UNIVERSITÄT LEIPZIG

Matching Large Schemas with COMA++

Erhard Rahm, Hong-Hai Do
David Aumueller, Sabine Massmann
http://dbs.uni-leipzig.de

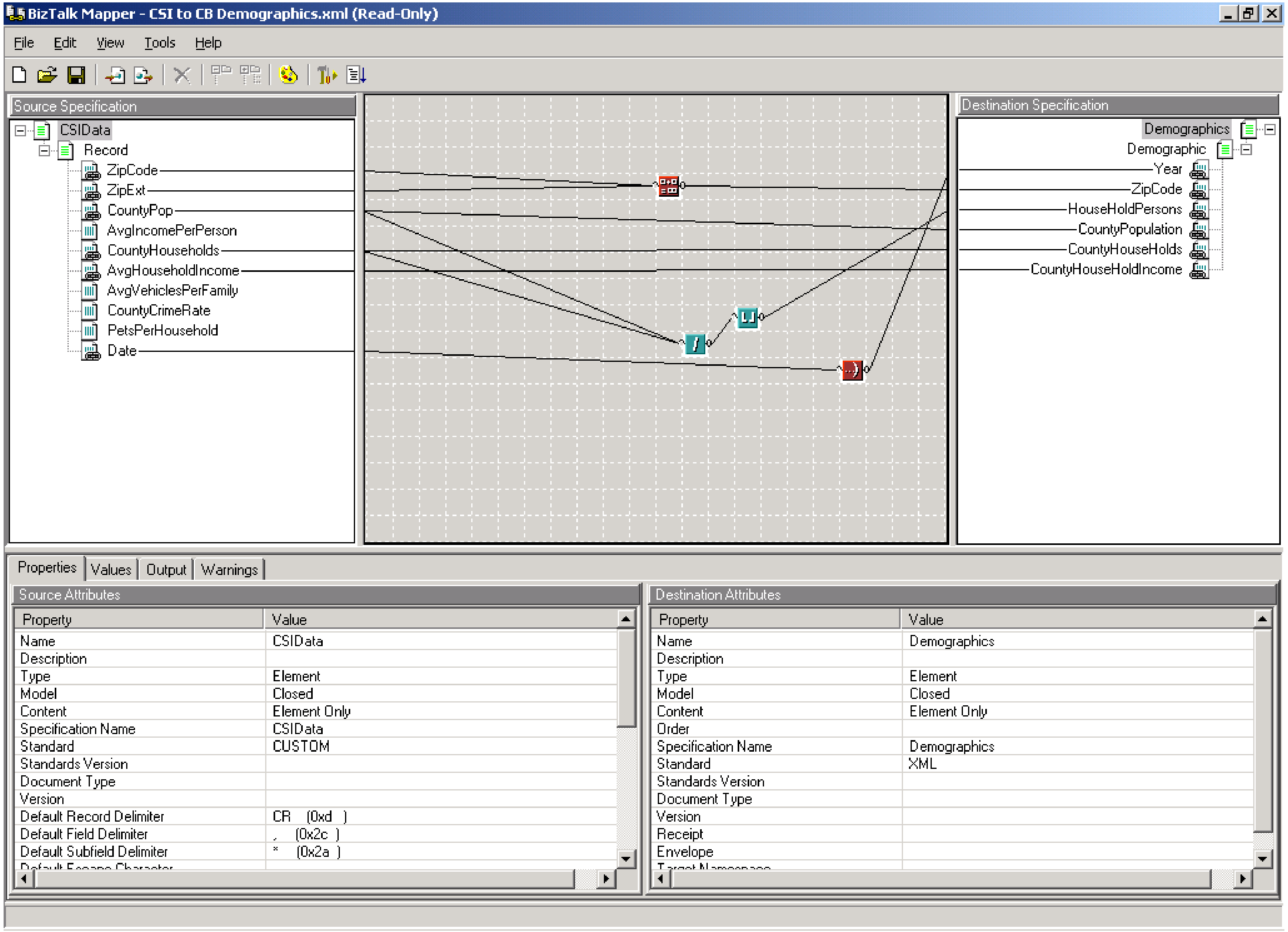
Database Group U Leipzig (Feb. 2005)

- Projects: Metadata Mgmt (COMA++), Data Integration (GeWare, GenMapper, *iFuice*), Adaptive web recommendations (AWESOME), E-Learning

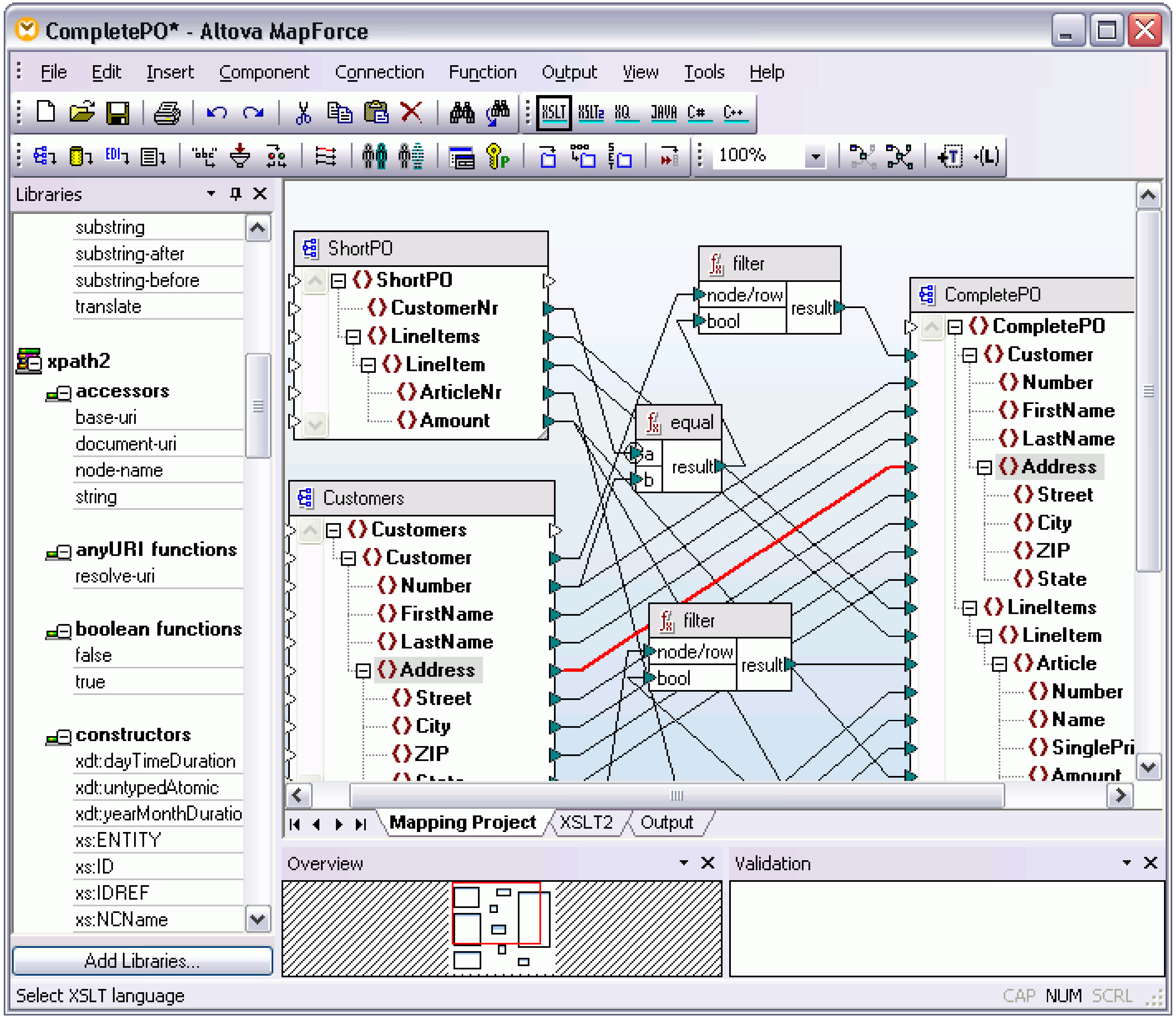
Schema Matching

- Find semantic correspondences between 2 schemas
  - DB schemas, XML schemas, web service interfaces, ontologies, …
- Key step in many metadata applications
  - Data integration: mediators, data warehouses
  - E-Business: XML message mapping; matching of product catalogs
  - Semantic Web: ontology matching (alignment)
- Input: 2 schemas S_1 and S_2
  - Possibly: instances of S_1 and S_2, *background knowledge*
- Output: Mapping between S_1 und S_2
  - Correspondences between schema components
  - Expressions, e.g. for data transformation
- Need to automate (many schemas, large schemas, error-prone)
  - manual control still necessary, especially for business apps

