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

Elisabeth Kapsammer, Angelika Kusel, Stefan Mitsch, Birgit Pröll, Werner Retschitzegger, Wieland Schwinger
Johannes Kepler University
Information Systems Group
Altenbergerstrasse 69
4040 Linz, Austria
{firstname.lastname}@jku.at

Johannes Schönböck, Manuel Wimmer, Martin Wischenbart
Vienna University of Technology
Business Informatics Group
Favoritenstrasse 9-11
1040 Vienna, Austria
{lastname}@big.tuwien.ac.at

Stephan Lechner
Netural GmbH
Europaplatz 4
4020 Linz, Austria
s.lechner@netural.com

ABSTRACT

User profile integration from multiple social networks is indispensable for gaining a comprehensive view on users. Although current social networks provide access to user profile data via dedicated APIs, they fail to provide accurate schema information, which aggravates the integration of user profiles, and not least the adaptation of applications in the face of schema evolution. To alleviate these problems, this paper presents, firstly, a semi-automatic approach to *extract schema information* from instance data. Secondly, *transformations* of the derived schemas to different technical spaces are utilized, thereby allowing, amongst other benefits, the application of established integration tools and methods. Finally, as a case study, schemas are derived for Facebook, Google+, and LinkedIn. The resulting schemas are analyzed (i) for completeness and correctness according to the documentation, and (ii) for semantic overlaps and heterogeneities amongst each other, building the basis for future user profile integration.

Categories and Subject Descriptors

D.2.12 [**Software Engineering**]: Interoperability—*schema extraction, model transformation*

Keywords

Schema extraction, model transformation, social network data integration, model driven approach, social networks, JSON Schema

1. INTRODUCTION

In recent years, online social networks have gained great popularity amongst internet users. These networks serve different purposes and communities, for instance, socializing on Facebook or Google+, or establishing professional networks in LinkedIn[1]. Their popularity led to a huge amount of collected data, and, hence, attracts the interest for exploitation by commercial and non-commercial applications (e. g., recommender applications, such as TripAdvisor[2]). Since often users are members of several social networks, integrated profiles from multiple networks are desired to achieve a *comprehensive view* on users, which would, for instance, increase the quality of personalized recommendations [1]. For this, various user modeling approaches have been proposed (e. g., standardization-based ones, such as GUMO [9], or mediation-based ones [3]), as revealed by a recent survey [21].

However, up to now the systematic integration of the gathered data is hampered, because the underlying data stores of social networks are built with a focus on extension and flexibility, and thus, they often use so-called NOSQL databases. Such databases may store data in large tables *without a traditional schema* (e. g., HBase in Hadoop[3], used by Facebook and LinkedIn), or in *schema-less* multidimensional maps (e. g., Cassandra[4], used by Twitter). The resulting absence of explicit schema descriptions not only prevents the application of integration tools (e. g., COMA++ [17] for schema matching, or MapForce[5] for schema mapping), but also hardens various data processing tasks, such as search, manipulation, optimization, translation, or evolution. As a consequence of the fast development pace of today's online social network platforms, and due to API documentations often being outdated and only exemplary, however, manual creation of explicit and up-to-date schemas is not an adequate option, not least since they may be extensive in size.

This paper tackles these challenges by means of a semi-automatic approach to derive social network schemas from social network data. Since JSON (JavaScript Object Notation[6]) is the leading format for data interchange supported by many APIs of social networks, we derive schema information expressed in the JSON Schema[6] language. For this,

[1]`www.facebook.com`, `plus.google.com`, `www.linkedin.com`
[2]`www.tripadvisor.com`
[3]`hbase.apache.org`, `hadoop.apache.org`
[4]`cassandra.apache.org`
[5]`www.altova.com/mapforce`
[6]`www.json.org`, `json-schema.org`

WWW 2012 Companion, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1230-1/12/04.

