www.social-nexus.net

User Profile Integration Made Easy – Model-Driven Extraction and Transformation of Social Network Schemas

Martin Wischenbart, S. Mitsch, E. Kapsammer, A. Kusel, B. Pröll, W. Retschitzegger, W. Schwinger, J. Schönböck, M. Wimmer, S. Lechner

MultiA-Pro Workshop, WWW Conference, Lyon, France, April 16th, 2012

Information Systems Group
Johannes Kepler University Linz
http://www.ifs.uni-linz.ac.at/

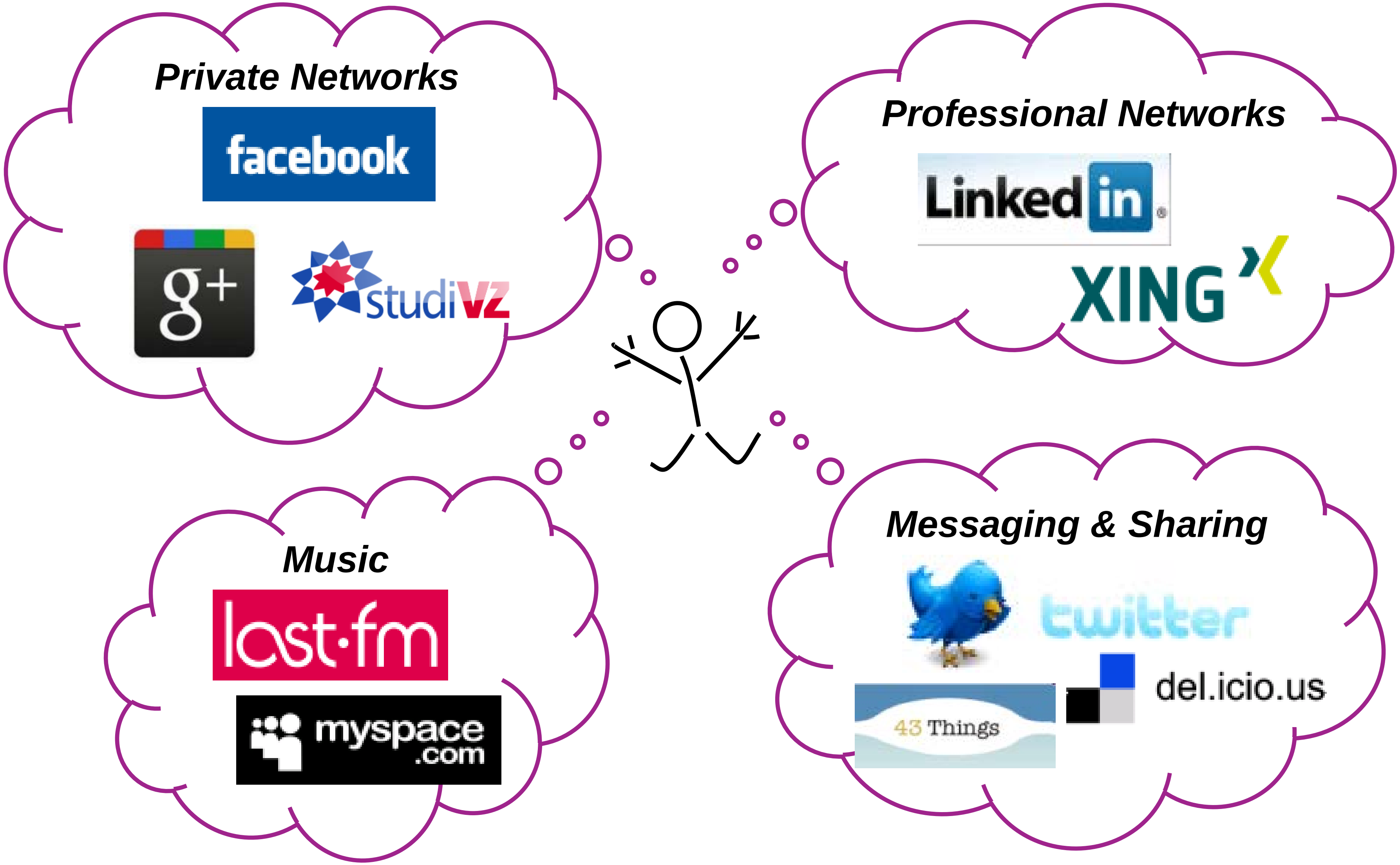
Business Informatics Group
Vienna University of Technology
http://www.big.tuwien.ac.at/

Netural GmbH
http://www.netural.com/

Funding: FIT-IT 825070

Motivation

Social Networks & User Profile Data



Private Networks

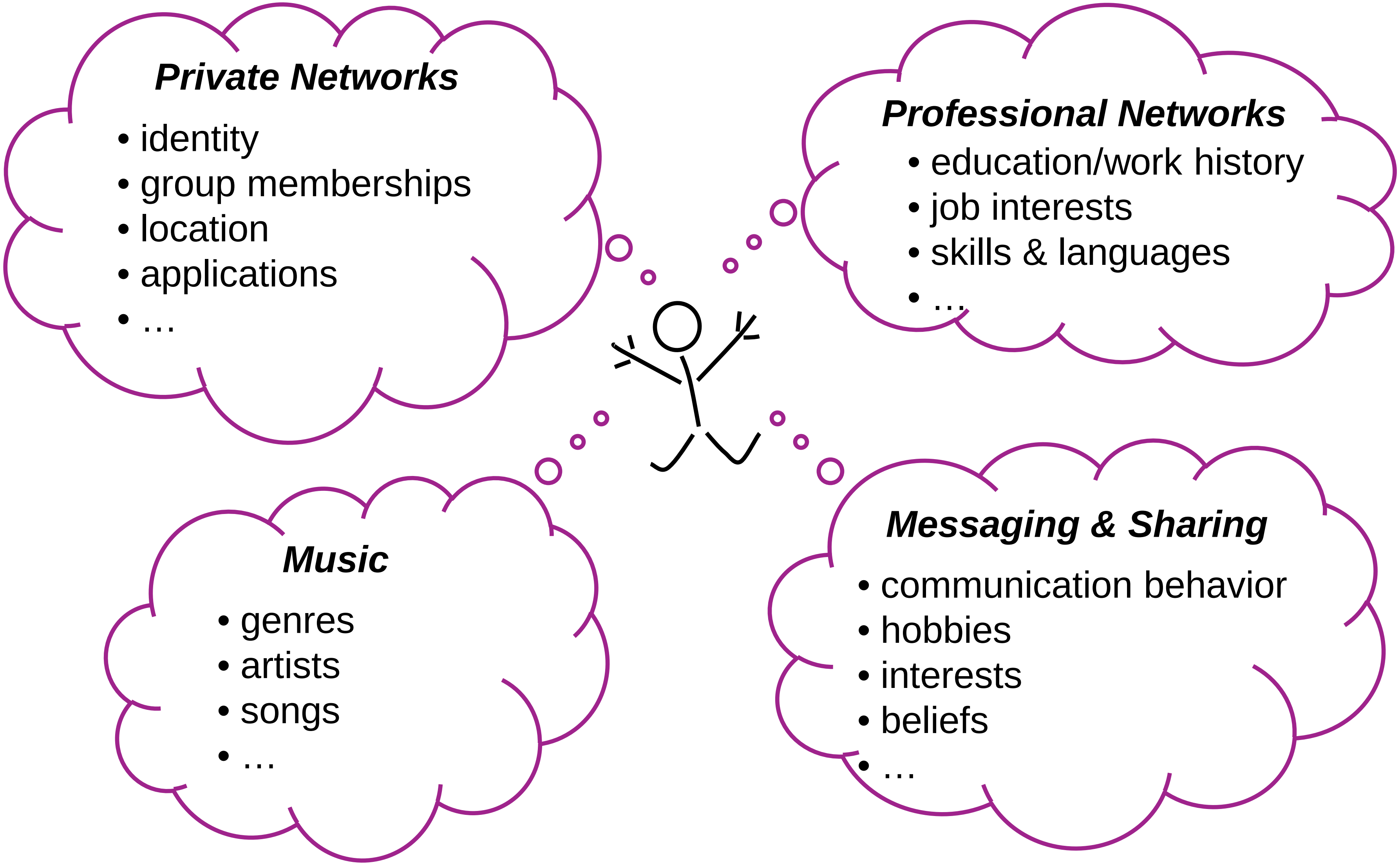
Professional Networks

Music

Messaging & Sharing

Motivation

Social Networks & User Profile Data



Private Networks

- identity
- group memberships
- location
- applications
- …

Professional Networks

- education/work history
- job interests
- skills & languages
- …

Music

- genres
- artists
- songs
- …

Messaging & Sharing

- communication behavior
- hobbies
- interests
- beliefs
- …

Motivation

Schema Information – What For?