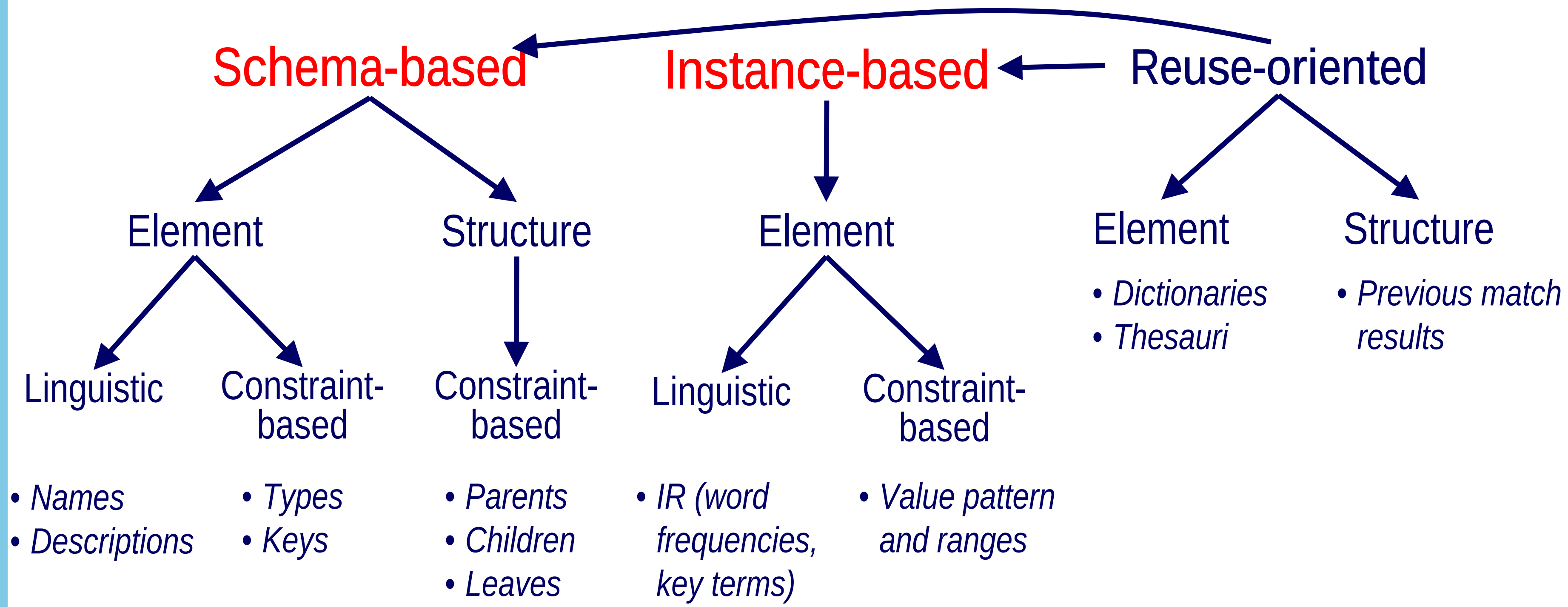
Tool Example: Biztalk Mapper



Tool Example: Altova MapForce



Automatic Match Techniques*



- Combined Approaches: Hybrid vs. Composite

*Rahm, E., P.A. Bernstein: *A Survey of Approaches to Automatic Schema Matching*. VLDB Journal 10(4), 2001

Current Situation

- High research interest in recent years
  - Many papers and several new prototypes:
    LSD (Sigmod01), Cupid (VLDB01), COMA (VLDB02), Clio (IBM), …
  - Semantic web research on ontology matching
- Published results mostly for small and simple schemas
  (< 50 elements, simple types, low degrees of nesting)
- Challenges
  - Match quality for large schemas (Increased likelihood for false matches)
  - Execution time for large schemas (quadratic complexity in schema size)
  - Advanced modeling capabilities, e.g. of W3C XSD, such as distributed schemas / namespaces, user-defined types, alternative design styles, ...
  - Context-dependent matching for shared components
  - User interface for large schemas
  - Verification of proposed match results
  - Evaluation of matchers on large schemas

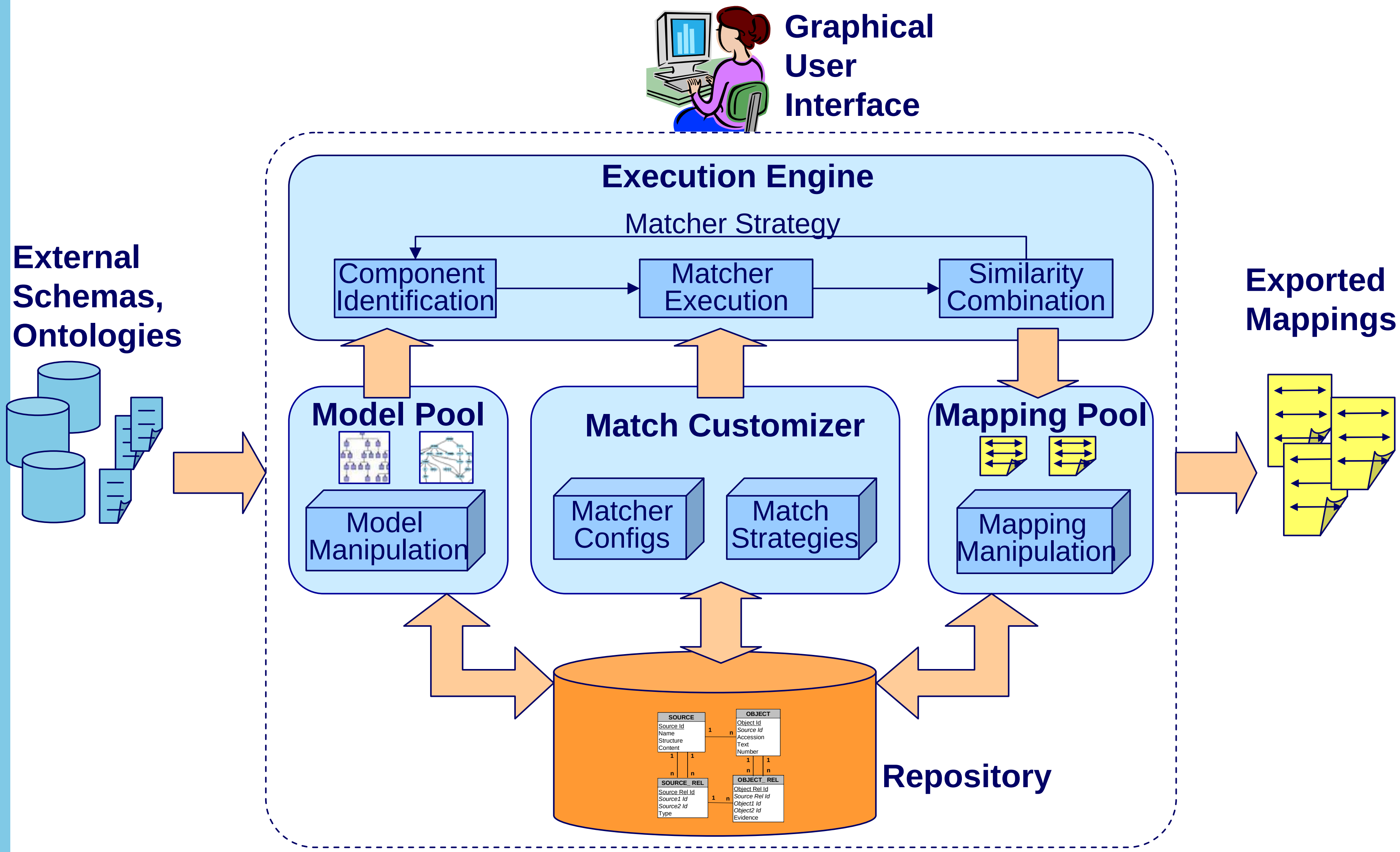
Talk Outline

- System architecture of COMA++
- Schema import
- Basic match processing
- Matcher construction
- Taxonomy matching / ontology support
- Context-dependent matching
- Reuse of previous match results
- Fragment-based matching
- Evaluation

COMA++ Characteristics

- Extends previous COMA prototype (VLDB2002)
- Support of XSD, OWL and relational schemas
- Repository to store schemas and mappings (match results)
- Many matchers including a new taxonomy matcher
- Flexible construction and configuration of matchers and match strategies
- Context-dependent matching for schemas with shared components
- Fragment-based matching for large schemas
- Reuse of previous match results
- Supports comparative evaluation of matchers and match strategies
- GUI
- Much faster than COMA