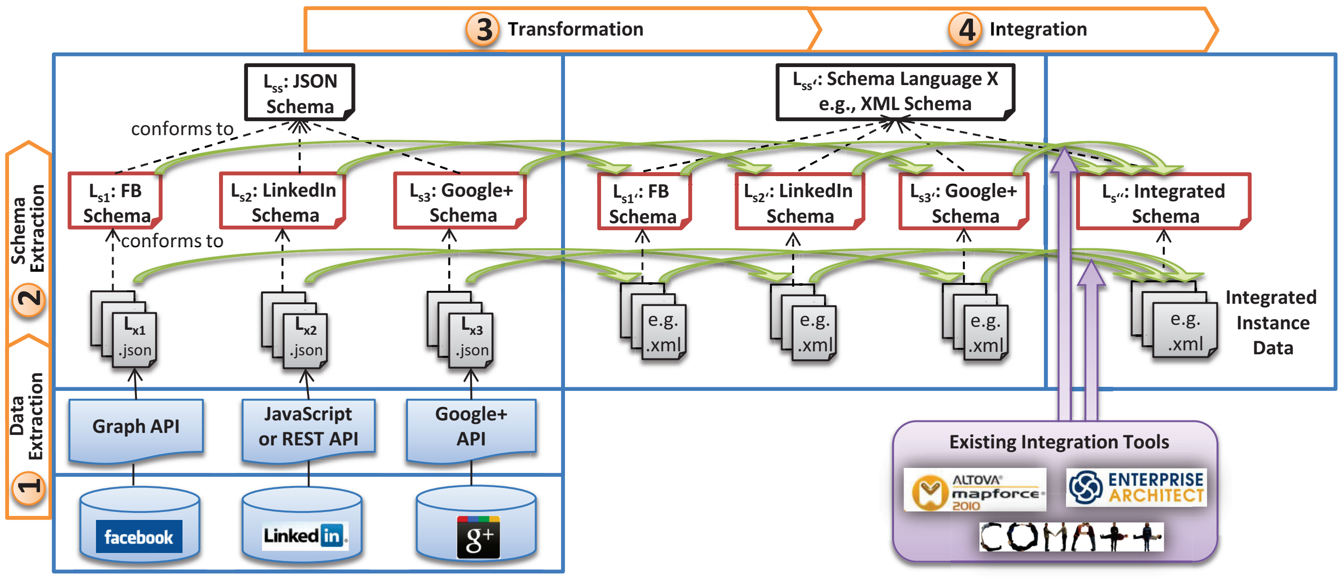


Figure 1: Overview of the extraction, transformation, and integration process

dedicated schema extraction strategies are provided. Since state-of-the-art integration tools build on different technical spaces (e. g., MapForce and COMA++ handle XML Schema), we propose to apply well-known techniques from the domain of *model-driven engineering* [5, 22] to *transform* schemas as well as instances. Finally, as a case study, schemas are extracted and transformed from three of the most popular social networks according to the Alexa[7] ranking, namely, Facebook, Google+, and LinkedIn. Subsequently, the resulting schemas are *analyzed*, to study (i) completeness and correctness of schemas with respect to the available online documentation of social networks, and (ii) *schematic overlaps* as well as *syntactic and semantic heterogeneities* [14] that need to be considered for integration.

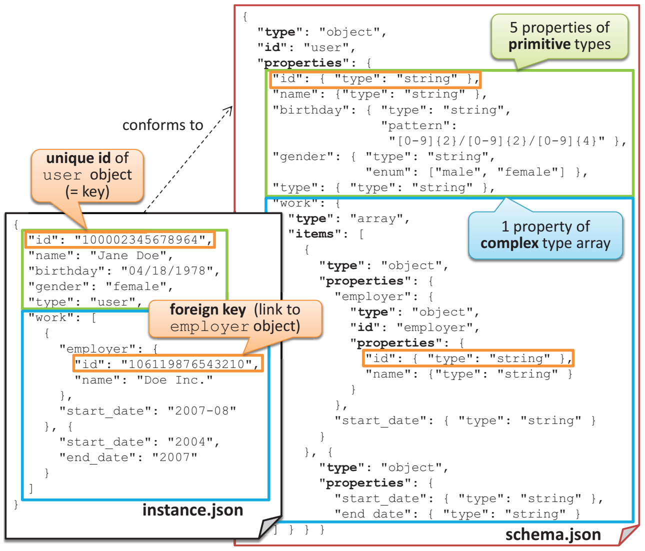
Outline. Section 2 gives an architectural overview of our approach, before Section 3 discusses related work. In Section 4 various *schema extraction strategies* to be applied for social networks are discussed. Furthermore, a transformation to ECORE[8], which builds the basis for presenting the extracted schemas in terms of class diagrams and their analysis is given in Section 5. Finally, Section 6 critically discusses the approach and provides directions of future work.

2. ARCHITECTURAL OVERVIEW

To integrate user profiles from different social networks, the proposed process (cf. Fig. 1) consists of four major phases: (i) data extraction, (ii) schema extraction, (iii) transformation, and (iv) integration, as discussed in detail below.

2.1 Data Extraction

In the *data extraction phase* (cf. ① in Fig. 1), instance data is extracted from social networks through their corresponding APIs. These extracted data fragments are each expressed in a generic markup language L_x, i. e., JSON in the case of most social networks. A short JSON example from a Facebook user is shown on the left hand side of Fig. 2. The object of type `user` (as indicated by property `type`) provides information about the user's *name* (string value), *birthday* (string, possibly following a particular *pattern*), *gender* (string, possibly with *restrictions* concerning allowed values), and *work experience* (*array* of objects). Note that, in order to enable fine-grained access to user profile information guarded with authorization techniques, APIs require multiple requests to retrieve complementing data fragments (e. g., an additional request using the employer ID would be needed to retrieve detailed employer information, which was not provided within the original response).


conforms to
unique id of user object (= key)
foreign key (link to employer object)
5 properties of primitive types
1 property of complex type array
instance.json
schema.json

```
{
  "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"
    }
  ]
}
```

```
{
  "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" }
] } } }
```

Figure 2: JSON data and extracted JSON Schema

[7]`www.alexa.com/topsites`
[8]ECORE is the Eclipse realization of MOF - `www.eclipse.org/modeling/emf`

2.2 Schema Extraction

In the *schema extraction phase* (cf. ② in Fig. 1, details in Sect. 4.1), a so far unknown schema L_s should be inferred, which in turn conforms to another schema L_{ss}. For instance, in case of JSON data (L_x = JSON), the concrete social network schema (L_s) should be inferred, which conforms to JSON Schema (L_{ss}) (cf. right hand side of Fig. 2). In this respect, schema extraction must find a mapping $L_x \rightarrow L_s$ from fragments expressed in the generic markup language L_x to an according schema L_s, also known as *inductive reasoning*. Since the extracted JSON data carries structural information, such as the names of properties and their datatypes, schema information may be automatically derived, as can be exemplarily seen in Fig. 2, and detailed below.

Generalization Strategies. Schema extraction has to cope with the challenge of going from specifics to *generalizations*. For this challenge, positive data examples for the source language L_x, i. e., examples that correspond to the schema to be extracted, may be exploited, because the data retrieved via the API are assumed to be valid instances. In order to prevent overfitting of the to be extracted schema L_s, i. e., the schema would only fit to a very limited amount of examples expressed in L_x, the extraction needs to be repeated with multiple diverse data samples, each comprising possibly complementing data fragments (e. g., two different employment types have overlapping but distinct properties).

