SmartMatcher – How Examples and a Dedicated Mapping Language can Improve the Quality of Automatic Matching Approaches

Horst Kargl and Manuel Wimmer
Business Informatics Group
Vienna University of Technology
Favoritenstrasse 9-11, 1040 Vienna, Austria
{kargl|wimmer}@big.tuwien.ac.at

Abstract

Information integration has a long history in computer science. It has started with the integration of database schemas in the early eighties. With the rise of the semantic web and the emerging abundance of ontologies, the need for an automated integration increased further. A lot of automated matching approaches and tools have been proposed so far. The typical output of such tools is a simple one-to-one alignment mostly based on schema information, e.g., similar names and structures of schema elements. However, these alignments cannot cope with schema heterogeneities, hence, these problems must be resolved manually. Furthermore, there is no automated evaluation of the quality of the alignments based on the instance level, because the matching approaches are not bound to a specific integration scenario, e.g., transformation or merge. In this work we propose the SmartMatching approach, which can be seen as an orthogonal extension to existing matching approaches for increasing the quality of the automatically produced alignments for the transformation scenario. This is achieved by using an executable mapping language for bridging schema heterogeneities and by using instance models to evaluate the quality of the alignments in an iterative and feedback-driven process inspired by machine learning approaches.

1. Introduction

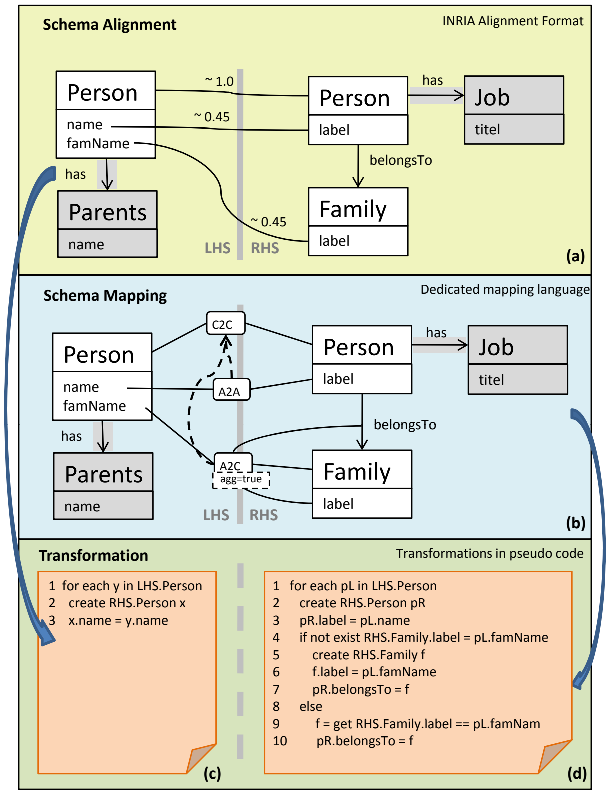
Information integration deals with the problem of building a general view on different kinds of data. The origin of information integration lies two to three decades ago in the area of database engineering, as autonomous databases started to federate [17]. With the advent of semantic web and its schema-based technologies (RDFS and OWL) for describing, storing, and exchanging data, the need for an automatism arises.

In general, we can distinguish between three categories of matching approaches [18]. The first category are schema-based approaches which use only schema information as input for the matching process. The second category are instance-based approaches which use instances as input for the matching process, only. And the third category is about hybrid approaches which use schema and instance information.

The main requirements of matching approaches are the completeness and correctness of the produced mappings as, well as usability. However, the following three problems complicate the realization of these three requirements.

Different mapping execution scenarios: Matching approaches can be applied to solve different kinds of integration problems. For example, alignments from existing matching approaches can be used to merge or transform instances. However, each integration scenario entails other interpretations of alignments with different side conditions. Due to the generality of common approaches, it is hard to cover all aspects of each scenario in the alignments, and most of the approaches lack a binding to an execution environment.