Needed for **various data processing tasks**:

- **Search**
- **Manipulation**
- **Optimization**
- **Translation**
- **Evolution**
- **Integration**

→ integrate user **data from multiple social networks** to achieve comprehensive profiles for **recommender** applications

Research Project

the hidden U

www.social-nexus.net

Motivation
Schema Information – Sources?

Social networks often use **NoSQL-DBs**

- **No traditional schema** (e.g., HBase in Hadoop)
- **Schema-less** multidimensional maps (e.g., Apache Cassandra)

→ **Explicit schemas not available**

RESTful social network **APIs**:

- Leading data format: **JSON** (supported by all surveyed social networks)
- Authentication: OAuth 1.0a/2.0

→ **Documentation (schema description)** is often **exemplary or outdated**

Evolution of social networks and APIs

- Requires **adaptation of existing** schemas and applications

→ **Repetitive manual creation** of **up-to-date schemas** is **not an option**

Motivation

Goal: Derive Schemas from Instances

- Semi-automatically **derive schemas** **from instance data**
- **Strategies** to handle **specifics** of social networks and **JSON** (**JSON Schema**)
- **Transformation** to different technical spaces (ECORE, XML Schema, OWL)

  → **Application** of **existing integration tools**
- Evaluate approach with **Facebook, Google+, LinkedIn**

Outline

- Related Work
- JSON Data & Schema
- Approach
  - Schema Extraction (3 steps)
  - Transformation
- Results & Evaluation
  - Test User Profile Setup
  - Extracted Schemas
  - Comparison to Documentation
  - Outlook on Integration
- Conclusions

Related Work
Approaches for Automatic Schema Extraction

- **No focus** on **JSON Schema** so far
- Generation of **DTDs or XML Schemas**
  (Bex et al. [4], Eki et al. [7], Hegewald et al. [10], Mylnkova [18] (survey))
  - Must use **XML APIs** or **transform instances before**
  - **Specificity** of extracted schemas
  - No **configurability** to social network APIs, does **not consider peculiarities of social networks**
- **Ontology learning**
  (Drumond et al. [6] (survey), Hazman et al. [8] (survey))
  - Focus on concepts and taxonomic relationships → **disregard non-taxonomic relationships** (i.e., **references between classes**)
- **Meta-models from models** *(Javed et al. [11])*
  - **Evolution** requires high number of transformations and grammars
  - **Not flexible and reusable** enough

→ Existing approaches **not applicable in social network integration scenario**

JSON Data & Schema
Example of Extracted Data

unique id of `user` object (= key)

```
{
  "id": "100002345678964",
  "name": "Jane Doe",
  "birthday": "04/18/1978",
  "gender": "female",
  "type": "user",
  "work": [
    {
      "employer": {
        "id": "106119876543210",
        "name": "Doe Inc."
      },
      "start_date": "2007-08"
    }, {
      "start_date": "2004",
      "end_date": "2007"
    }
  ]
}
```

foreign key (link to `employer` object)

JSON instance

```
{
  "id": "106119876543210",
  "name": "Doe Inc.",
  "picture": "http://www.doe.net/logo.jpg",
  "link": "http://www.facebook.com/doeinc",
  "likes": 25946937,
  "category": "Food/beverages",
  "username": "doeinc",
  "founded": "April 1st. Seriously.",
  "company_overview": "Doe Power Drink is a
     functional beverage. Thanks to a
     unique combination of high quality
     ingredients Doe Power Drink vitalizes
     body and mind. \n\n Doe Power Drink
     has been developed for people who want
     to have a clear and focused mind,
     perform physically, are dynamic and
     performance-oriented whilst also
     balancing this with a fun and active
     lifestyle. \n\n In short, Doe Power
     Drink gives wings to people who want
     to be mentally and physically active
     and have a zest for life."
}
```

JSON instance

JSON Data & Sche
JSON Schema

conforms to

```
{
  "id": "100002345678964",
  "name": "Jane Doe",
  "birthday": "04/18/1978",
  "gender": "female",
  "type": "user",
  "work": [
    {
      "employer": {
        "id": "106119876543210",
        "name": "Doe Inc."
      },
      "start_date": "2007-08"
    }, {
      "start_date": "2004",
      "end_date": "2007"
    }
  ]
}
```

JSON instance

```
{
  "type": "object",
  "id": "user",
  "properties": {
    "id": { "type": "string" },
    "name": {"type": "string" },
    "birthday": { "type": "string",
                  "pattern":
                   "[0-9]{2}/[0-9]{2}/[0-9]{4}" },
    "gender": { "type": "string",
                "enum": ["male", "female"] },
    "type": { "type": "string" },
    "work": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "employer": {
              "type": "object",
              "id": "employer",
              "properties": {
                "id": { "type": "string" },
                "name": {"type": "string" }
              }
            },
            "start_date": { "type": "string" }
          }
        }, {
          "type": "object",
          "properties": {
            "start_date": { "type": "string" },
            "end_date": { "type": "string" }
} } ] } } }
```

5 properties of **primitive** types

1 property of **complex** type array

JSON schema

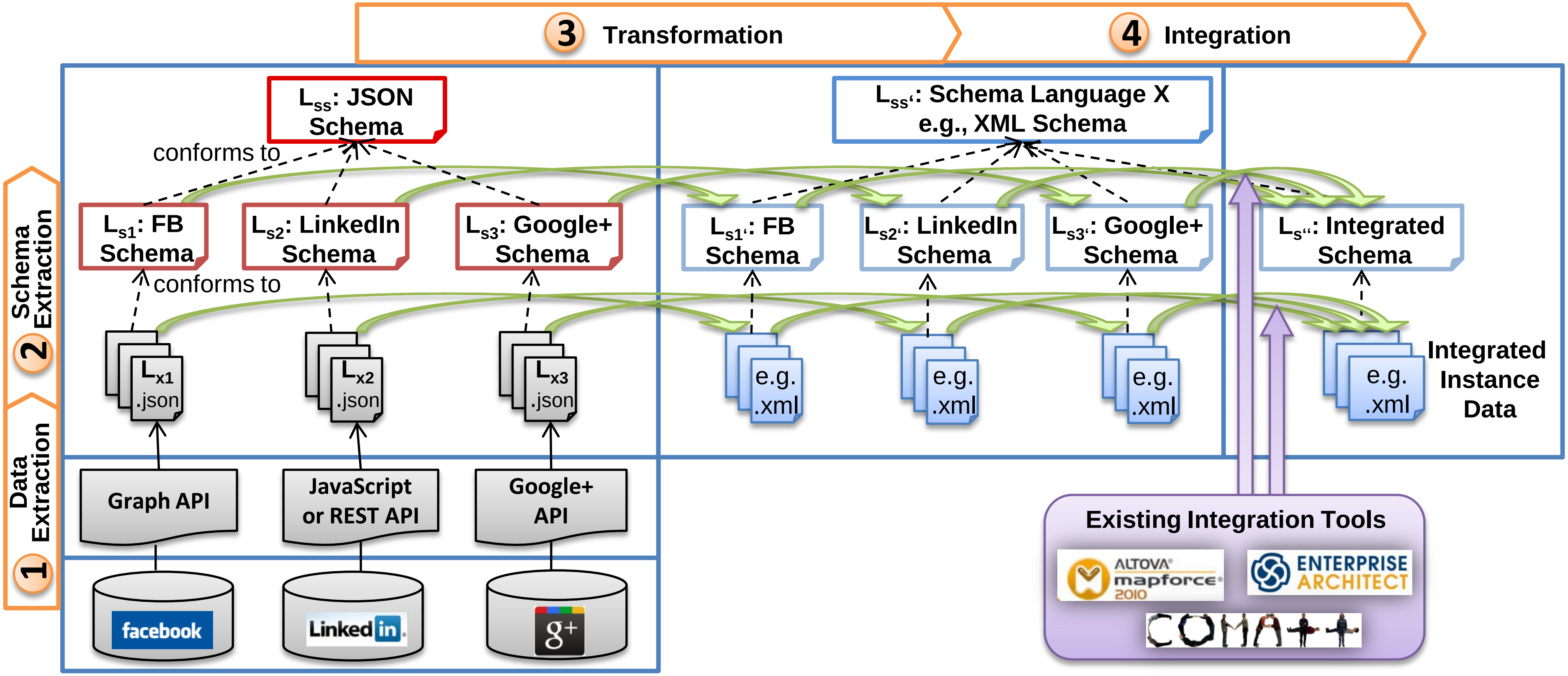
Approach
Four Phases to Integration – Overview

❶ **Data Extraction:** **extract instance** data from social networks via APIs (**JSON**)

❷ **Schema Extraction:** **derive** separate **schemas** for each social network, corresponding to a schema language (**JSON Schema**)

❸ **Transformation:** transform to different **technical space** (**XML Schema/XML**)

❹ **Integration:** **integrate** (**integrated XML Schema/XML**)