Merging and Clearance. Therefore, at first, multiple schemas L_s^1 to L_s^n (subsumed by $\vec{L_s^i}$) for each social network are created (e. g., schemas L_{s1}^1 to L_{s1}^n for Facebook L_{s1}). These extracted schemas $\vec{L_s^i}$ can then be merged to construct a consistent schema L_s ($\vec{L_s^i} \rightarrow L_s$). In the course of this, schema clearance, such as *additions* (e. g., additional properties) and *modifications* of previously extracted schemas (e. g., change of cardinality) must be performed.

In order to account for the design characteristics of social network APIs (e. g., references in Facebook are specified via `id` properties) prior knowledge about L_s may be utilized to configure generalization and merging strategies. Such prior knowledge, for instance, may be retrieved from documentation, naming conventions, or by manual investigation of examples. Especially, *linking data fragments from multiple requests* requires additional knowledge, as the implementation of links differs between the APIs of social networks.

Schema Refactoring. After having extracted a first version of the schema from various instances, postprocessing steps are needed to improve the resulting schemas. Such improvements include the introduction of an inheritance hierarchy or the specialization of general types (e. g., strings) to more specific ones (e. g., enumerations).

2.3 Transformation

In order to reuse existing integration and modeling tools, or otherwise exploit social network schemas and data, in the *transformation phase* (cf. ③ in Fig. 1, details in Sect. 4.2) the extracted schemas as well as the corresponding instance data have to be transformed to possibly many different technical spaces. For automatically transforming the schemas, a transformation $L_{ss} \rightarrow L_{ss'}$ has to be specified, for instance, from JSON Schema to XML Schema or ECORE, which allows to transform the corresponding schemas, i. e., L_s is transformed to $L_{s'}$. For transforming instances, according transformations between pairs of L_s and $L_{s'}$ are needed. For example, to automatically transform JSON data into XML documents, a transformation specification between the Facebook JSON Schema and a Facebook XML Schema is required. So far these transformations must be specified manually, but are envisioned to be derived automatically from $L_{ss} \rightarrow L_{ss'}$.

2.4 Integration

To create an integrated user profile from the user profiles of multiple social networks, in the *integration phase* (cf. ④ in Fig. 1), we then resort to existing integration processes, which are supported by general purpose modeling tools, such as Enterprise Architect[9], and by a multitude of dedicated integration tools, such as COMA++[17] for similarity matching, or MapForce[10] for mapping.

In the following, the focus of this paper is on the schema extraction and transformation phase, both being crucial prerequisites for user profile integration.

3. RELATED WORK

This section discusses related work on schema extraction in the closely related field of *data engineering*, and in the more broadly related areas of *model* and *ontology engineering*.

3.1 Data Engineering

Although JSON Schema for describing the structure of JSON documents has been proposed and published as draft by the Internet Engineering Task Force (IETF)[11], to the best of our knowledge, prior research did not focus on automated extraction of JSON Schemas from JSON documents. So far, only approaches that derive other artifacts from JSON documents exist, for instance, JSONGen[12] enables on-the-fly generation of Java classes.

Several studies have focused on extraction of XML Schemas and DTDs from XML documents [4, 7, 10, 18]. The XStruct algorithm [10], for example, is capable of processing multiple instance documents to produce a single comprehensive schema. Heuristics to cluster similar entities can be used to reduce the number of classes in the inferred schema [19], and also an inductive logic programming approach was proposed to infer the structure of XML documents [7]. In our approach, we pursue a similar *homogeneous* schema extraction approach (i. e., schema extraction within the same technical space) for JSON data, which differs, however, in several aspects, such as configurability to (social network) APIs.

As an alternative, one could employ *heterogeneous* schema extractions by translating JSON documents to XML first, for which dedicated tools exists. However, these tools do not allow to consider social network peculiarities by configuration, resulting in several shortcomings. To exemplify these shortcomings, XMLSpy[13] as a prominent representation thereof has been applied to the sample JSON instance shown in Fig. 2. As result, first, XML schemas can be extracted from a single XML document only, resulting in *high specificity* of the extracted schema (e. g., all elements are marked mandatory, although they may not be present in all instance documents). Second, the extraction process can only be con-

[9] www.enterprisearchitect.at
[10] www.altova.com/mapforce
[11] tools.ietf.org/html/draft-zyp-json-schema-03
[12] jsongen.byingtondesign.com
[13] www.altova.com/xmlspy

figured in a generic manner (e. g., to decide whether type definitions of elements should be kept local or global), but does not allow one to consider specifics of social networks, such as the linking of JSON instances by means of keys and foreign keys as shown in Fig. 2.

3.2 Model and Ontology Engineering

As more broadly related work we additionally investigated methods from model and ontology engineering. In recent years, approaches from model engineering to extract metamodels from models have been proposed. Javed et al. [11] base on extraction techniques for programming language grammars from source code (e. g., [16]). They propose automatic translation of model examples into a context free grammar format, which then is fed into grammar extraction tools as input. Finally, a transformation from the grammar to an according metamodel is provided. Although this approach would be theoretically applicable in our scenario, specifics of social networks as well as their rapidly changing schema would lead to numerous different transformations and corresponding context free grammars. Consequently, our approach is more flexible and reusable, since it allows easy configuration and application of extraction strategies.

Research on schema extraction in the field of ontology engineering is commonly subsumed under the term *ontology learning* [6, 8]. As revealed by a recent survey [8], most of todays ontology learning approaches focus on the extraction of *concepts* and their *taxonomic relationships* only. Finding non-taxonomic relations (i. e., references between classes) and concept properties are the least considered problems [15], whereby only a first approach for ontology extraction from domain APIs is described in [20]. In social network schema extraction, we go beyond these approaches by providing heuristics for analyzing the links between JSON documents to derive non-taxonomic relations between concepts, as well as concept properties.

