 1 SGM ← 3 STR ← Aabenraa		

hasSynsetId	108524735 →
isCalled	agglomération urbaine → aglomeración urbana → aglomeração urbana → alu → banwa → baþkent → başkent → belt → borg → býur → burh → capital → capital → capital → capitale → capitale → capitalã → cathair → ceity → chi'nashtalichi → ...
	administrative district → district → entity → geographical area → location →

Span of an Access Pattern over the Domain Diagram



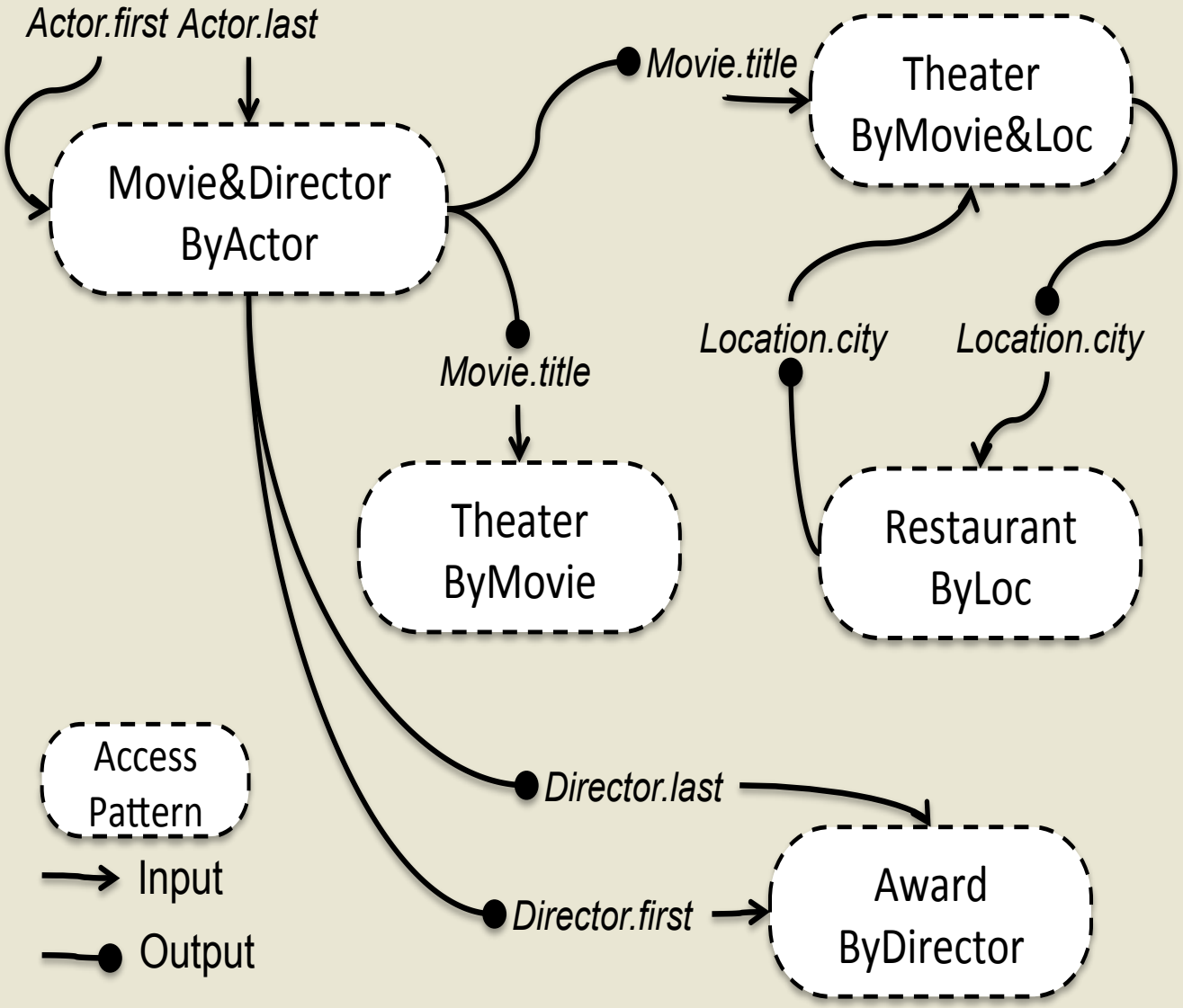
Access Patterns spanning over the Domain Diagram