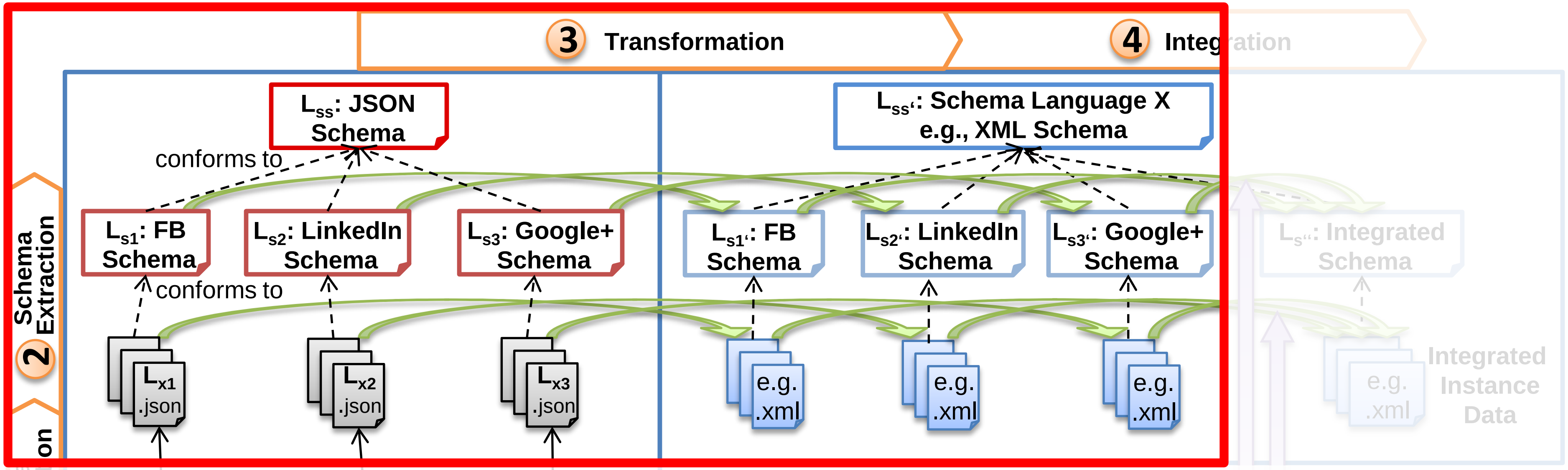
Approach
Four Phases to Integration – Overview

❶ **Data Extraction:** extract instance data from social networks via APIs (**JSON**)

❷ **Schema Extraction:** **derive** separate **schemas** for each social network, corresponding to a schema language (**JSON Schema**)

❸ **Transformation:** transform to different **technical space** (**XML Schema/XML**)

❹ **Integration:** **integrate** (**integrated XML Schema/XML**)



Phase ❷: Schema Extraction

(1) **Generalization Strategies**

(2) **Merging and Clearance**

(3) **Refactoring**

Phase ❸: Transformation

Approach
Phase ❷: Schema Extraction – 3 Steps

(1) Generalization Strategies

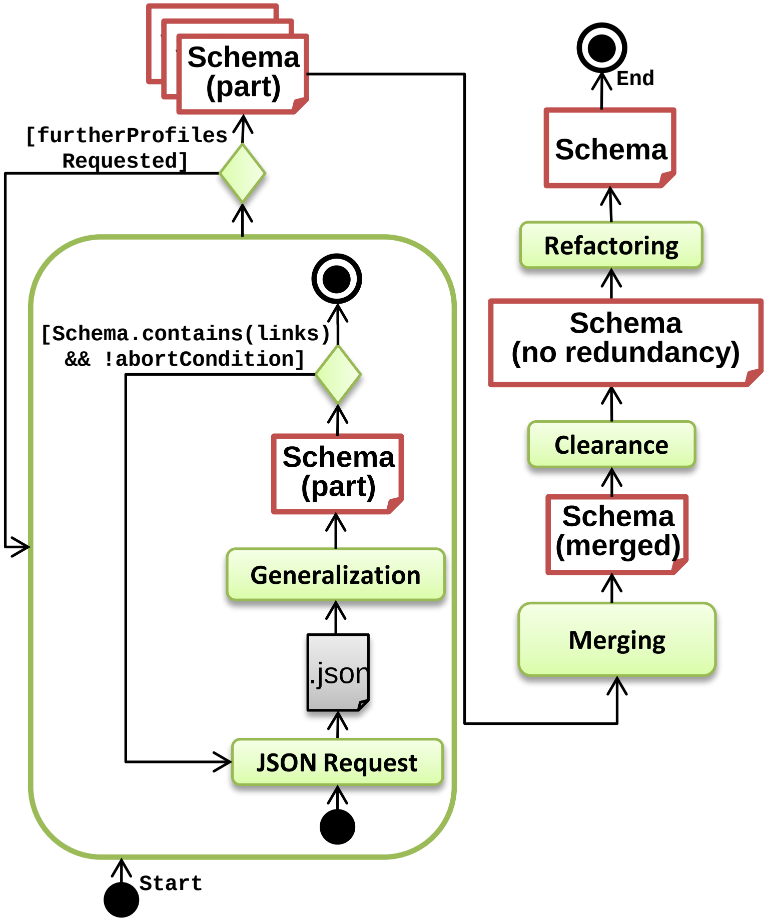
- Create **schema parts** from instances
- Introduce **links** between schema parts

(2) Merging and Clearance

- **Merge** linked schema **parts** into single schema
- **Clear duplicate** schema parts (merge into coherent schema)

(3) Refactoring

- Build **class hierarchy**
- **Homogenize** array types
- **Lookup** class names **in ontology**



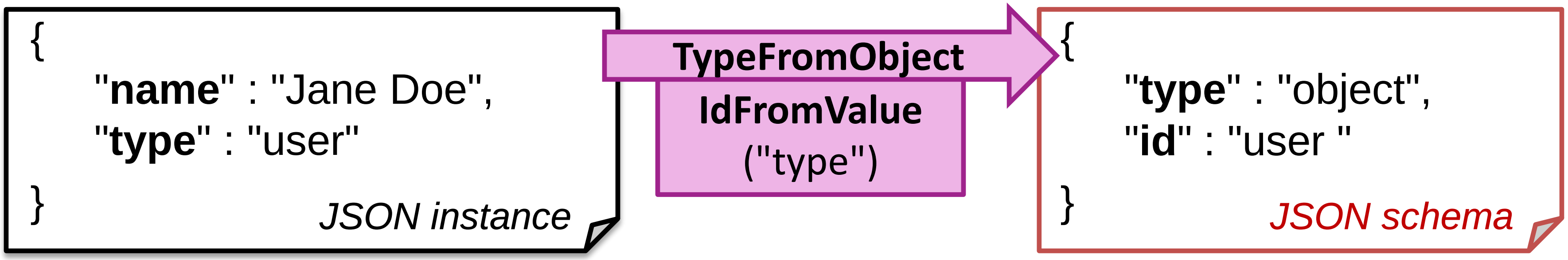
Approach
Phase ❷-1: Generalization Strategies

Different **alternative strategies available**

- **Configure once** per **social network**

	Strategy	Configuration Options	Priority	Optional	Description
Type Extraction	TypeFromObject				derives a type for each object
	IdFromValue	names of keys	1		derives the name of a type from the value of a property
	IdFromReferenceName		2		derives the name of a nested type from the reference name
	IdFromNameConcat		3		derives the name of the type by concatentating the names of the contained properties
Property Extraction	PropertyFromKeyValuePair				derives a property for each key/value pair
	NameFromProperty				derives the name of the property from the key of the key/value pair
	TypeFromValue				derives the type of the property from the type of the value (String, Boolean, Number, Array, Object)
	EnumFromValue	names of keys		✓	derives an enumeration for the key/value pair
	IntervalFromValue	names of keys		✓	derives an interval for the key/value pair
Link Intro.	LinkFromProperty				derives links between types
	LinkRoleFromName	names of keys			derives the role name of the link from the key of a key/value pair
	LinkPatternFromValue				derives the href of the link from values that are valid URLs