System Architecture



Import / Design Unification

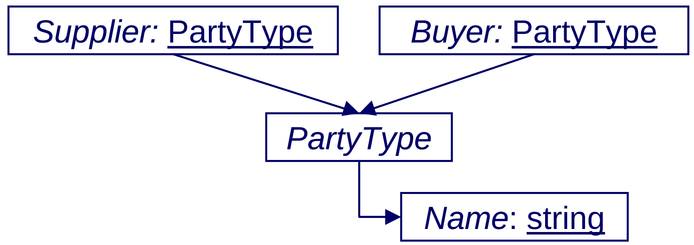
- Transform distributed schemas to monolithic

po.xsd

```
<include schemaLocation="PartyType.xsd"/>
<element name="Supplier" type="PartyType"/>
<element name="Buyer" type="PartyType"/>
```

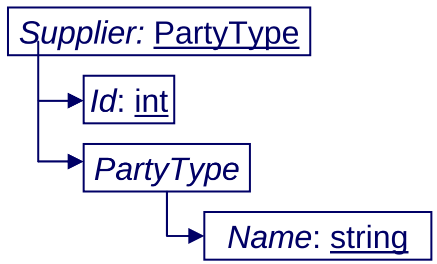
PartyType.xsd

```
<complexType name="PartyType">
   <element name="Name" type=string/>
</complexType>
```



- Transform type derivation to composition

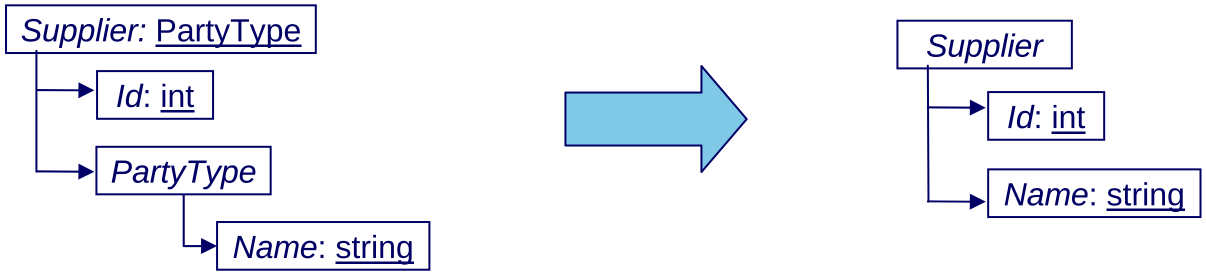
```
<complexType name="Supplier">
   <extension base='PartyType>
      <element name="Id" type="int"/>
   </extension>
</complexType>
<complexType name="PartyType">
   <element name="Name" type="string"/>
</complexType >
```



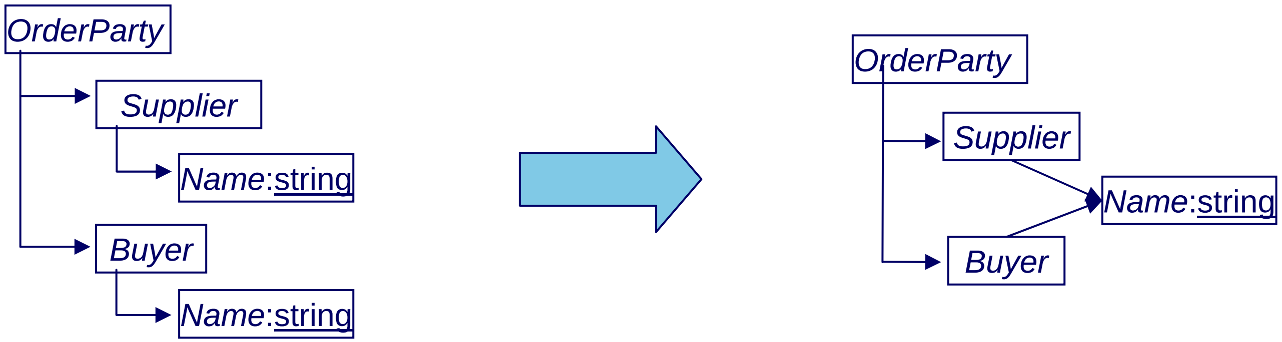
Legends: *Name*: Type → Containment

Design Unification (Cont)

- Transform type reuse to element reuse



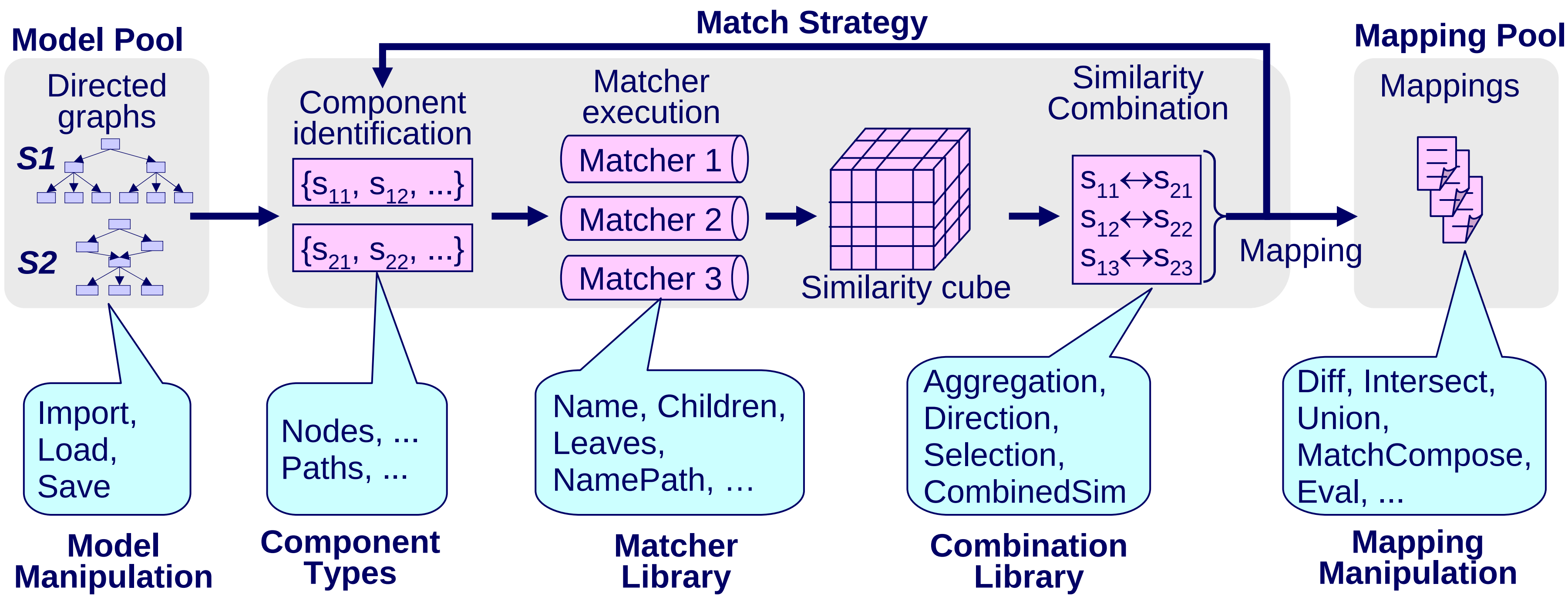
- Reducing inline declarations: Transform inline to shared



- Result:
  - Connected graph of instantiable components (elements, attributes)
  - Fewer schema components with max number of shared elements

Match Processing

- *Combine*: Composite approach to combine independently executed matchers
- *Refine*: Successive refinement of previously identified match results
- Default and user configuration



Example: Schema Elements

match(S1, S2)

1. *Determine all paths from roots of S1, S2*
2. *Execute Name & NamePath matchers*

S1 element	S2 element	Matcher	Sim
ShipTo.shipToCity	*DeliverTo.Address.City*	Name	0.6
		NamePath	0.8
ShipTo.shipToStreet	*DeliverTo.Address.City*	Name	0.5
		NamePath	0.7

A) Similarity Cube

3. *Aggregation*

Average

S1 element	S2 element	Sim
ShipTo.shipToCity	*DeliverTo.Address.City*	0.7
ShipTo.shipToStreet	*DeliverTo.Address.City*	0.6

B) Similarity Matrix

4. *Direction*

|S1|<|S2| SmallLarge
Ranking S1 elements
for elements of larger schema S2

S1 element	S2 element	Sim
ShipTo.shipToCity	*DeliverTo.Address.City*	0.7
ShipTo.shipToStreet	*DeliverTo.Address.City*	0.6

C) Directional Ranking

5. *Selection*

Max1

S1 element	S2 element	Sim
ShipTo.shipToCity	*DeliverTo.Address.City*	0.7

D) Match Results

CombineMatcher

- Interactive construction of new matchers from existing ones
- Combining results of independently executed matchers