4. EXTRACTION & TRANSFORMATION

To overcome the identified shortcomings of existing approaches, this section first introduces a schema extraction process to automatically derive a social network schema expressed in JSON Schema from previously acquired positive JSON examples, like the ones given in Fig. 2. Second, a transformation of the extracted schemas to conceptual models is presented to enable the application of state-of-the-art integration tools.

4.1 Schema Extraction

Several sub-steps are proposed to *refine the schema extraction phase*, as shown in Fig. 3. These sub-steps may be configured, in order to tailor the schema extraction phase for a single social network or application. The steps include the application of *generalization* strategies, resulting in one schema part per JSON object. As mentioned before, the derived schema parts often contain links to other JSON objects (e. g., a user contains links to his/her current and past employments), for which according schema information must be derived to gain a more comprehensive schema. After having derived the schema parts, they need to be *merged* to one single coherent schema, i. e., links between schemas are replaced by the according schema contents. Furthermore, assuming that more than one user profile has been available for extraction to prevent overfitting (cf. `furtherProfilesRequested`), the merging step must also incorporate the merging of several derived schemas. Subsequently, redundancies caused by merging are eliminated in a *clearance* step. Finally, *refactorings* are applied to improve the schema, for instance, by introducing inheritance hierarchies.

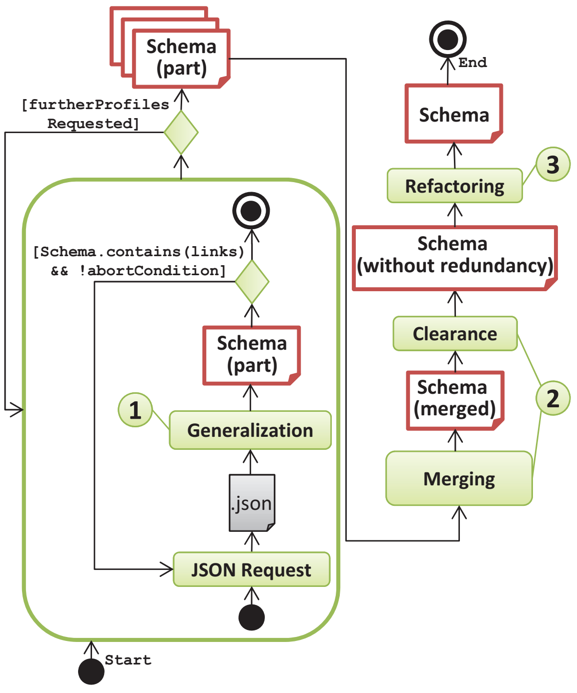


Figure 3: Steps of the schema extraction phase

4.1.1 Generalization Strategies

The task of the generalization phase is (i) to extract JSON schema parts starting from a JSON instance obtained from a single request (entry point of process), and (ii) to introduce schema links between the resulting part and other JSON schema parts that were extracted by traversing the links in JSON instances (which may be obtained until a certain predefined depth of requests is achieved, cf. `abortCondition` in Fig. 3). Consequently, the generalization strategies performing these tasks may be divided into, firstly, those which allow to extract JSON schemas from JSON instances (cf. *type extraction strategies* and *property extraction strategies* in Table 1), and secondly, those which allow to introduce links referring to further JSON instances (cf. *link extraction strategies* in Table 1).

To actually derive a schema from JSON instances, for each object a *type* and for each key/value pair a corresponding *property* is created.

Type Extraction. The main challenge in type extraction arises in deriving a *unique name* for the extracted type. For this, three strategies are proposed. First, the value of a key/value pair may be employed, if it contains such information (cf. *IdFromValue* in Table 1). For example, in Facebook `"type":"user"` may be used to identify the corresponding type. To apply this strategy, the extraction algorithm must be configured with those keys, whose values may potentially carry type information in a specific social network (cf. configuration options in Table 1). Second, if an object is nested, the name of the link to the nested object may be employed

Table 1: Generalization strategies

	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

as type name (cf. *IdFromReferenceName* in Table 1). For example, in Facebook the `work` property of a user contains only anonymous objects with no type information; hence, the key "work" of the key/value pair may be used as name for the types generated for these objects. Finally, if none of the above strategies is applicable, the name may be derived by concatenating the names of the keys contained in the object (cf. *IdFromNameConcat* in Table 1).

Property Extraction. The main challenge in property extraction is the derivation of *specific types*. Although JSON instances allow the distinction between the primitive types of strings, booleans, and numbers and the complex types of arrays and objects (cf. *TypeFromValue* in Table 1), more specific types may be desired, such as enumerations. Therefore, the specific strategies *EnumFromValue* and *IntervalFromValue* are offered, which allow to add restrictions to strings and numbers. For this, the names of the keys, for which enumerations should be generated, must be provided by the user (cf. configuration options in Table 1).

Link Introduction. Concerning the introduction of links, two main challenges arise: first, a *role name* of a link must be found (in JSON schema, the role name is represented by the property `rel`, with predefined values `self`, `full`, `describedBy`, and `root` that can be complemented with custom ones). Second, an actual *reference* (`href`) to another JSON document must be defined. Therefore, the name of a property, whose value is a valid URL may be employed as the role name of a link (*LinkRoleFromName*). Alternatively, a user may select to introduce a link with one of the pre-defined role names (or a custom one) whenever a property with a particular name is encountered. For example, in Facebook every property `id` represents a link to the *full* representation of the enclosing object. To apply this strategy, the extraction process may be configured with the role names and those keys that potentially represent such a link. Second, the value of a URL property may be used as the actual reference (*LinkPatternFromValue*). In case that during application of the first strategy a key/value pair was specified, whose values are not URLs, an additional URL pattern must be provided. This pattern turns a value into a valid URL. For example, in Facebook the above-mentioned convention that IDs point to full representations of an object (e. g., as employed for the user's `hometown`) can be exploited to issue an additional request to `graph.facebook.com/{id}`, where `{id}` is replaced with the value of the property `id`.