Example:



```
{
    "name" : "Jane Doe",
    "type" : "user"
}
```

JSON instance

TypeFromObject

IdFromValue
("type")

```
{
    "type" : "object",
    "id" : "user "
}
```

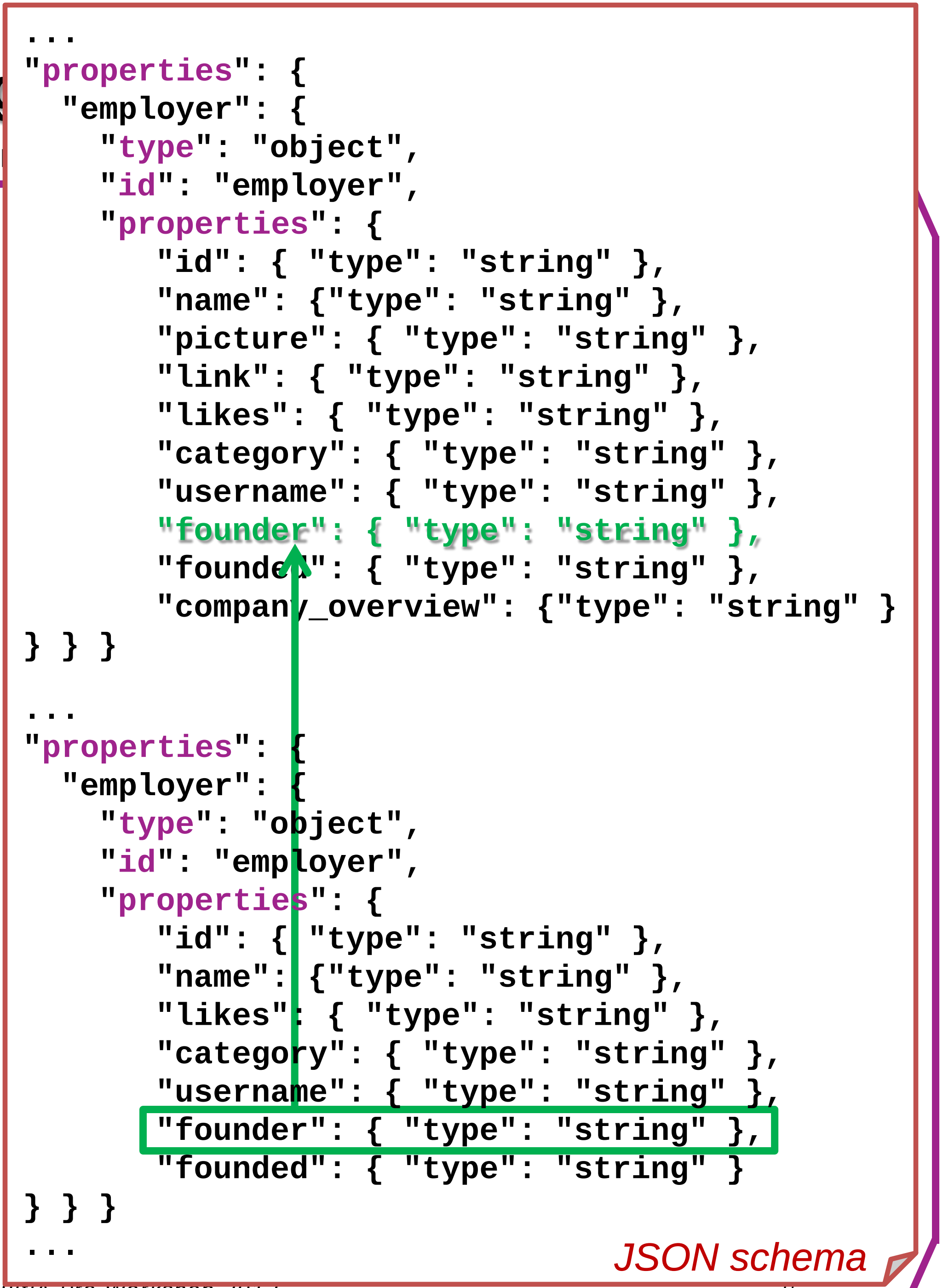
JSON schema

Approach
Phase ❷-2: Merg

- **Merge properties** of equal types
- **Clear duplicates**

Example:

- Merge **multiple employers** to single type



```
...
"properties": {
  "employer": {
    "type": "object",
    "id": "employer",
    "properties": {
       "id": { "type": "string" },
       "name": {"type": "string" },
       "picture": { "type": "string" },
       "link": { "type": "string" },
       "likes": { "type": "string" },
       "category": { "type": "string" },
       "username": { "type": "string" },
       "founder": { "type": "string" },
       "founded": { "type": "string" },
       "company_overview": {"type": "string" }
} } }

...
"properties": {
  "employer": {
    "type": "object",
    "id": "employer",
    "properties": {
       "id": { "type": "string" },
       "name": {"type": "string" },
       "likes": { "type": "string" },
       "category": { "type": "string" },
       "username": { "type": "string" },
       "founder": { "type": "string" },
       "founded": { "type": "string" }
} } }
...
```

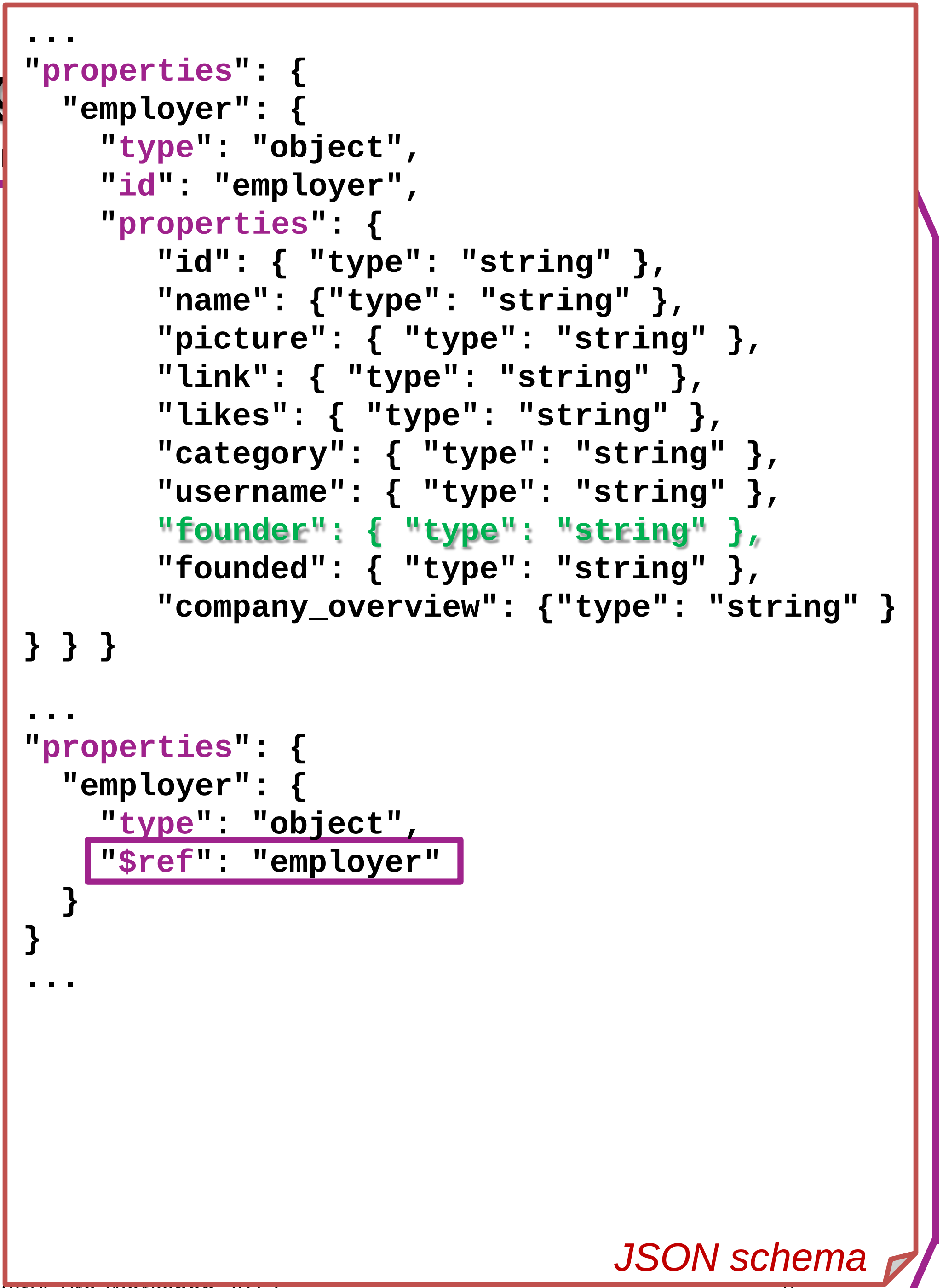
JSON schema

Approach
Phase ❷-2: Merg

- **Merge properties** of equal types
- **Clear duplicates**

Example:

- Merge **multiple employers** to single type
- Clear duplicate employer (**replace with reference**)