Access Pattern Connections

- Access Patterns having DD entities in common may be connected
  - Underlying Service Interfaces may be combined!
- Connection types:
  - *Serial*: AP2's input is a subset of AP1's output
  - *Parallel*: AP1's and AP2's output are equivalent (same semantic types)
  - *Generalized*: AP3's input is a subset of the union of AP1 and AP2's output

Access Pattern Connections



Part 2

SERVICE REGISTRATION

Definition

- Service Registration is the operation by which a *data service* becomes known to the *semantic framework*
  - It is mapped to a service interface which in turn refers to an access pattern within a service mart
- Registered services can then be queried; possibly, queries combining several services can be handled
  - Further details about actors that played in a movie we'll see: Movie service *then* actor service
  - Finding a lodging for a conference & how to get there: Conference service *then* Hotel service *then* Bus service
  - Finding a hotel close to our favorite restaurant and the theater: Restaurant service *and* Theater service *then* Hotel service

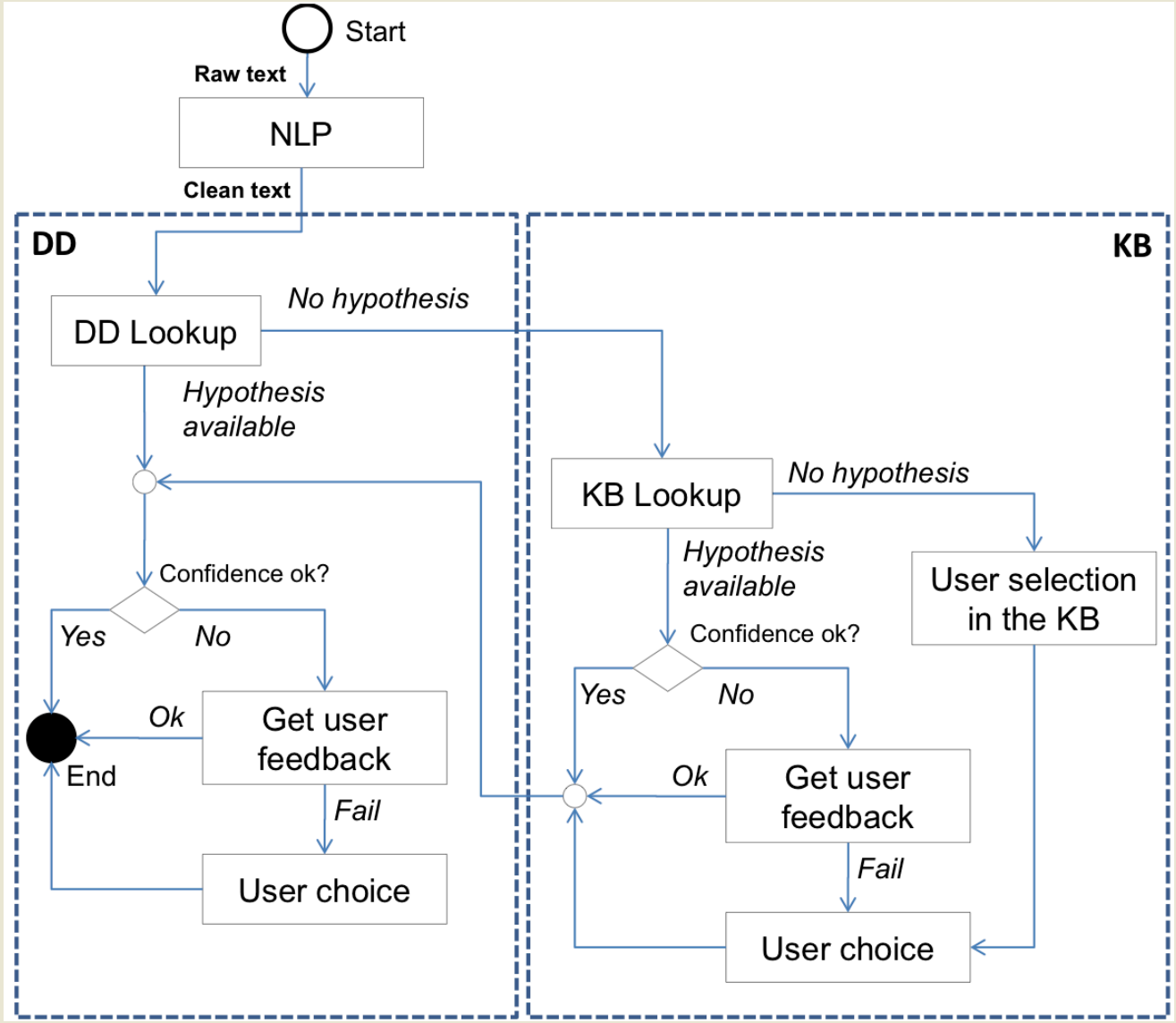
A bottom-up approach to service integration