4.1.2 Merging and Clearance

Merging. In order to produce a single coherent schema, in the merging phase, starting at the root schema of each user profile, *links* must be resolved recursively. Therefore, links between schemas are replaced by the according schema contents, i. e., a merging $\vec{L_s^i} \rightarrow L_s$ is performed. After having performed link resolving and schema merging for each JSON object, the resulting schema L_s may obviously contain duplicate types.

Clearance. In the clearance phase, hence, all but the first of these duplicate types are removed from the schema, and the respective properties are defined with a reference to the remaining sole type. Additional properties of the removed duplicate types are copied into the remaining type, leading to more comprehensive schemas.

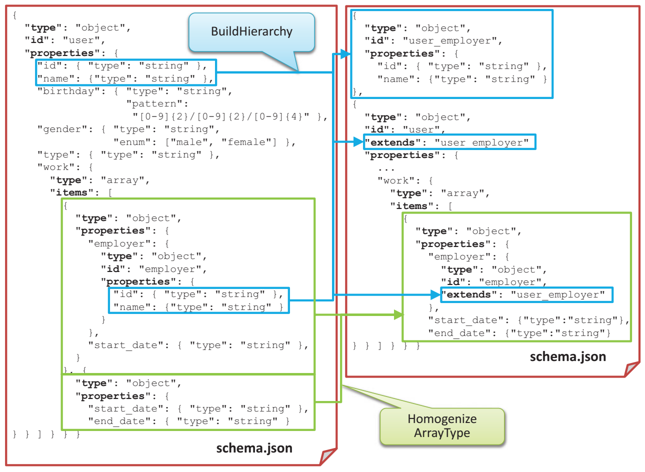
4.1.3 Schema Refactoring

After completing the merging and clearance phase, refactorings may be applied on the schema to improve its quality. In the following paragraphs, potential refactorings to be applied in our case study are discussed in detail.

BuildHierarchy. Types with different names and overlapping properties (i. e., types having non-empty intersections of property names) can be structured into an inheritance hierarchy, as can be seen in Fig. 4. Thereby, this refactoring allows to specify the minimal number of intersecting properties in order to avoid the introduction of unreasonably fine-grained inheritance hierarchies.

LookupOntology. Top-level and domain specific ontologies as external sources of knowledge may be employed to improve the quality of the generated JSON schema. For instance, commonly agreed type names could be inferred by searching ontologies for types that have considerable overlap of properties with the types in the extracted schema. Also, ontologies may be used in combination with the strategy *BuildHierarchy* in order to infer accepted taxonomies.

HomogenizeArrayTypes. In JSON, arrays may contain objects of different types. In order to respect the common assumption that arrays contain only a single type (i. e., all elements in the array are of equal type or assignable to a shared base type), this refactoring homogenizes the types of elements contained in the same array. Two different approaches to homogenizing types are considered: (i) a new class containing the union of attributes is defined as sole element in the `items` schema element of an array, or (ii) alternatively, a class hierarchy can be inferred, with a base class containing the intersection of properties, and subclasses extending this base class with additional properties.



```
{
  "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" }
} } ] } } }
```

```
{
  "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"}
} } ] } } }
```

Figure 4: Sample refactoring

MergeReferenceIntoSource. In order to reduce the number of classes, the properties of a referenced class may be merged into the respective source class. For example, in Facebook partly filled objects with a property `id` result in schemas that have a dedicated partial type referring to a full type using a reference named `full`. While such a schema reveals the actual sequence of requests and the structure of the respective responses and, hence, is particularly helpful to organize the requests of an application, in some cases this information might be unneeded. Then, the full type may replace the partial type and the reference can be removed.

Assuming that the schema has been derived from multiple user profiles, additional refactorings may be employed to improve the quality, as discussed below.

UniteConstraints. Since datatype restrictions are difficult to detect from single JSON objects, heuristics may be applied to detect such limitations from a set of objects (i. e., from the respective types extracted for each object in isolation). For instance, the user property `gender` may be constrained with an enumeration `"enum": [ "female", "male" ]`, since every example only contains either value. In order to ensure that all positive examples remain valid with respect to the extracted constraints of properties (enumeration or value intervals), *unions* of enumerations and minimum/maximum intervals are built when merging the extracted schemas of several extraction runs.

ExtractPattern. From multiple schemas it is also possible to detect complex patterns described by regular expressions, such as that various dates in Facebook correspond to `MM/DD/YYYY`, in contrast to JSON's `date` (`YYYY-MM-DD`).

DefineMandatoryProperty. Those properties, which are present in multiple positive examples (i. e., values are set in every instance), can be assumed to be mandatory, and, hence, they can be set to `required` in the JSON schema.