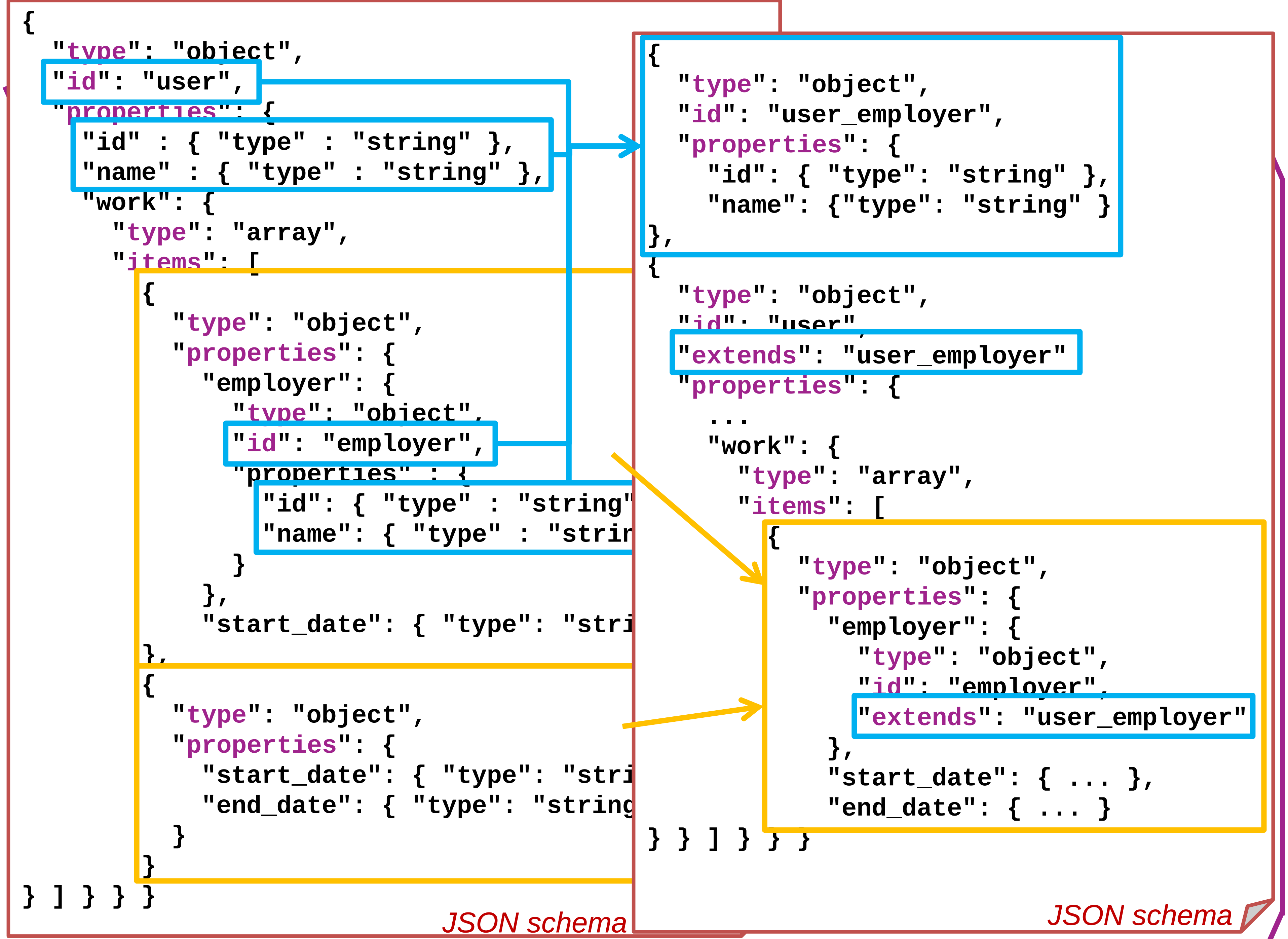
```
...
"properties": {
  "employer": {
    "type": "object",
    "id": "employer",
    "properties": {
       "id": { "type": "string" },
       "name": {"type": "string" },
       "picture": { "type": "string" },
       "link": { "type": "string" },
       "likes": { "type": "string" },
       "category": { "type": "string" },
       "username": { "type": "string" },
       "founder": { "type": "string" },
       "founded": { "type": "string" },
       "company_overview": {"type": "string" }
} } }

...
"properties": {
  "employer": {
    "type": "object",
    "$ref": "employer"
  }
}
...
```

JSON schema

Approach
Phase ❷-3: Refactoring

- Build **class hierarchy**:
  - introduce **superclass** for user and employer
- **Homogenize array** types
  - homogenize **types of work** array



```
{
 "type": "object",
 "id": "user",
 "properties": {
  "id" : { "type" : "string" },
  "name" : { "type" : "string" },
  "work": {
   "type": "array",
   "items": [
    {
     "type": "object",
     "properties": {
      "employer": {
       "type": "object",
       "id": "employer",
       "properties" : {
        "id": { "type" : "string"
        "name": { "type" : "strin
       }
      },
      "start_date": { "type": "stri
    },
    {
     "type": "object",
     "properties": {
      "start_date": { "type": "stri
      "end_date": { "type": "string
     }
    }
} ] } } }
```

JSON schema

```
{
 "type": "object",
 "id": "user_employer",
 "properties": {
  "id": { "type": "string" },
  "name": {"type": "string" }
},
{
 "type": "object",
 "id": "user",
 "extends": "user_employer"
 "properties": {
  ...
  "work": {
   "type": "array",
   "items": [
    {
     "type": "object",
     "properties": {
      "employer": {
       "type": "object",
       "id": "employer",
       "extends": "user_employer"
      },
      "start_date": { ... },
      "end_date": { ... }
} } ] } } }
```

JSON schema

Approach
Phase ❸: Transformation

Transformation to different technical space

- **Mapping** of **meta-models**
- E.g., from **JSON Schema** to **ECORE**

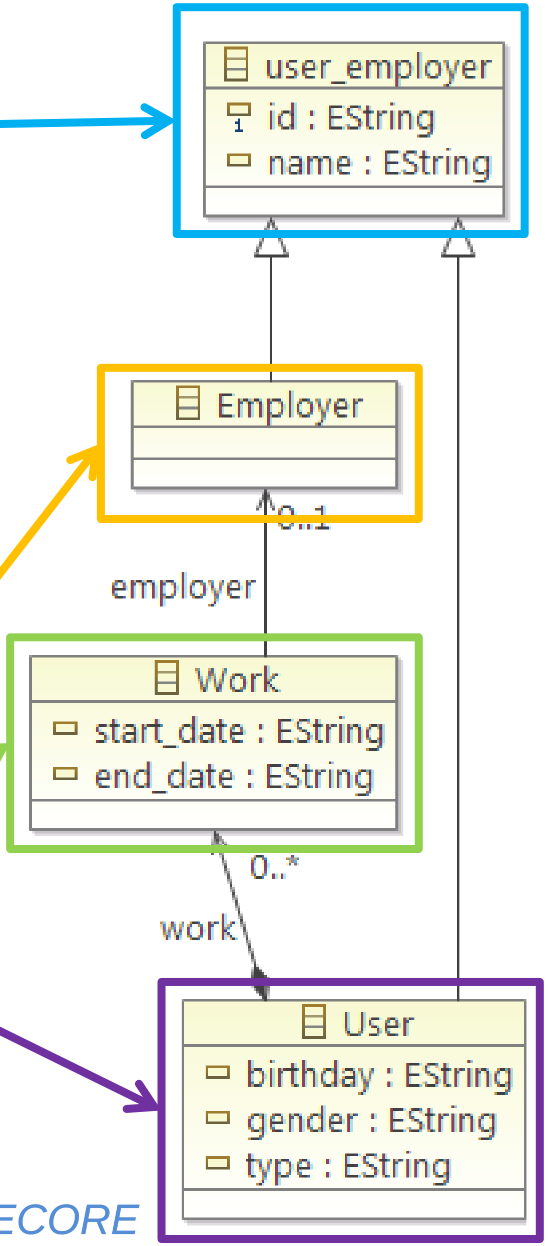
Source concept (JSON)	Target concept (ECORE)
Type	EClass
Primitive property	EAttribute (with corresponding datatype EDataType)
Nested type (without link)	EReference (composition with multiplicity 1)
Nested array (without link)	EReference (composition with unbounded multiplicity)
Link	EReference (reference with maximum multiplicity 1)
Array of links	EReference (reference with unbounded multiplicity)

→ from **ECORE** draw **class diagram**

Approach
Phase ❸: Transformation Example

```
{
  "type": "object",
  "id": "user_employer",
  "properties": {
    "id": { "type": "string" },
    "name": {"type": "string" }
},
{
  "type": "object",
  "id": "user",
  "extends": "user_employer"
  "properties": {
    ...
    "work": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
           "employer": {
             "type": "object",
             "id": "employer",
             "extends": "user_employer"
           },
           "start_date": { "type": "string" },
           "end_date": { "type": "string" }
} } ] } } }
```

JSON schema



ECORE

Results & Evaluation
Test User Profile Setup

Manually created **test users**

- 1 **user + connected friend**
- For **Facebook, Google+, LinkedIn**
- **Equal** properties and activities