- Challenge: data service integration
- Lack of semantic description: how do we get to the Semantic Framework?
- During registration, we infer the semantics **bottom-up**, starting from data services and producing both the Service Description Framework and the Semantic Annotation Framework
  - Result: Service Interfaces, Access Patterns, Service Marts + the Domain Diagram common to all services
- The registration process is aided by:
  - Lookup on the reference knowledge base (KB)
  - Minimal supervision from a human expert to validate domain to KB mapping

Service registration

- Input: Service Interface
- Each SI i/o field *f* is mapped into one (or more) AP i/o field(s), described in terms of a DD item
  - If DD contains no suitable item to represent *f*, a suitable KB item K is found & new DD item is created after it > the DD is progressively populated from scratch!
  - Heuristics: expert intervention in choice of KB items/ disambiguation amongst candidate DD items (e.g. “film” = *plastic foil* or *movie*?)
  - Light natural language preprocessing: field name is converted into a set of possible “canonical” names (e.g. *mTitle* > *title*)
- Output: new Access Pattern
  - If AP is equivalent to a previous AP in the Service Description Framework, merge into a single AP, unifying service interfaces
  - APs are clustered by focus into Service Marts

Mapping a SI i/o field to a DD item



Registration experiments

- Is this method feasible? Is term unification useful?
- To find out, we registered service interfaces deriving from 3 repositories in the SeCo "portfolio"
  - Rep A: 4 services, B: 19 services, C: 40 services
- Size of DD remains compact

	Rep. A	Rep. B	Rep. C
# service interfaces	4	19	40
# service interface terms	38	239	495
# resulting DD objects	36	136	188
# resulting DD entities	4	14	21
# resulting access patterns	4	18	40
# resulting service marts	4	12	19

Registration experiments

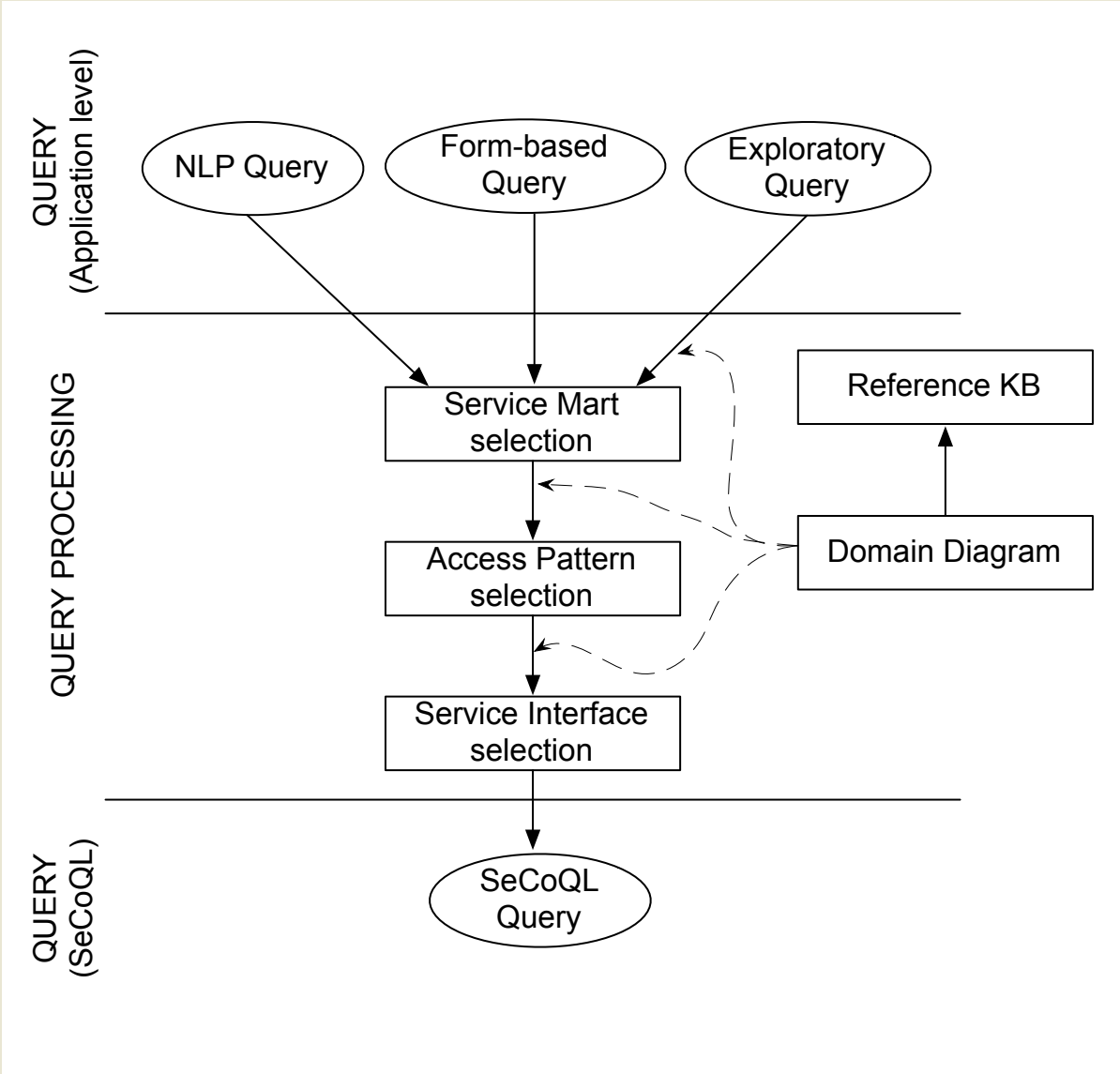
- Further measurements:
  - KB recall: NLP methods allow to find automatic KB mapping for about 90% of service interface parameters