4.2 Transformation

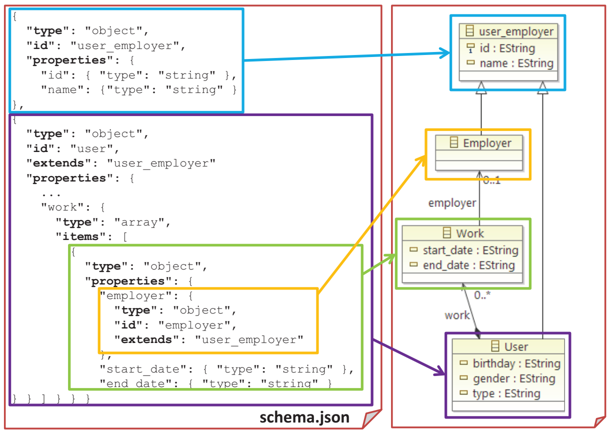
After having extracted a social network schema L_s, standard *model transformation* techniques known from the area of model-driven engineering [5] may be employed to bridge gaps between different technical spaces, i. e., we have to specify transformations of the form $L_{ss} \rightarrow L_{ss'}$. In our context,

Table 2: Transformation from JSON 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)

a transformation from JSON Schema to ECORE has been specified in order to transform a concrete social network schema to an according class diagram, as shown in Table 2. Based on this schema representation in ECORE, various other transformations may be specified, for instance, to build an XML Schema, or a schema description in an OWL ontology, or to create Java class files (POJOs).

The transformation from JSON Schema to ECORE has been specified in Xtend[14], one of the most prominent available transformation languages. For the JSON Schema after refactoring introduced in Fig. 4 above, the transformation leads to the class diagram depicted in Fig. 5.



```
{
  "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" }
} } ] } } }
```

Figure 5: Sample transformation

5. CASE STUDY & EVALUATION

The resulting schemas of the schema extraction and transformation process build the basis for the envisioned user profile integration. As a first step in this direction, in this section we employ a case study of schema extraction and transformation from user profiles of three major social networks—namely Facebook, Google+, and LinkedIn—to determine completeness and correctness with respect to the available online documentation. Second, we identify semantic overlaps, as well as structural and semantic heterogeneities between these social network schemas and, from these, derive lessons learned for user profile integration.

[14]`www.eclipse.org/xtend`

]

5.1 Case Study Setup

In order to obtain comparable results, two connected test user profiles were created in each of the selected social networks. Both profiles provide information about name, city, birthday, and a profile picture. Additionally, education and work information was entered: high school, university, and two jobs. Finally, the test users were connected as friends for exchanging messages, also with attached images (images not supported in LinkedIn), both, directly and within a group (shared circle in Google+). On the basis of these profiles, the schema extraction process was configured and applied to each of the selected social networks separately, as detailed below. As a result, three distinct schemas were extracted, reflecting that partial user profiles, but not the complete information available in each of the social networks. Nonetheless, as these schemas were created in a consistent manner, they are suited for a first evaluation.

Facebook. During Facebook schema extraction, type IDs were obtained using the strategy *IDFromValue* on the properties `category` and `type`, and as a fallback, using *IdFromReferenceName*. Links were automatically created whenever a property `id` was found, using *LinkRoleFromName* and a Facebook specific URL pattern based on *metadata* (content of the `connections` information is provided specifically upon request using the HTTP parameter `metadata=1`).

Google+. Type IDs in Google+ were generated from property values using *IdFromValue* on `kind`, `type`, and `objectType` (prefix `#plus` was removed). Analogously to Facebook, as a fallback *IdFromReferenceName* was used, and links were created whenever a property `id` was found. Using this strategy, only links from views on objects to their respective *full* representation can be created in Google+. Thus, in order to enable automatic merging, two additional links (from user to activities, and from activities to comments) had to be added manually.

LinkedIn. Hints for type IDs are not provided in LinkedIn JSON data. Thus, all IDs were created automatically using *IdFromNameConcatenation*, and later replaced manually. Analogously to Google+, links were automatically created for `id` properties, whereas four were added manually: between persons and their group memberships, their connections and their suggestions, as well as between activities and comments.

5.2 Extraction and Transformation

To evaluate the generated schemas as well as the subsequent transformation phase, we compared the conceptual models to the API documentation of the corresponding social networks[15]. Due to reasons of brevity, in the following we discuss only a subset of the extracted schemas, focusing on the properties of *users* and their references to other entities. The extracted schemas (as UML diagrams and ECORE models), as well as manually created ones on the basis of online documentation, can be found on our project website[16]. To give a first impression of the extracted schemas here, Table 3 summarizes the number of classes, properties and references as metrics. These metrics provide a first hint on the degree of schema heterogeneity. For example, the extracted Facebook schema comprises around twice as many classes as Google+ and LinkedIn do, since Facebook provides metadata, which allowed us to extract further information linked indirectly to the test users (e. g., a city in Facebook is represented as a first-class entity with further links to other users). When comparing Google+ and LinkedIn, which both do not provide such metadata, one can conclude that LinkedIn tends to represent information in terms of first class objects (34 classes in LinkedIn versus 25 in Google+) whereas Google+ provides these information rather in terms of properties (on average, 3 properties per class in Google+ vs. 2 properties per class in LinkedIn).

Table 3: Metrics for extracted schemas

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

To get an impression of the completeness of the schemas, Table 4 contrasts the number of properties and references found in the documentation with the ones successfully extracted. Interestingly, for Facebook and LinkedIn our process was actually able to extract more schema elements than we would have expected to be present in the test user profiles, as detailed below. However, the extracted set is not a superset, as indicated by their intersection.

Table 4: Documentation vs. extracted schemas

	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

Facebook. The created Facebook schema (cf. excerpt in Figure 6) contains 19 of 24 expected user properties and references. The extracted schema contains an additional property `type`, originating from a *metadata* property contained in the response, which was used to extract the class name (but is not listed in the documentation). This meta-information also provides names of properties, references, and types, which thus, match the documentation. Concerning the types of properties, all primitive types match. Still, the documentation gives additional constraints on the properties `link` (string that may only contain a URL), `locale` (ISO language/country code), and `updated_time` (ISO date-time). As documented, the references `work` and `education` match to type array with unbounded cardinality. Notably, Facebook often encloses collections of references (e. g., list of friends) inside dedicated types to support retrieving subsets of the collection (e. g., the reference from `friends` to `paging`).