Properties

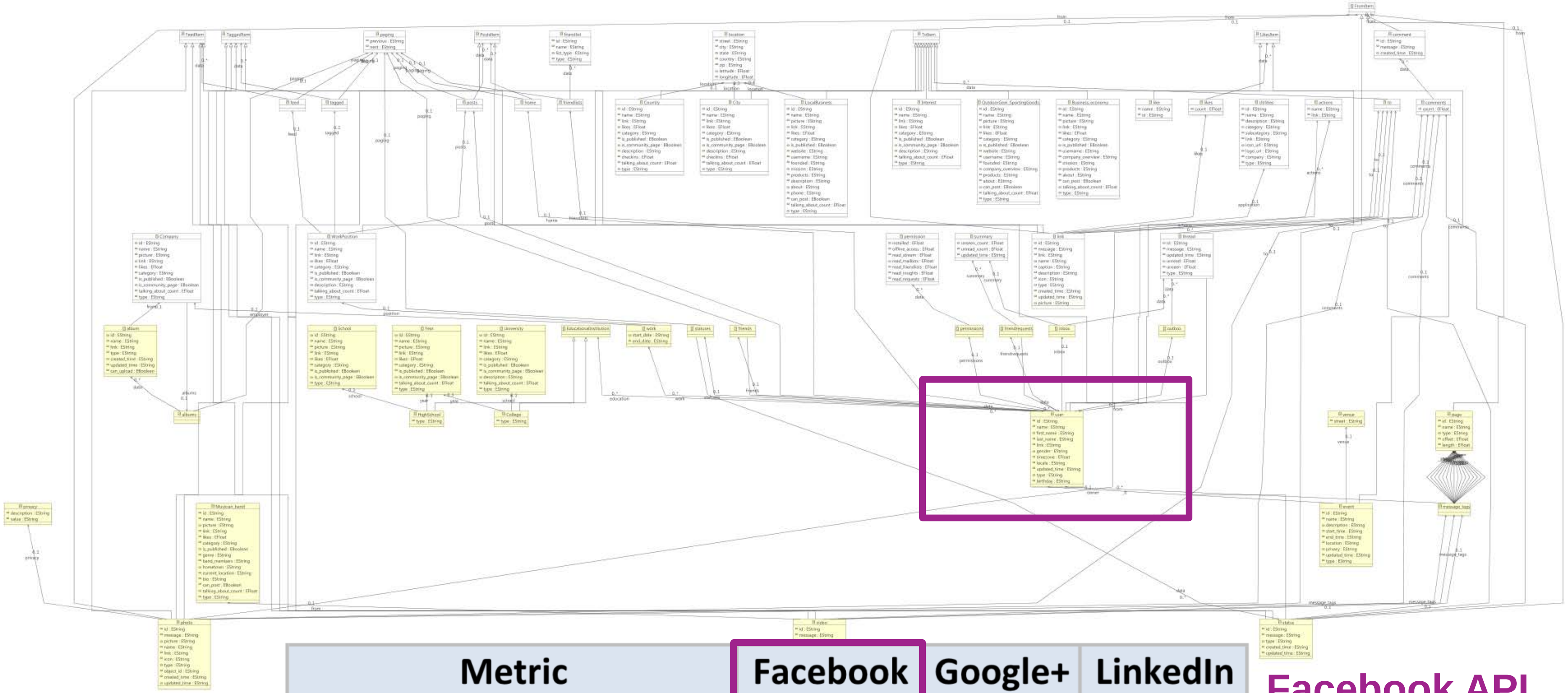
- **Name**
- Email
- City
- Birthday
- Status update
- **Education** and **work**
  - High school & university
  - Current & previous jobs

Activities

- **Connect to friend**
- Direct **communication**
- Group **conversation**
  - Comments
  - Likes
  - Pictures

Results & Evaluation
Extracted Schemas: Facebook

Extracted **schema** for **Facebook** test user profile



Metric	Facebook	Google+	LinkedIn
Number of Classes	58	25	34
Number of Properties	269	71	75
Number of References	93	23	58

Facebook API provides optional meta-information

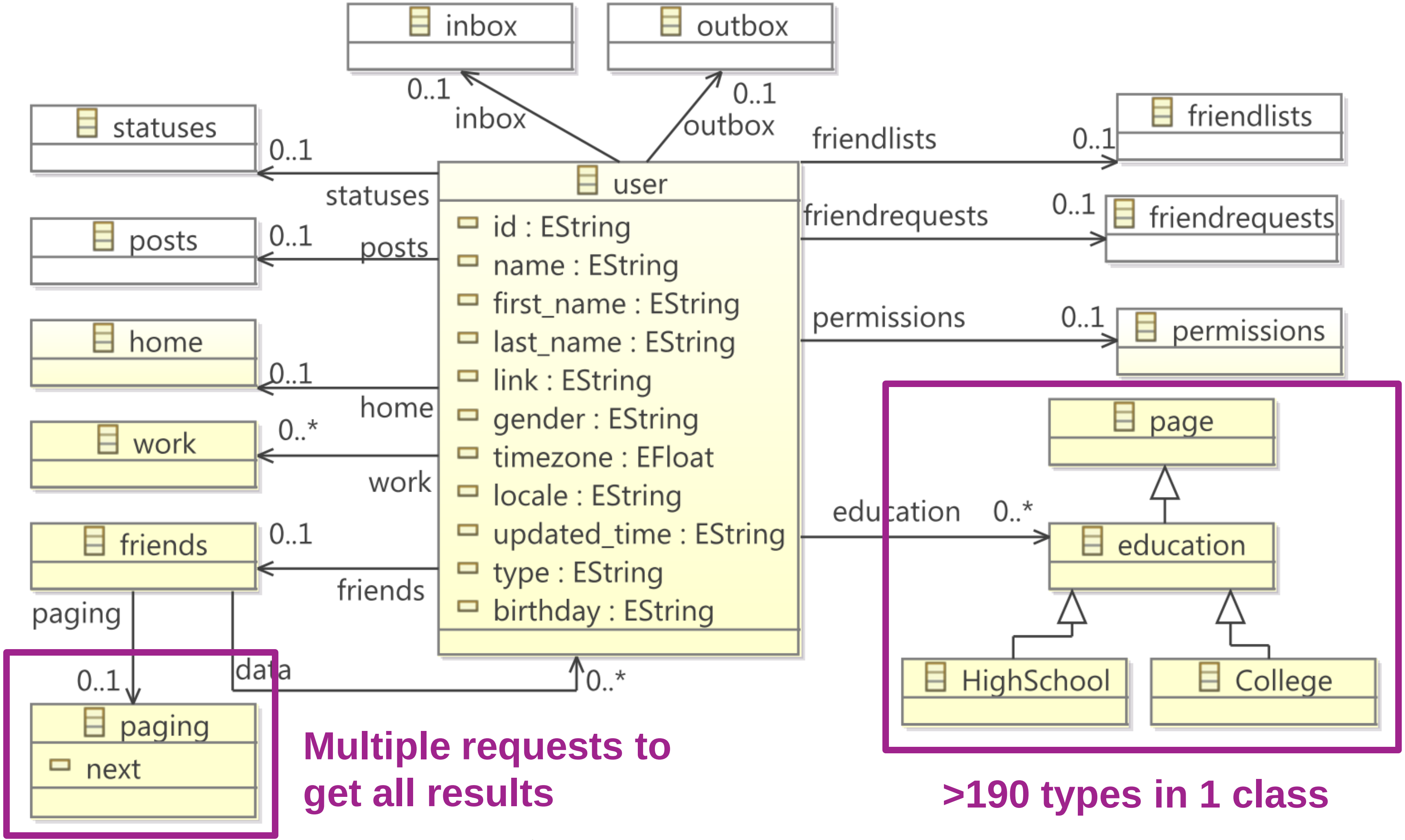
Results & Evaluation
Documentation vs. Extracted Schemas

- Carefully analyzed **documentation** **of User/Person** (properties and references)
- Compared to **extracted data** **from API** (instances & created schema)

	No. of properties & references		
Source	**Facebook**	**Google+**	**LinkedIn**
API documentation of user	71	45	60
Subset expected for test user	24	27	24
Successfully extracted	30	20	29
Intersection of expected & extracted	19	18	23
Documented but missing	**5**	**9**	**1**
Not documented	**11**	**2**	**6**