Schema heterogeneity: Matching approaches produce alignments which express correspondences between elements belonging to different schemas. Most of the schemas to be integrated share similar semantics but describe these semantics with different structures. Currently used equivalence correspondences cannot cope with schema heterogeneities. Hence, the results of such approaches, often described in the INRIA alignment format [8], must be interpreted, and missing alignments between schematically heterogeneous elements must be explored by the user in further steps when the transformation or the merge scenario is written in an executable language. To fulfill the need of an executable and evaluable approach, a more powerful mapping language that can cope with schema heterogeneities is needed.



```
for each y in LHS.Person
  create RHS.Person x
  x.name = y.name
```

```
for each pL in LHS.Person
  create RHS.Person pR
  pR.label = pL.name
  if not exist RHS.Family.label = pL.famName
     create RHS.Family f
     f.label = pL.famName
     pR.belongsTo = f
  else
      f = get RHS.Family.label == pL.famNam
      pR.belongsTo = f
```

Figure 1. Simple mappings (1a), complex mappings (1b), transformation code (1c, 1d)

Automatic evaluation: Matching results are suggestions and one cannot rely on them. To make assertions about the quality of the resulting alignments, quality measures can be applied [2, 16, 21]. To calculate those quality measures, all correct alignments must be already given by the user. Typically mistakes are wrong alignments, and alignments which exist, but are not found by the matching approach.

In this work we introduce the *SmartMatching* approach which is dedicated to the transformation scenario. It will tackle the three aforementioned problems, namely execution problem, schema heterogeneities, and evaluation of the mappings. To cope with the execution problem and schema heterogeneity, we do not use simple alignments which describe similarities between schema elements. Instead, we use a dedicated mapping language that can cope directly with schema heterogeneities. This mapping language has a direct link to an execution environment. Hence, the mappings found can be directly executed without any further user interaction.

The evaluation of the mappings can be done automatically in the *SmartMatching* approach during the mapping process. Therefore, we need instances which describe the same real-world domain. With these semantically equivalent examples, expressed in both of the schemas to be integrated, we can compare the results of the transformations. If there are still differences between the transformed instances, further adaptation of the mappings are necessary, otherwise the mappings represent correct correspondences between these two schemas.

The *SmartMatching* approach consists of three components, namely, the *Initial Matcher*, *Fitness Function* and the *Mapping Engine*, and components responsible for the transformation scenario. Depending on the types of schemas to be integrated, different kinds of mapping languages can be plugged into the *MappingEngine*. As a side condition, the mapping language must be declarative and the mapping operators must have statically typed interfaces. Based on the interface definition, different selecting strategies for mapping operators can be implemented in the *MappingEngine*.

In the rest of the paper, we use the term *matching* as the automatic process of finding bridges between schema elements, and the term *mapping* and *alignment* for those bridges. Furthermore, the term *mapping* is used for complex mappings which can bridge schema heterogeneities and the term *alignment* for simple equivalent correspondences.

The paper is structured as follows. In Section 2 we give further motivation for the *SmartMatching* approach. In Section 3 we introduce the *SmartMatching* approach in detail. Section 4 shows related approaches and describes the differences to the *SmartMatcher*. Finally, in Section 5, we sum-up our approach and give an outlook on future work.

2. Motivating Example

Figure 1(a) depicts an example of two semantically similar but schematically different schemas. In this paper, we use UML class diagram notation for representing schemas. The classes with white background color on the left hand side (LHS) and on the right hand side (RHS) are semantically equivalent but schematically different. The classes with gray background color are not semantically equivalent and therefore not considered in the mappings. The LHS schema describes a concept person (cf. class *Person*) with its first name (cf. attribute *Person.name*) and family name (cf. attribute *Person.famName*). The RHS schema describes the same semantics, but using a different schema representation. It consists of a class *Person* and an attribute *label*, which represents the name of the person. The information of the family name is represented in an own class, which has also an attribute *label*, representing the family name value. On the LHS schema, the same value of the attribute *famName* can occur repeatedly in different instances. In contrast, on the RHS only one *Family* instance exists for each person with the same family name. This means, the values of the *Family.label* attribute are unique.

Existing matching tools can deliver alignments with similarity values between schema elements mostly described in the INRIA alignment format [8]. Figure 1(a) shows an alignment between LHS *Person* and RHS *Person* with a similarity of 1.0. This mapping was found due to name similarity with schema-based matching techniques. Because of different names for semantically equivalent attributes, only instance-based or hybrid matching techniques [18] can find alignments between the attribute *Person.name* and *Person.label* as well as *Person.famName* and *Family.label*.