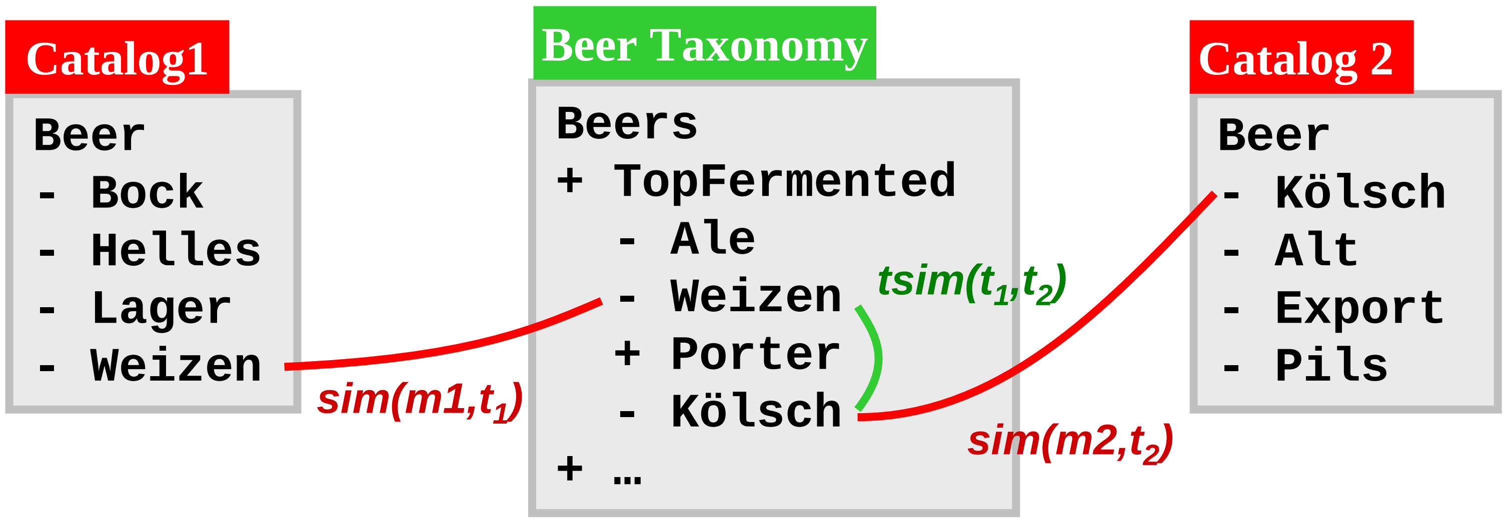
```
CombineMatcher(oType, defMatchers, agg, dir, sel)

match(s1, s2) {
 //Step1: Determine elements/constituents to match
 s1.objects = determineObjects(oType, s1)
 s2.objects = determineObjects(oType, s2)
 //Step2: Compute similarity cube
 allocate simCube[compMatchers][s1.objects][s2.objects]
 for each m in defMatchers
    for each o1 in s1.objects
       for each o2 in s2.objects
          simCube[m][o1][o2] = m.sim(o1, o2)
 //Step 3, 4, 5: Similarity combination
 matchResult = selection(
                 direction(
                    aggregation(simCube, agg), dir), sel)
 return matchResult
}
```

Matcher Library

- Simple matchers:
  - String matchers: EditDistance, Trigram,, …
  - Type matcher, Synonym matcher
  - Statistics (Euclidean distance between structural statistics captured by a feature vector)
  - Taxonomy matcher, Reuse matcher
- Predefined combined matchers

Name	*Elements / Constituents*	*Default Matchers*	*Combination Scheme*
Name	Name tokens	Synonym, Trigram	Avg, Both, Max1, Avg
NameType	Self	Name, Type	Wgt(0.7,03), Both, Max1, Avg
NameStat	Self	Name, Statistics	
Children	Children	NameType	Avg, Both, Max1, Avg
Leaves	Leaves	NameType	
Parents	Parents	Leaves	
Siblings	Siblings	Leaves	
NamePath	Ascendants	Name	

Taxonomy Matcher



- Reference taxonomy helps find correspondences

  `sim(Weizen,Kölsch) = 0.8`

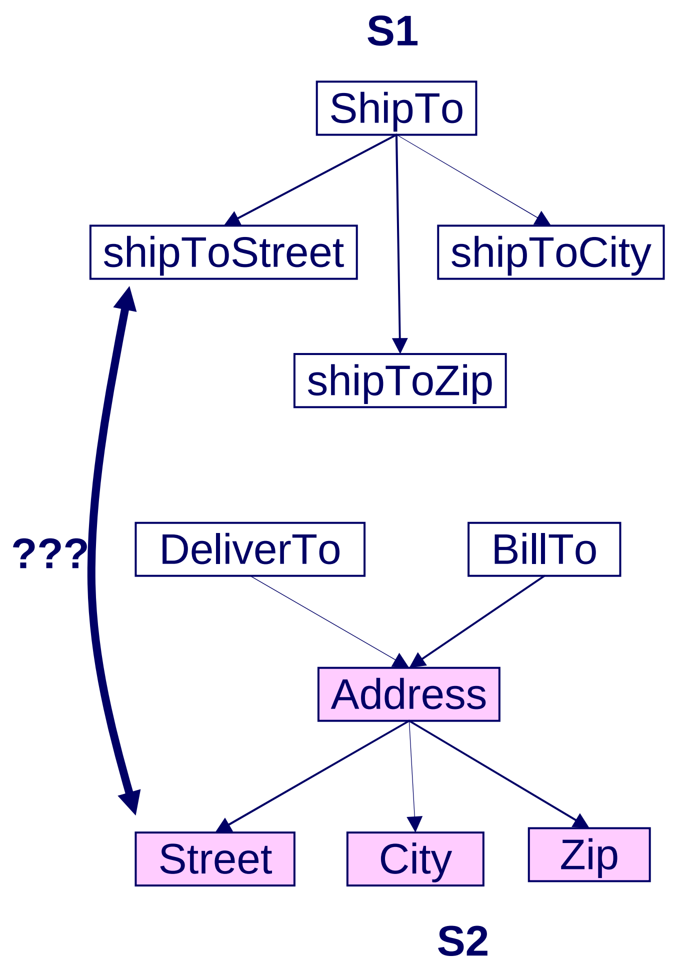
- Similarity of schema elements → combination of *sim(m1,t1) tsim(t1,t2) and sim(m2,t2)*
- *tsim:* measures semantic distance between concepts *within* taxonomy
  - different approaches possible
  - precomputation of tsim for most related concepts

OWL Support

- Ontology access using OWL API
- Ontology Matching can use all existing matchers, e.g. Name and structural matchers
- High effectiveness for EON Ontology Alignment test tasks even without tuning (no synonyms, etc.)