Concerning Facebook *pages*, the prime concept to represent entities apart from users, the documentation does not specify concrete subclasses. However, many subclasses of page exist, as indicated in the web UI during the creation of pages, as well as in extracted JSON data. Thus, the property `category` was turned into subclasses of page, and additional intermediary classes distinguishing between refer-

[15]`developers.facebook.com/docs`, `developers.google.com/+/api`, `developer.linkedin.com/apis`
[16]`social-nexus.net/publications`

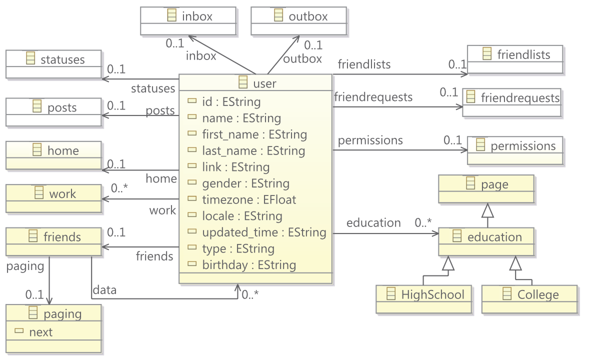


Figure 6: Excerpt of Facebook schema

enced page subsets were introduced (e. g., class `education` with its two subclasses `HighSchool` and `College` created by *HomogenizingArrayType*, each having different additional properties). Obviously, with our limited test user profiles, these inferred class hierarchies are far from being complete, not least, since new page categories are introduced by Facebook every now and then.

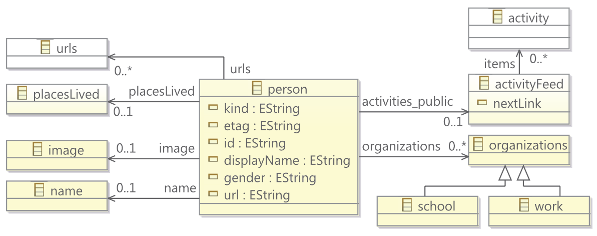


Figure 7: Excerpt of Google+ schema

Google+. In case of Google+, the extracted schema (cf. excerpt in Fig. 7) contains 18 of the 27 expected properties and references from the documentation, whereupon all names and types match. An additional one, namely `activities_public`, comes from manually adding a link to the schema, which resulted in a request to the Activities API of Google+. Similar to Facebook, Google+ provides a concept for retrieving subsets of collections (e. g., utilized in `activityFeed`). Interestingly, even though the user's birthday was entered for the Google+ user, the API does not deliver a value for the corresponding documented property.

LinkedIn. The schema extraction process for LinkedIn found 23 of 24 expected properties and references (cf. excerpt in Fig. 8). Note, that in ECORE dashes are not allowed within property names, thus, they have been removed. Similar to Facebook, the documentation notes restrictions for several properties of type string, which were not inferred automatically due to the small number of input profiles.

Note, that LinkedIn uses the same structure for all references of JSON objects. This structure is similar to Facebook and Google+. It consists of referenced objects in an array named `values`, and a property `_total` providing their *total count*. Hence, all classes referenced by person were initially named `_totalvalues`, and had to be renamed manually for clarity of presentation in this paper. As a remainder of the initial structure, the schema excerpt of LinkedIn contains a class named `_total`, which is the target of five references.

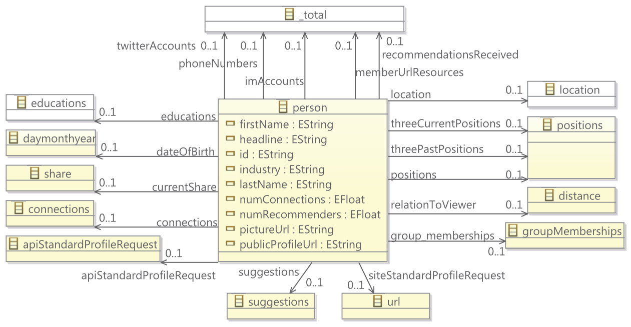


Figure 8: Excerpt of LinkedIn schema

The name of this class was derived automatically from its sole property `_total`. The `values` arrays resolved to be empty, since no instances were provided in the test user profile. Hence, the schema extraction process was unable to derive a single concrete class per reference.

5.3 Integration—Cross-Network Analysis

Overlap. To complement our previous work [12, 13] on comprehensive user modeling, we tried to find mappings between semantically equivalent schema elements of the different social networks. Therefore, as a starting point, we used the well-known schema and ontology matching tool COMA++ [17] [2, 17]. To import the schemas in COMA++, we simply exported the ECORE models to XML Schemas, being conform to the XMI[18] interchange format. The three schemas were then compared in a pairwise manner.

A brief manual evaluation showed, that the results at hand already provide a solid basis for integration, specifying multiple *semantically correct 1:1 correspondences* between attributes. However, after a first examination the number of false positives seems to be rather high. Certainly, the capabilities of COMA++ were not fully exhausted. Also, for a thorough evaluation of the matching results, manually created "perfect mappings" would be required, which are, however, not yet available. A few information samples about users and their addresses are shown in Table 5, already giving an impression of heterogeneities amongst social network schemas, along with the computed similarity values from COMA++ in Table 6. The heterogeneities found between these properties are discussed in the following.