When interpreting the alignments between the LHS and RHS elements, the transformation code depicted in Figure 1(c) can be derived. All instances of type *Person* on the LHS are transformed into instances of type *Person* on the RHS. Additionally, the alignment between the two attributes *Person.name* and *Person.label* leads to the code in which each value of the attribute *Person.name* of a *Person* instance on the LHS is shifted into the value of the attribute *Person.label* of a *Person* instance on the RHS.

When transforming this alignments to transformation code, only some parts of the required transformation code can be derived without any further interpretation of the alignments. These alignments do not express that each value of *Person.famName* attribute should be transformed into an instance of the class *Family*. Additionally, the value of the *Person.famName* attribute should be set to the value of *Family.label* attribute and a link between *Person* instance and *Family* instance should be created.

To express richer mappings between two schemas, a dedicated mapping language which is aware of schema heterogeneities can be used. For the mapping example in Figure 1(b) we have used the *CAR* mapping language [22], which is an executable and declarative mapping language that can cope with most common schema heterogeneities. It is based on a combination of *classes*, *attributes* and *references*. Hence, it is applicable to schemas which are based on an object oriented core. But *CAR* is not an integral feature of the *SmartMatcher*. Each declarative mapping language that can cope with schema heterogeneities and has a binding to an execution environment can be used.

The two classes *Person* and its attributes *Person.name* and *Person.label* can be mapped straightforwardly with an class2class (C2C) and attribute2attribute (A2A) mapping. The attribute *Person.famName* is mapped with an attribute2class (A2C) mapping to the class *Family*. This means making an object of type *Family* from a value of the *Person.famName* attribute. The aggregation flag of A2C is set to *true*, hence, each distinct value of *Person.famName* is transformed into one object of type *Family*. Additionally, the value of *Person.famName* is transformed into the value of *Family.label*, and the link between the instances of class *Person* and *Family* must be set accordingly.

The interpretation of the *CAR* mappings allows to transform instances of the LHS into instances of the RHS as described in pseudo code in Figure1(d). Additionally to the code shown on the LHS of Figure 1(c), the creation of family instances and the linking between family instances and person instances can be established. The operational semantics of the A2C mapping operator is described in pseudo code on the right hand side of Figure 1(d) from line 4 to 10. The first step is to check wheather an instance of type family with the value of the attribute *Family.label* which equals the family name value of the actual person instance (cf. line 4) exists. If such an instance does not exist, a new instance of type family is created (cf. line 5) and the value of the attribute label is set to the value of the family name attribute of the actual person (cf. line 6). In addition, the actual person instance and the newly created family instance are connected with a *belongsTo* link (cf. line 7). In the opposite case, the correct matching family instance is selected (cf. line 9) and only the link between the actual person and the selected family instance must be set (cf. line 10).

Many automatic approaches produce alignments between schema elements, but lack the expressiveness to cope with schema heterogeneity problems. More powerful mapping languages can tackle the schema heterogeneity problem, but no automatism for finding the correspondences is provided. The *SmartMatching* approach will combine the strength of both to automatically generate mappings with a more powerful mapping language. Furthermore, an automatic self-evaluation of the found mappings is provided.

3. The *SmartMatching* Approach

In this section, we give a detailed introduction to the *SmartMatcher*. Figure 2 gives an overview of the *SmartMatcher's* architecture and its integration process. It shows the core components, namely, the *Initial Matcher*, *Mapping Engine* with its *Mapping Language Repository*, and *Fitness Function*. As a prerequisite for using the *SmartMatching* approach, a *Mapping Language* with an a binding to an *Execution Environment* must be provided. For the self-evaluation feature, additionally instances representing a real world domain modeled with both schemas are required. The requirement for the mapping language in the SmartMatching approach is that the mapping operators have typed interfaces. Hence, the mapping operators can be used like pieces of puzzles to build a bridge between two schemas. Finding appropriate mappings between them can be seen as searching a way through a search graph. Each node expresses the state of the mapping, and each edge is the application of a mapping operator. The aim of the *SmartMatching* approach is to find heuristics to reduce the search space and find a correct mapping quickly.

Section 3.1 gives a generic description of the workflow of the matching process. Due to lack of space, a concrete example is not presented in this paper, but may be found at the project web-site [blinded]. In Section 3.2, we describe the role of the core components in detail. In Section 3.3, we give the motivation for using concrete examples and describe how they can increase the efficiency of the automated matching.