Results & Evaluation
Extracted Schemas: Facebook

Schema excerpt for Facebook

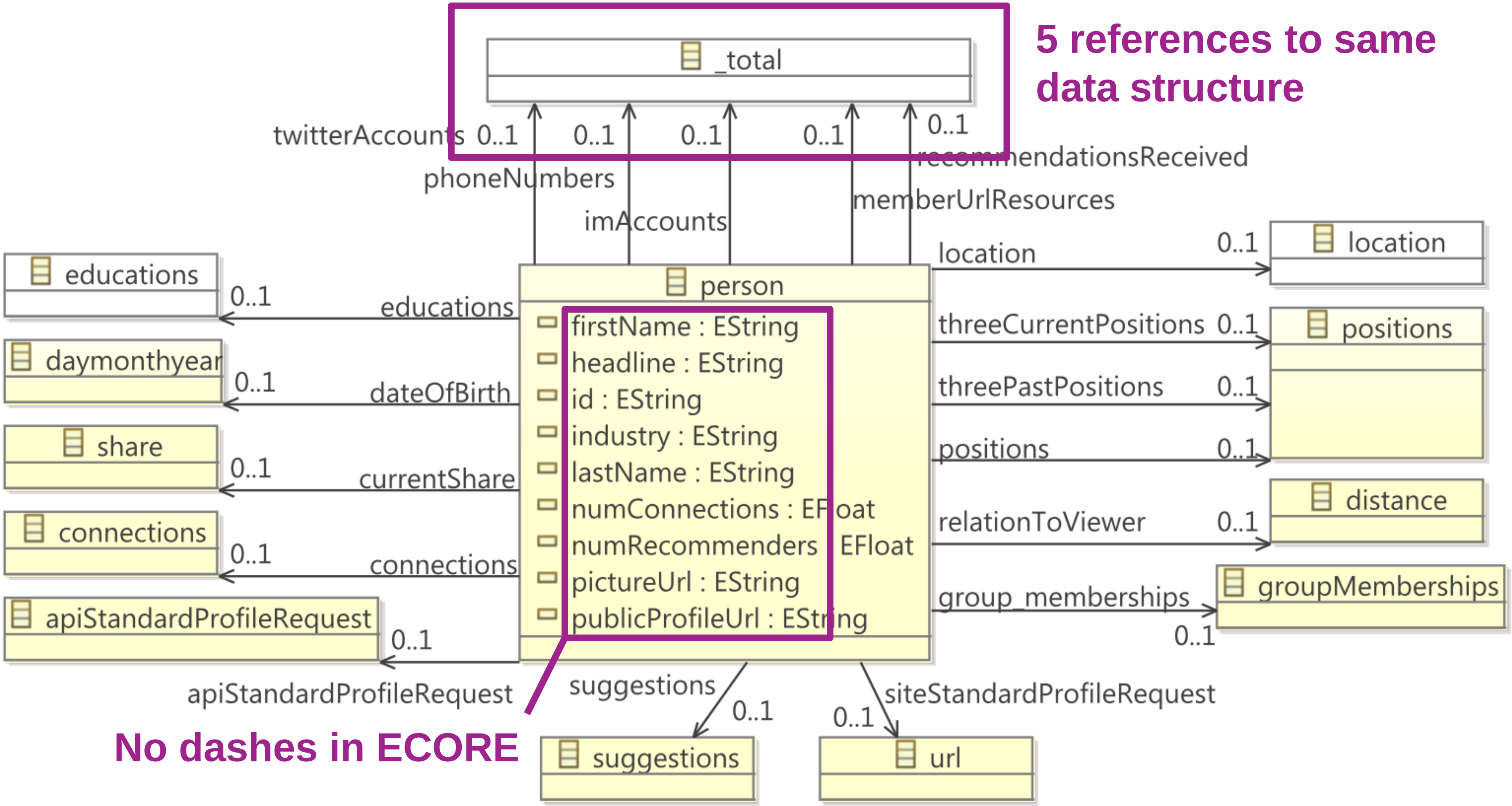


Multiple requests to get all results

>190 types in 1 class

Results & Evaluation
Extracted Schemas: LinkedIn

Schema excerpt for **LinkedIn**

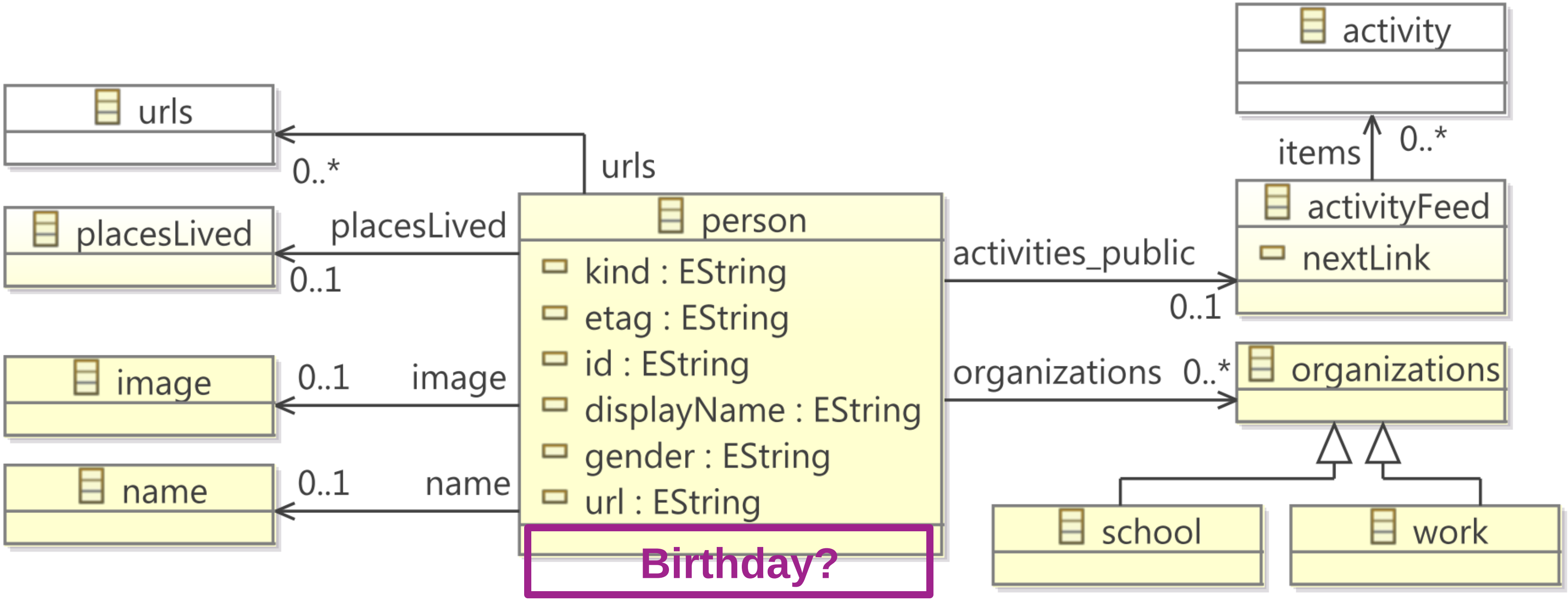


5 references to same data structure

No dashes in ECORE

Results & Evaluation
Extracted Schemas: Google+

Schema excerpt for Google+



APIs do not comply to documentation

Results & Evaluation
Outlook on Integration

- **Compared schemas** of Google+, Facebook, LinkedIn (user and address only)

		Google+	Facebook	LinkedIn
User	username	Person.displayName	User.name	*(not available)*
	firstname	Name.givenName	User.first_name	Person.firstName
	lastname	Name.familyName	User.last_name	Person.lastName
	gender	Person.gender	User.gender	*(not available)*
	date of birth	*(not available)*	User.birthday	Person.dateOfBirth
Address	zip	PlacesLived.value	Location.zip	*(not available)*
	city	PlacesLived.value	Location.city	Location.name
	country	PlacesLived.value	Location.country	Location.country

		Google+ vs. Facebook	Google+ vs. LinkedIn	Facebook vs. LinkedIn
User	username	0.55	*(not applicable)*	*(not applicable)*
	firstname	0.48	0.41	0.73
	lastname	0.[illegible]6	0.39	0.73
	gender	0.74	*(not applicable)*	*(not applicable)*
	date of birth	*(not applicable)*	*(not applicable)*	0.34
Address	zip	*(not applicable)*	*(not applicable)*	*(not applicable)*
	city	0.2	0.33	0.31
	country	0.2	0.17	0.88
Average		**0.33**	**0.16**	**0.37**

- **Structural and semantic heterogeneities**
  - **Missing** information
  - **Naming** differences
  - Fine- and coarse-grained **cardinality** differences
- **Schema matching tool COMA++**
  - First **indicator for similarity** of social network schemas

Conclusions
Lessons Learned

- Approach **depends** on **availability of user data**
  - **Complete** schemas require complete **real/pseudo** profiles
- **Schema extraction** requires **manual intervention**
  - Facebook: **meta-information** to automate process
  - Google+ & LinkedIn: **links** from documentation
- **Different views** on same objects **require schema merging**
  - Merging of **potentially many view classes**
- Differences in **support for query restriction**
  - LinkedIn: **requested information only** (i.e., no "**SELECT ***" possible)
- Nested **objects must have ID to be reusable**
  - Facebook: anonymous **work** elements, pages for **years**
- **Heterogeneous arrays** in JSON Schema
  - **Not representable** in every technical space (e.g., UML)

Conclusions
Future Work & Discussion

- **Transformation of instances**
  - Prerequisite for **integration**
  - Automatically **derive instance transformation rules from schema transformations**
- Deal with **incomplete and unstable interfaces**
  - Create **request code** for data extraction
  - Build **dynamic self-evolving social network adaptors**
- **Co-evolution of extraction and integration applications**
  - Derive **integration rules** from schema mappings
  - **Meta models for change** of schemas and dependent artifacts
    → **automatic co-evolution** of request code and integration rules

Martin Wischenbart
wischenbart@big.tuwien.ac.at

www.social-nexus.net

Thank you!