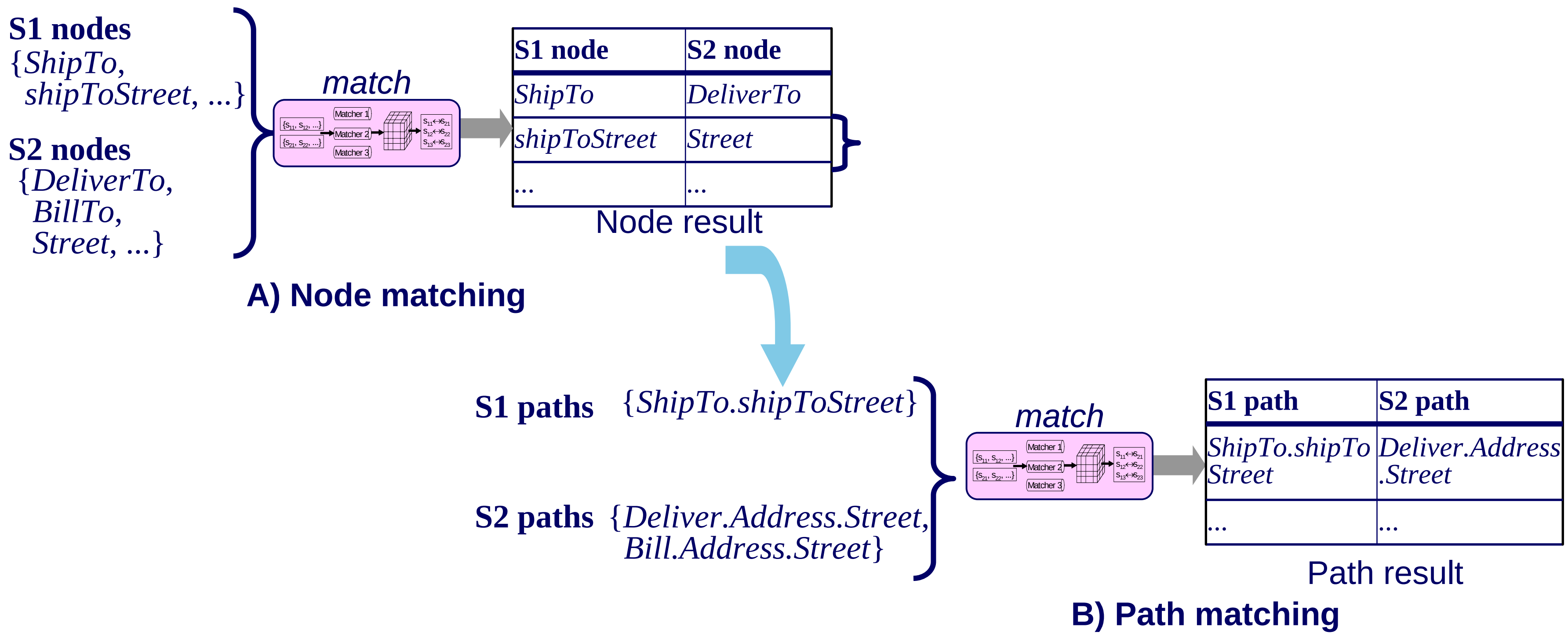
Context-dependent Matching

- Problem: Shared components with context-dependent semantics
  - e.g. user-defined types
- NoContext strategy: most previous systems
  - only node correspondences
  - efficient but poor match quality:
    e.g. *shipToStreet* ↔ *Street*
- AllContext strategy: Cupid, COMA
  - match between all unique contexts, e.g. paths
  - scalability problems for large schemas with many shared components
  - explosion of search space (|S1 paths|*|S2 paths|)



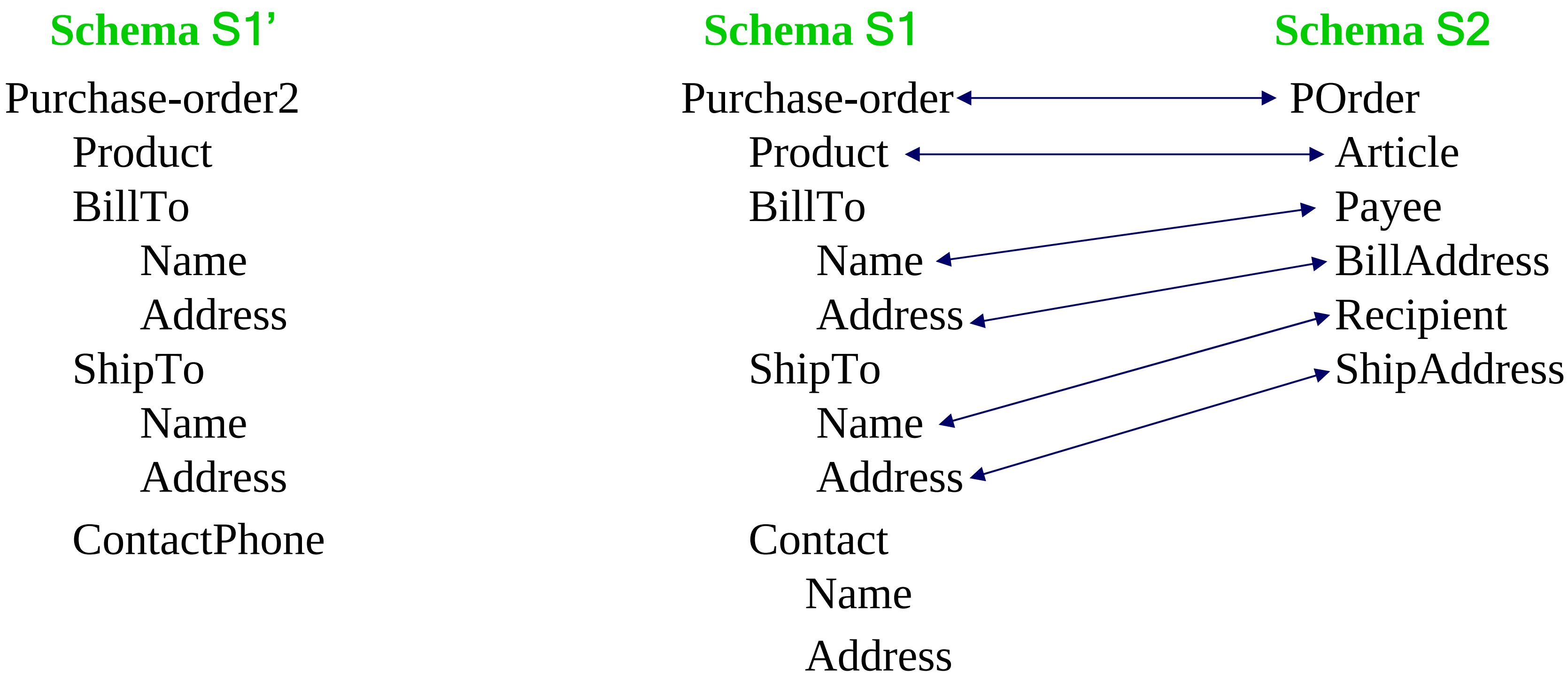
FilteredContext Strategy

- Two-phase matching
  1. Node matching with multiple matchers
  2. Context (path) matching only for most similar node pairs
- Complexity: mainly in node matching (comparable to NoContext)
  - Significant reduction of complexity (if |nodes|<<|paths|)



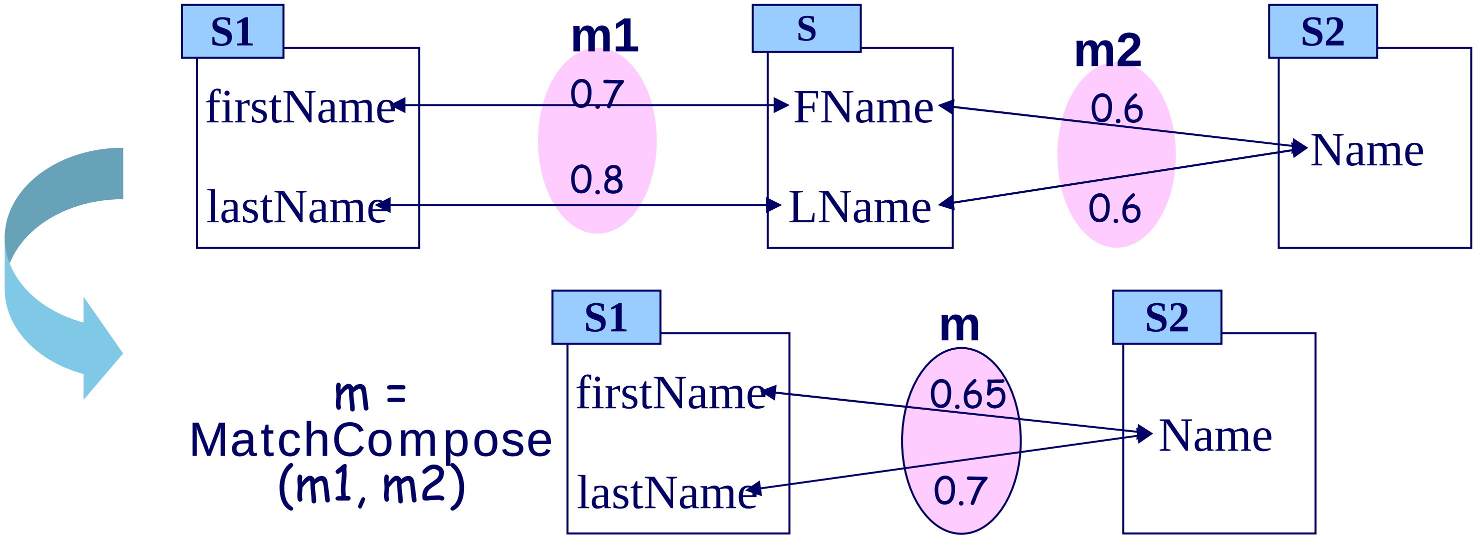
Reuse of Previous Match Results

- Example: Use result for S1—S2 to match S1'—S2



Reuse (2)