3.1. The *SmartMatcher* Process

In the following, we describe the workflow of the SmartMatching process, which can be divided into 8 steps.

Step 1) Develop example instances. In this step, instances are developed for each of the schemas to be integrated. Figure 3 shows the general idea. The aim is to define a real world example that uses most of the schema concepts. Describing the same real world example with both schemas produces instances which use semantically equivalent schema concepts (see the right hand side of Figure 3). Hence, not overlapping schema elements can be filtered out.

Step 2) Generate initial matching. In this step we use existing matching tools to create basic alignments between similar schema concepts. We require the alignments to be expressed in the INRIA alignment format [8]. Due to the INRIA alignment format, we can use all matching tools which deliver such a format.

Step 3) Interprete initial mappings.. We can translate the alignments produced in step 2 to an initial mapping model based on the types of the elements referenced by the alignments. The use of the *Initial Matcher* increases the performance, i.e., the search space for the *SmartMatching* approach is reduced if several correct mappings have been found.

Step 4) Derive transformation. In this step a transformation is generated from the mapping model. The transformation is responsible to transform LHS instances into RHS instances. Based on the schemas to be transformed, the mapping model can be convert into different kinds of transformations. For example, if XML Schema is used as schema language, XSLT and XQuery can be used as transformation environments for transforming the actual XML documents.

Step 5) Transform instances. The execution environment is responsible for reading the instances conforming to the LHS schema and transforming them into instances conforming to the RHS schema, based on the derived transformations.

Step 6) Calculate differences. In this step the *actual* and the *target* RHS instances are compared by means of their attribute values and links. The differences are collected in a *diff model* which can be used to evaluate the quality of the mappings between *Schema A* and *Schema B*. Furthermore, in step 5 we have two termination conditions for the

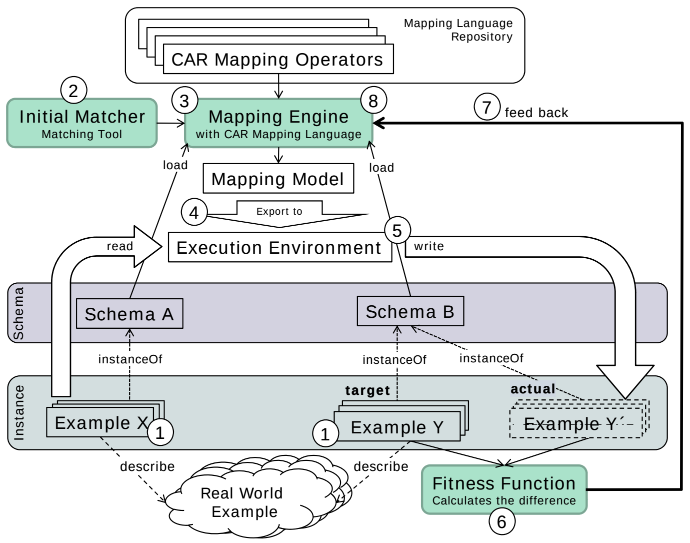


Figure 2. Architecture and process of the *SmartMatching* approach

SmartMatching process. The first one occurs if no more differences between the actual and target instances exist, i.e., the mapping is complete. The second one occurs if the differences remain the same during several iterations, i.e., a final point is reached for a certain set of mapping operators and example instances.

Step 7) Propagate differences. In this step, the differences calculated by the *Fitness Function* are propagated back to the *Mapping Engine*. More specifically, missing and wrong values are propagated back, expressed in the *diff model* of the actual and target instances.

Step 8) Interpret differences and adjust mapping model. This step is responsible for analyzing the propagated differences and for adapting the mappings between the LHS and RHS schema by searching for and applying appropriate mapping operators for missing or wrong mappings.

After step 8, a new iteration starts at step 4 until step 6. In step 6 the actual and the target RHS models are compared again. If there are no more differences, the process is finished, else the iteration is continues until step 8, where a new iteration begins.

3.2. The *SmartMatcher* Components

In this section, we give more insights into the components of the *SmartMatching* approach.