Table 5: Sample user and address properties

		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

[17] COMA version 2008f, matcher: COMA_OPT
[18] www.omg.org/spec/XMI

Table 6: User and address cross-network overlap

		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	0.46	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

Heterogeneities. Analyzing the corresponding schema elements manually, we found that even in such tiny extracts of social network data, manifold heterogeneities exist, as outlined in Table 5. These include *semantic* as well as *structural heterogeneities* [14]. Concerning the former, we see that certain information is not available from all social networks, such as gender (cf. LinkedIn), or birthday (cf. Google+). Consequently, the integration of user profiles from several social networks will lead to semantically enriched user profiles, as also indicated by Abel et al. [1]. Concerning the latter, one may find that information is represented structurally different. Such structural differences [23] range from simple *naming differences* (e. g., the concept of a first name is represented by three differently named attributes being `givenName`, `first_name` and `firstName`) over *fine-grained cardinality differences* with respect to attributes (e.g., zip code, city, and country, the concepts making up an address, are represented as a single string value in Google+) to *coarse-grained cardinality differences* with respect to classes (e.g., Google+ represents user information in two separate classes `Person` and `Name`, whereas Facebook and LinkedIn represent the same information within a single class). Consequently, the application of integration tools for schema matching and mapping is mandatory to support the task of user profile integration in the context of social networks.

6. LESSONS LEARNED & FUTURE WORK

In the following paragraphs, we summarize lessons learned from schema extraction, transformation, and analysis for integration, and thereupon discuss directions of further work.

Obtainment of user profile data is a critical success factor. The proposed process obviously depends on the availability of user profile data. This data may be either obtained from real users or from generated pseudo users. Concerning the former case, authorization from users is required to allow requests for user profile information. Concerning the latter case, the mass of information that may be entered into today's social networks makes the creation of artificial user profiles a tedious task. For example, in Facebook, entering information for the multi-purpose class `page` is supported with a wizard-like user interface, which allows selection of a page category by combining a main category and a sub-category from drop-down lists. Each combination leads to a dynamically created input form for specifying property values. The properties of a particular page category may partly overlap with the ones of other page categories. Altogether, 193 different page categories could be entered (cf. Excel sheet on our website[19]). However, the fact that Facebook does not delete information leads to somewhat surprising results: For example, upon selection of a main category, such as `Hotel`, specific text fields (e. g., public transit and general manager) can be filled in. When switching to a different main category (e. g., `Books&magazines`), surprisingly these input fields are still available, although they are not when this category was selected initially. Moreover, instances created this way also provide public transit and general manager information when requested from the API.

[19] `social-nexus.net/publications`

Comprehensive schema extraction requires manual intervention. In order to request complementary parts of user profiles, links between JSON documents are necessary. Facebook provides such links upon request (`metadata=1`), but Google+ and LinkedIn do not. Consequently, links must be obtained from the documentation and added manually. Since the documentation is available in HTML, information extraction techniques may reduce this manual effort.

Same objects are represented differently. All three investigated social networks provide different views on the same objects, depending on the kind of issued request. For example, in Facebook a user's name and profile picture are provided with each feed entry, whereas only the name is provided in the list of friends. As a result, extracted schemas may comprise a large number of view classes (one for each different view that is employed in a particular context), each ultimately referring to the same full class. Consequently, the merging of resulting type definitions is indispensable.

Queries: restriction vs. relaxation. Social network providers apply different policies concerning their API requests: while Facebook and Google+ deliver complete responses and support restriction with queries on demand, LinkedIn by default delivers only restricted responses and demands that further information is queried. In the latter case, automated schema extraction is only possible for the default information (alternatively, extraction can base on different input information, such as HTML documentation).

Completely private objects vs. reusable objects. In the three social networks, some objects (e. g., a particular employment period of a user) are designed to be completely private, which means that they do not have an ID and cannot be requested separately. Others are reusable and can be referred to from many objects. Sometimes, surprising entities are designed reusable: For example, in Facebook, years are dedicated pages, which are referenced from other objects instead of being modeled as properties thereof.

Heterogeneous arrays may not be representable in every technical space. In JSON Schema, arrays can be completely heterogeneous (i. e., each element in the array may have its own schema). In contrast, in UML a common base type is assumed to exist as the target of an association, which may lead to schemas comprising a generic root class and wrappers for primitive types, such as known from Java.

Transformation of instances are prerequisite for profile integration. Besides the transformation of models to bridge technical spaces between different modeling tools, a particularly interesting question towards the integration of user profiles arises: Can we automatically derive transformations on the model level from those being defined on the meta model level, such that the instances of the source model (e. g., JSON data of a user profile) can be automatically translated into instances of the newly generated target model (e. g., instances of Java classes)? Answering this question, firstly, would give us further insight on the value and accuracy of the extracted schemas. Secondly, it may bring us towards automated creation of mapping tools, not only for user profile integration, but for various other scenarios as well.

Co-evolution of instance extraction and user profile integration applications with social networks. As further work basing on automatically extracted schemas, we plan to create a model-driven instance extraction framework for social networks, which, given schemas of social networks and transformations, can be used to (i) create request code for these networks, (ii) create test mockups for third-party applications thereof, (iii) derive user profile integration rules from schema mappings, and especially, (iv) support co-evolution of request code and integration rules with the accessed social networks. As a first implementation of that framework, on the basis of a transformation from social network schemas to OWL ontologies, social network information extraction into RDF graphs is planned. Thereupon, user profile integration shall be implemented, using semantic web techniques, such as rule reasoners fed with integration rules derived from schema mappings.

7. ACKNOWLEDGMENTS

This work has been funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under grant FIT-IT 825070.

8. 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.