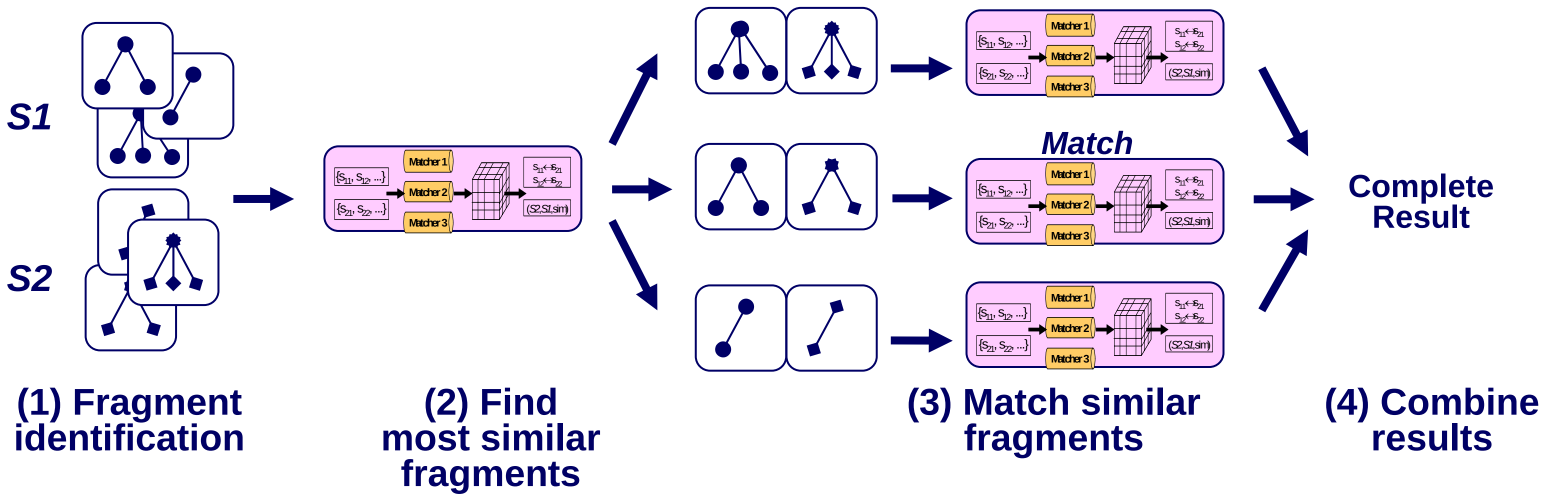
- MatchCompose operation



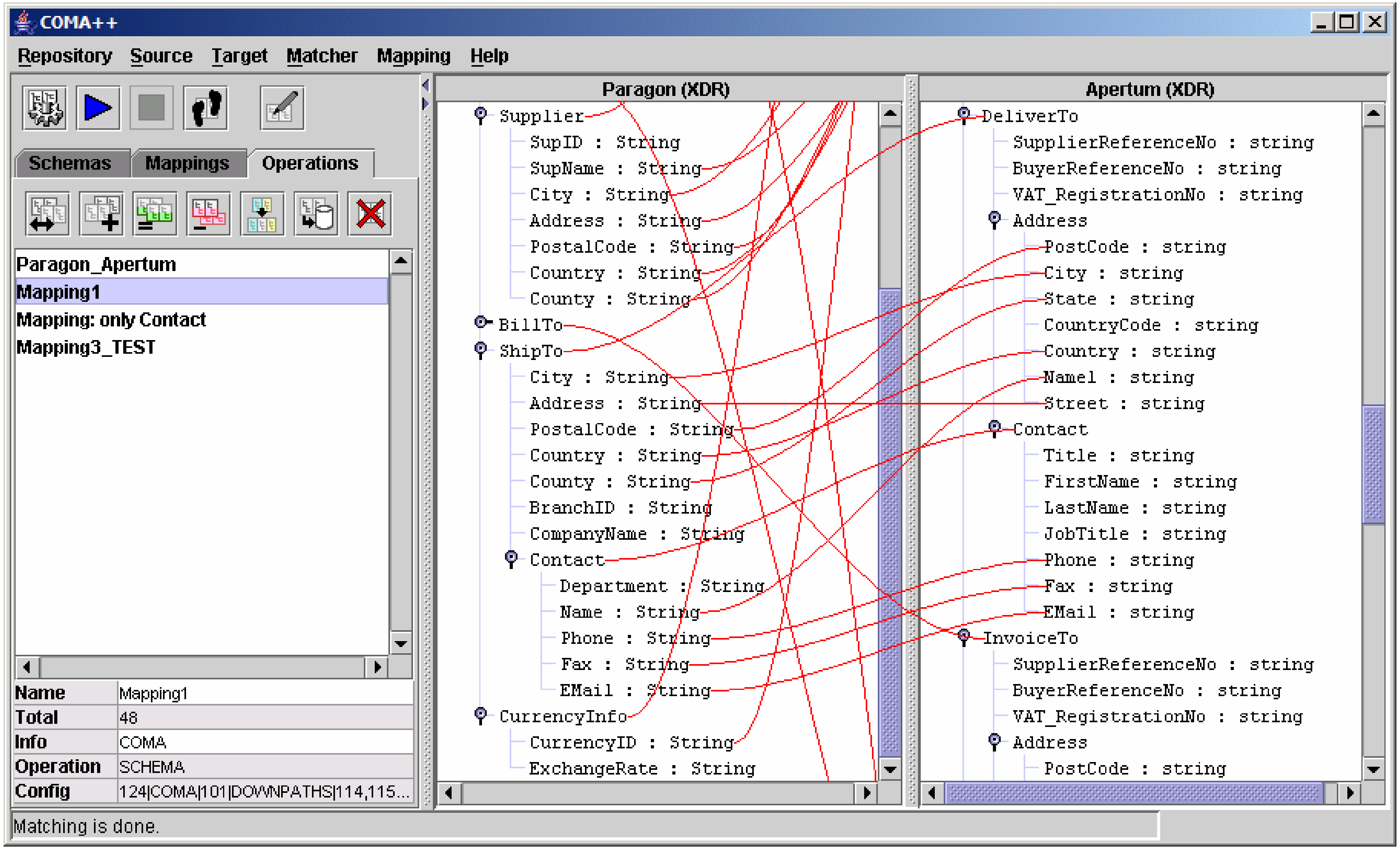
- COMA++ reuse options using mapping repository (match problem S1-S2)
  1. Use already existing direct mapping(s) S1-S2
  2. Look for *mapping paths* (S1-S3-S2, S2-S4-S5-S1, …)
  3. Use of a *pivot schema* (global schema) P: Compose S1-P with P-S2
  4. Look for similar mappings, e.g. for different schema versions

Fragment-based Matching

- Decompose large match problems into smaller ones
  - Reduce search space and potential for false matches
  - Simplify user control
- Fragments: Rooted subgraphs down to leaf level
  - Special case: complete schema
  - Instantiable subschemas, e.g. tables, message formats
  - Shared schema components
  - user-selected fragments



Graphical User Interface



Real-world Evaluation

- PO schemas (XDR) & E-Business standards (XSD)

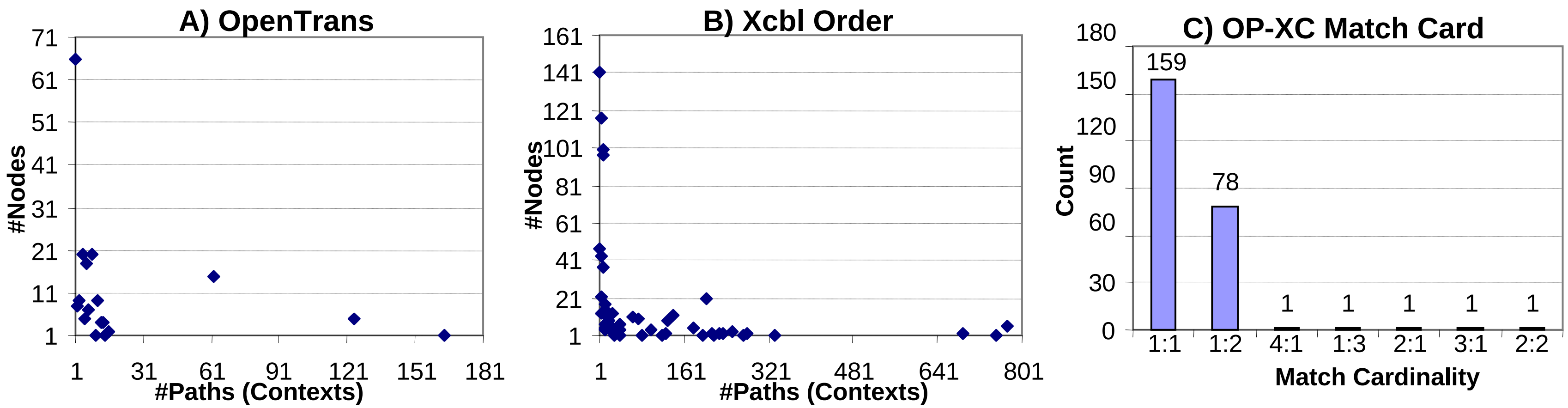
#	*Schema*	*#Nodes*	*#Roots / Inners/ #Leaves / Shared*	*#Paths*	*Max/Avg Path Len*
1	CIDX	27	1 / 7 / 20 / 7	34	4 / 2.9
2	Excel	32	1 / 9 / 23 / 11	48	4 / 3.5
3	Noris	46	1 / 8 / 38 / 18	65	4 / 3.2
4	Paragon	59	1 / 11 / 48 / 13	77	6 / 3.6
5	Apertum	74	1 / 22 / 52 / 24	136	5 / 3.6
6	OpenTrans	195	8 / 85 / 110 / 129	2,500	11 / 7
7	XCBLOrder	843	10 / 382 / 461 / 702	26,228	18 / 8.8