	Rep. A	Rep. B	Rep. C
NLP Recall	92%	88%	90%

  - Impact of human supervision with different heuristics: more mapping precision but more time required, too!

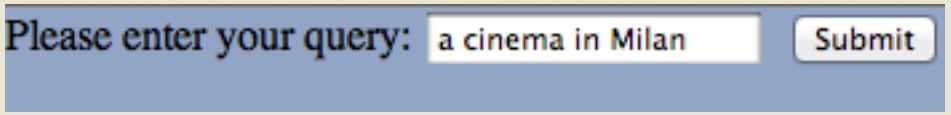
Heuristic		Precision		
DD	KB	Rep. A	Rep. B	Rep. C
IMP	*IMP*	76%	74%	71%
SEL	*IMP*	90%	98%	87%
SEL	*COR*	100%	99%	97%

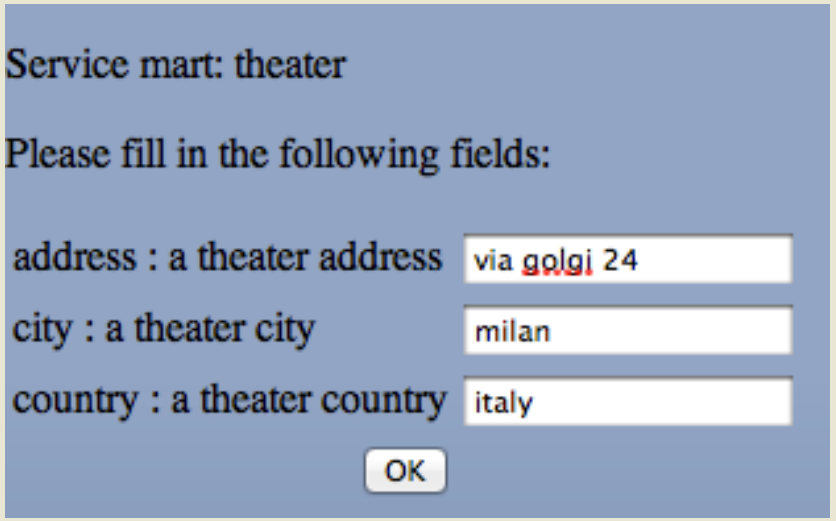
A Look at Query Analysis



Towards Natural Language Queries over SeCo services

- Step 1: understand query class, i.e. relevant service mart
- Step 2: identify relevant access pattern, then fill in I/O fields
- Step 3: identify relevant Service Interface, then create a logical query





SI attribute-value pairs: {iAddress=via golgi 24, iCity=milan, iCountry=italy}

```
    "__type__" : "CustomServiceInterface",
    "__id__" : "google_theatre_by_address",
    "accessPatternId" : "GET theater WITH location AND movie BY
theater.address AND theater.city AND theater.country",
    "chunked" : "false",
    "className" : "org.seco.wrapper.googlemovies.ServiceManager",
```