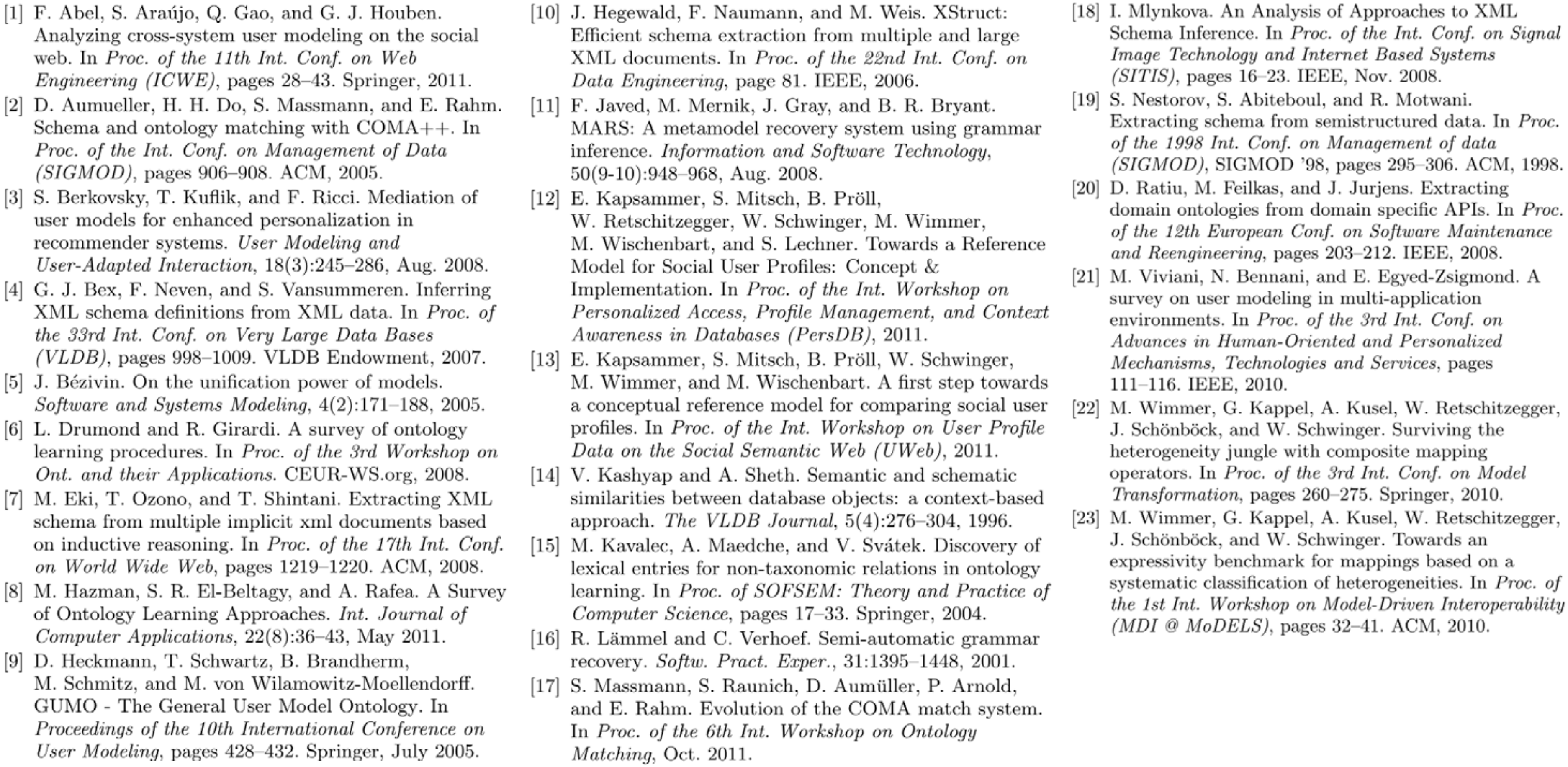
Questions & Comments?

Information Systems Group
Johannes Kepler University Linz
http://www.ifs.uni-linz.ac.at/

Business Informatics Group
Vienna University of Technology
http://www.big.tuwien.ac.at/

Netural GmbH
http://www.netural.com/

References



[1] F. Abel, S. Araújo, Q. Gao, and G. J. Houben. Analyzing cross-system user modeling on the social web. In *Proc. of the 11th Int. Conf. on Web Engineering (ICWE)*, pages 28–43. Springer, 2011.

[2] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *Proc. of the Int. Conf. on Management of Data (SIGMOD)*, pages 906–908. ACM, 2005.

[3] S. Berkovsky, T. Kuflik, and F. Ricci. Mediation of user models for enhanced personalization in recommender systems. *User Modeling and User-Adapted Interaction*, 18(3):245–286, Aug. 2008.

[4] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In *Proc. of the 33rd Int. Conf. on Very Large Data Bases (VLDB)*, pages 998–1009. VLDB Endowment, 2007.

[5] J. Bézivin. On the unification power of models. *Software and Systems Modeling*, 4(2):171–188, 2005.

[6] L. Drumond and R. Girardi. A survey of ontology learning procedures. In *Proc. of the 3rd Workshop on Ont. and their Applications*. CEUR-WS.org, 2008.

[7] M. Eki, T. Ozono, and T. Shintani. Extracting XML schema from multiple implicit xml documents based on inductive reasoning. In *Proc. of the 17th Int. Conf. on World Wide Web*, pages 1219–1220. ACM, 2008.

[8] M. Hazman, S. R. El-Beltagy, and A. Rafea. A Survey of Ontology Learning Approaches. *Int. Journal of Computer Applications*, 22(8):36–43, May 2011.

[9] D. Heckmann, T. Schwartz, B. Brandherm, M. Schmitz, and M. von Wilamowitz-Moellendorff. GUMO - The General User Model Ontology. In *Proceedings of the 10th International Conference on User Modeling*, pages 428–432. Springer, July 2005.

[10] J. Hegewald, F. Naumann, and M. Weis. XStruct: Efficient schema extraction from multiple and large XML documents. In *Proc. of the 22nd Int. Conf. on Data Engineering*, page 81. IEEE, 2006.

[11] F. Javed, M. Mernik, J. Gray, and B. R. Bryant. MARS: A metamodel recovery system using grammar inference. *Information and Software Technology*, 50(9-10):948–968, Aug. 2008.

[12] E. Kapsammer, S. Mitsch, B. Pröll, W. Retschitzegger, W. Schwinger, M. Wimmer, M. Wischenbart, and S. Lechner. Towards a Reference Model for Social User Profiles: Concept & Implementation. In *Proc. of the Int. Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB)*, 2011.

[13] E. Kapsammer, S. Mitsch, B. Pröll, W. Schwinger, M. Wimmer, and M. Wischenbart. A first step towards a conceptual reference model for comparing social user profiles. In *Proc. of the Int. Workshop on User Profile Data on the Social Semantic Web (UWeb)*, 2011.

[14] V. Kashyap and A. Sheth. Semantic and schematic similarities between database objects: a context-based approach. *The VLDB Journal*, 5(4):276–304, 1996.

[15] M. Kavalec, A. Maedche, and V. Svátek. Discovery of lexical entries for non-taxonomic relations in ontology learning. In *Proc. of SOFSEM: Theory and Practice of Computer Science*, pages 17–33. Springer, 2004.

[16] R. Lämmel and C. Verhoef. Semi-automatic grammar recovery. *Softw. Pract. Exper.*, 31:1395–1448, 2001.

[17] S. Massmann, S. Raunich, D. Aumüller, P. Arnold, and E. Rahm. Evolution of the COMA match system. In *Proc. of the 6th Int. Workshop on Ontology Matching*, Oct. 2011.

[18] I. Mlynkova. An Analysis of Approaches to XML Schema Inference. In *Proc. of the Int. Conf. on Signal Image Technology and Internet Based Systems (SITIS)*, pages 16–23. IEEE, Nov. 2008.

[19] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proc. of the 1998 Int. Conf. on Management of data (SIGMOD)*, SIGMOD '98, pages 295–306. ACM, 1998.

[20] D. Ratiu, M. Feilkas, and J. Jurjens. Extracting domain ontologies from domain specific APIs. In *Proc. of the 12th European Conf. on Software Maintenance and Reengineering*, pages 203–212. IEEE, 2008.

[21] M. Viviani, N. Bennani, and E. Egyed-Zsigmond. A survey on user modeling in multi-application environments. In *Proc. of the 3rd Int. Conf. on Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services*, pages 111–116. IEEE, 2010.

[22] M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. Surviving the heterogeneity jungle with composite mapping operators. In *Proc. of the 3rd Int. Conf. on Model Transformation*, pages 260–275. Springer, 2010.

[23] M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. Towards an expressivity benchmark for mappings based on a systematic classification of heterogeneities. In *Proc. of the 1st Int. Workshop on Model-Driven Interoperability (MDI @ MoDELS)*, pages 32–41. ACM, 2010.