Match Tasks

- 16 match tasks in 3 series

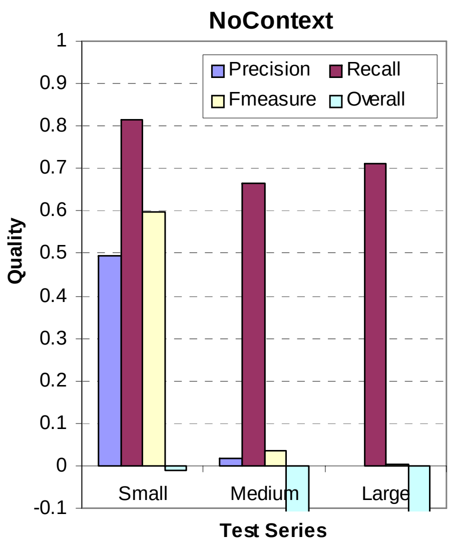
Series	*Match tasks*	*#Tasks*	*Avg Source paths*	*Avg Target paths*	*Avg Corresp*	*Avg Schema Sim*
Small	PO-PO	10	49	95	48	0.57
Medium	PO-OP	5	72	2,500	55	0.04
Large	OP-XC	1	2,500	26,228	331	0.02

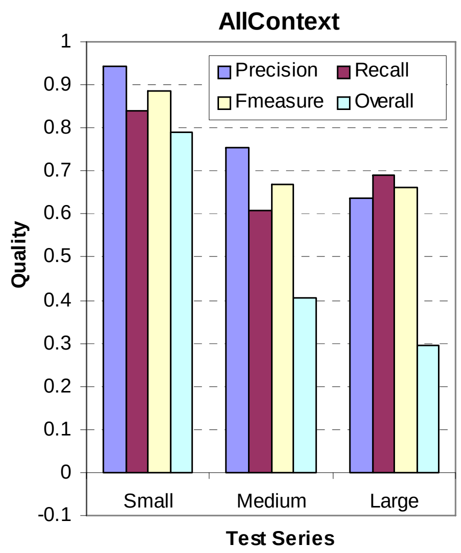
- Largest match task: OpenTrans – XcblOrder
  - # contexts per node up to 160 in OpenTrans and 800 in Xcbl Order
  - many m:n match relationships
  - traditional approaches, i.e. AllContext, MaxN or Threshold, unlikely successful

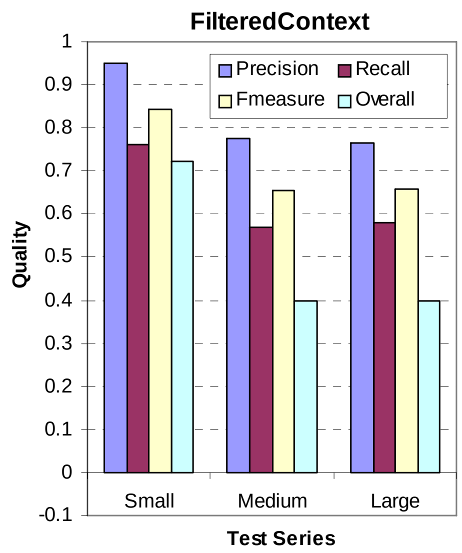


Context Matching: Quality

- Quality decreases with increasing schema size, i.e. harder problem
- NoContext: Low quality even in small series
  - high recall but low precision, i.e. many false matches, fmeasure ~0.0
  - Unacceptable with many shared components
- FilteredContext slightly worse than AllContext in Small, but equal in Medium, Large

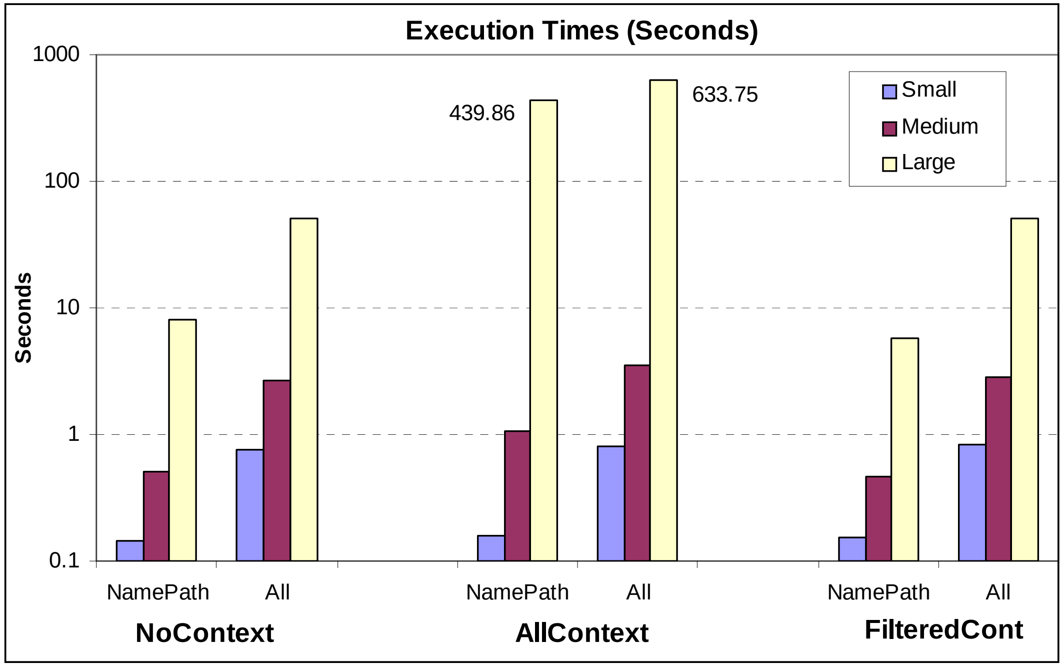






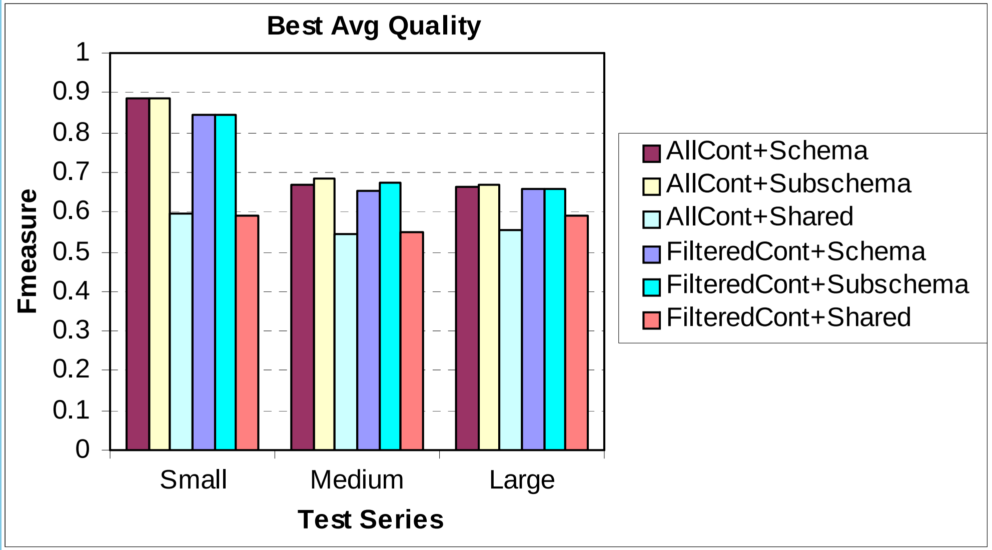
Context Matching: Time

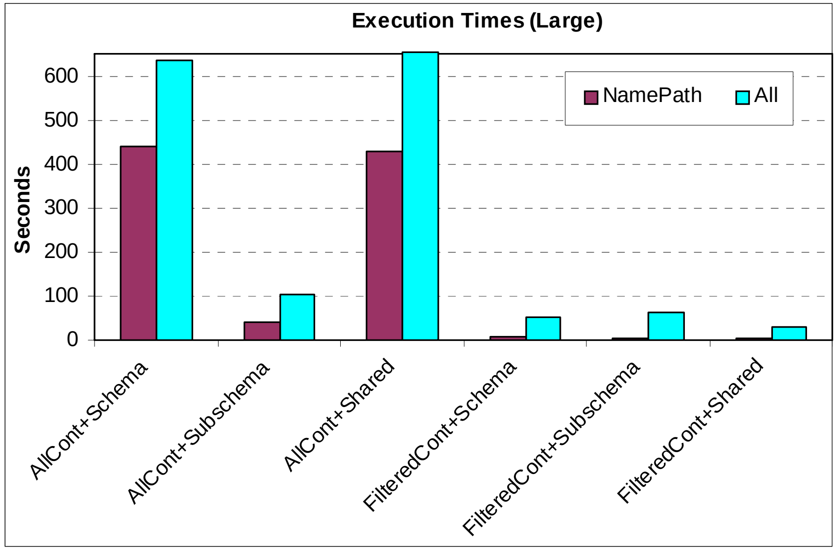
- 1 matcher (NamePath) vs. 8 matchers (All)
- Time increases with schema size & number of matchers
- Similar, fast execution times in Small, Medium, but differences in Large
- NoContext: fast execution times, max ~50 secs for Large
- AllContext: not scalable, 7-10 min for Large
- FilteredContext: ~ NoContext indicating negligible effort for path matching -> best context-dependent strategy for large schemas



Fragment Matching: Quality and Time

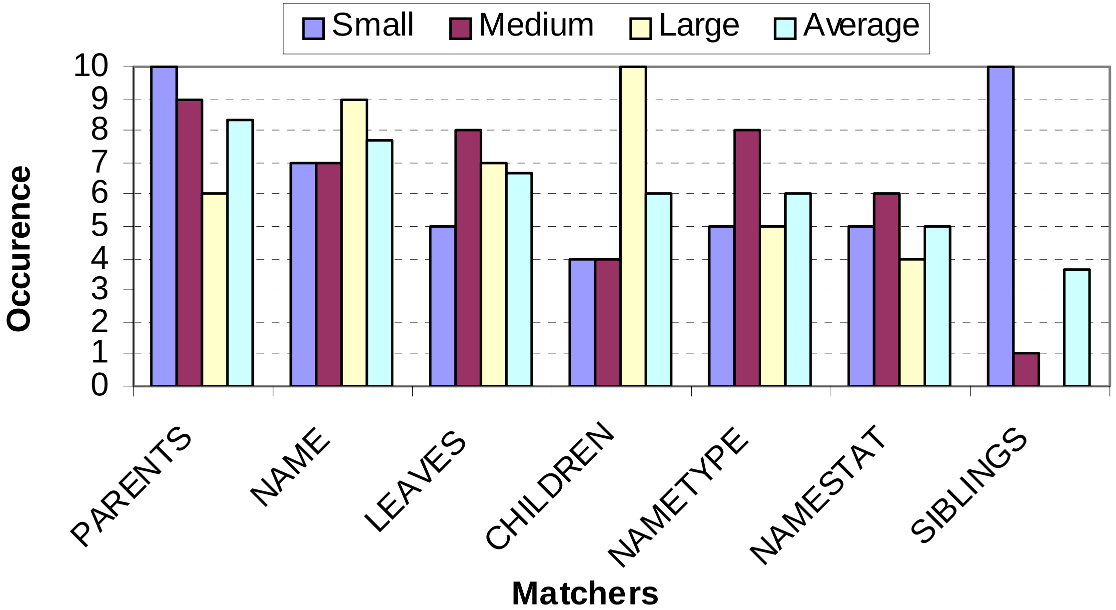
- Quality:
  - Subschema equal or better than Schema due to reduced search space
  - Shared worst due to incomplete schema coverage, still attractive for large schemas
- Time (Large series):
  - Significant improvement of Subschema over Schema in AllContext
  - Shared ~ Schema in AllContext: large number of Shared elements (i.e. fragments) to be compared
  - Similar, fast times between fragment types in FilteredContext





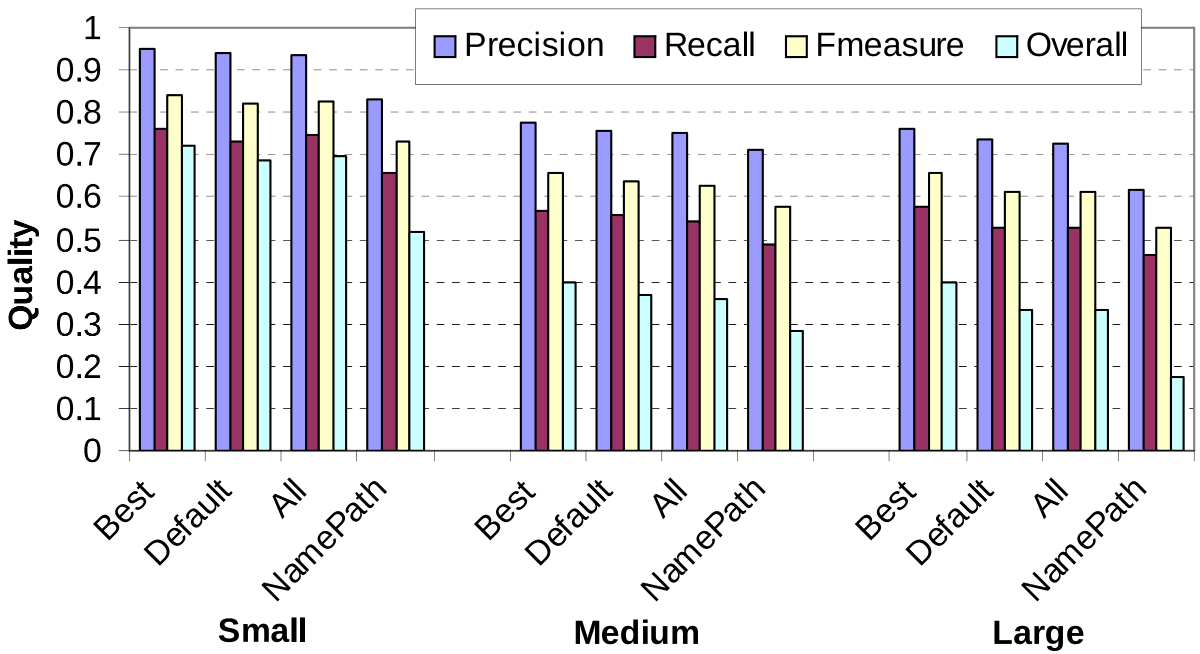
Combination of Matchers (1)

- *Method*:
  - 128 matcher combinations (NamePath + selection from 7 other matchers)
  - For each matcher, determine #occurrences within the top-10 matcher combinations
- FilteredContext: more stable matcher occurrence than AllContext
  - Parents, Names, Leaves with high occurrence (>=5) in all series
  - -> Default configuration: NamePath + Parents, Name, Leaves



Combination of Matchers (2)

- *Method*: Best average Fmeasure of Best, Default, All and NamePath for FilteredContext
- NamePath:
  - worse quality than others
  - combining several matchers for better quality
- Default ~ Best ~ All in all series
  - All: very expensive, Default: good combination with only 4 matchers



Conclusions and Future Work

- Comprehensive platform for schema and ontology matching
  - Flexible construction of new matchers, match strategies
  - Strategies for context-dependent matching, fragment-based matching, reuse
  - Repository of schemas and mappings
  - GUI
- Evaluation results
  - New match strategies with high quality and fast execution times for large schemas
  - Fmeasure ~0.9 for small tasks, ~0.7 for large tasks
  - FilteredContext combines good quality and fast execution
  - Fragment matching: Subschema better than Schema in both quality and time
  - Default combination of 4 matchers (NamePath, Parents, Name, Leaves) delivers good results
- Future work
  - Evaluation of reuse
  - Additional approaches for ontology matching
  - Instance matching

References

- Aumüller, D., Do, H.H., Massmann, S., Rahm, E:
  Schema and Ontology Matching with COMA++.
  Proc. SIGMOD 2005 (Software Demonstration), Baltimore, June 2005
- Rahm, E., Do H.H., Massmann, S.: Matching Large XML Schemas.
  Sigmod Record 33(4), December 2004
- Do, H.H., Melnik, S., Rahm, E.: Comparison of Schema Matching Evaluations. In: Web, Web-Services, and Database Systems. Springer-Verlag, LNCS 2593, 2003
- Do, H.H.; Rahm, E.: COMA - A system for flexible combination of schema matching approaches. Proc. 28th Intl. Conf. on Very Large Databases (VLDB), Hongkong, Aug. 2002
- Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching.
  VLDB Journal, Vol. 10, No. 4, Dec. 2001

http://dbs.uni-leipzig.de