Initial Matcher: The aim of the initial matching process is to narrow the search-space. Without an initial matching, the operators of a mapping language must be applied randomly. This would increase the search time. The *Ini-*

tial Matcher component reuses existing matching tools like COMA++, Alignment API, FOAM etc. The results of such matching tools are equivalent alignments with a similarity value delivered in the INRIA Alignment Format [8]. These results are the input of the next component, the *Mapping Engine*.

Mapping Engine: The *Mapping Engine* is the core component of the *SmartMatcher*. Its role is twofold. First, it must interpret the results of the *Initial Matcher* and generate mappings for them, Step 3 in Section 3.1. Due to the fact that the *Mapping Engine* is aware of the mapping language, appropriate mapping operators can be taken from the *Mapping Language Repository* for the delivered alignments. Second, it must interpret the results of the *Fitness Function* and adopt the given mappings, Step 8 in Section 3.1. The adoption is based on differences between the transformed instances. This is the evaluation part of the approach. If no differences were found, the mapping ought to be correct.

Fitness Function: The aim of the *Fitness Function* is to calculate the differences between the *actual* and *target* instances. Those differences are propagated back to the *Mapping Engine* to adopt the current mappings. The calculation of differences is based on attribute values and the amount of instances and their structure. The *by-example* approach, discussed in Section 3.3, leads to semantically equivalent instances, only the structure of the instances of the LHS and the RHS schema can differ. After the transformation of the LHS instances into the *actual* RHS instances, this instances can be compared with the *target* RHS instances by counting the amount of instances of a specific type, and by comparing the values of attributes. The differences are marked in the target RHS instances (the *diff model*) and can be propagated back to the *Mapping Engine*.

3.3. How can Concrete Examples Improve the Matching Process?

Concrete examples are the core elements of improving the mapping quality and supporting the self evaluation mechanism. The bottom of Figure 2 shows a cloud which stands for a concrete real world example of a specific domain. This example is described in natural language.

This real world example is modeled with each of the schemas to be integrated with their schema concepts. To find the overlapping parts of the schemas to be integrated, both instances must describe the same parts of the real world example.

Figure 3 gives an insight how the overlapping parts can be modeled. The real world example is expressed in one of the schemas to be integrated, e.g., *schema A*. The schema elements used are marked as a filled circle. The part of the real world example modeled with *schema A* is also depicted

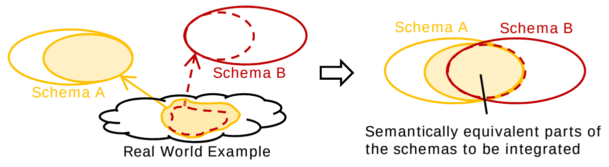

Schema A
Schema B
Real World Example
Schema A
Schema B
Semantically equivalent parts of the schemas to be integrated

Figure 3. Building instances from a real world example

in the real world example cloud. Now the same parts of the real world example described with *schema A* must be described with *schema B*. Parts of the real world example that can be modeled with *schema B* but not with *schema A* must be left out. If not all parts modeled with *schema A* can be expressed in *schema B*, they must also be left out. The overlapping part of both schemas is depicted on the right hand side of Figure 3.

This process of describing the same real world example with both schemas leads to instances which describe the same concepts but with different schema elements, that means we get the semantically overlapping part of both schemas. Hence, we do not have to cope with semantic schema heterogeneities.

Other instance-based approaches also compare values of instances to find similarities between schema elements. Due to term ambiguities like synonyms, homonyms, the results of such a comparison cannot be trusted. By using the same real world example and the same terms for the same concepts the term ambiguities problem can be tackled.

The concrete examples can further be used to increase the completeness and correctness of the found matchings. The concrete examples serve to evaluate the mappings between two schemas by comparing the transformed examples. Thanks to the real world example, the *actual* and the *target* instances on the RHS should be the same. If this is the fact, all automatically found mappings are correct.

4. Related Work

In the last decades a lot of contributions have been presented in the field of schema integration [3, 10, 13]. Based on those works, a lot of schema matching approaches have been built [14, 19, 18].
To the best of our knowledge there is no equivalent approach to the *SmartMatcher* in respect to how instances are used to find schema similarities and provide self-evaluation. In general, we can classify our approach due to the dimensions given in [14] as hybrid meta-matcher approach.

Matching approaches: The *SmartMatching* approach is orthogonal to existing matching approaches, such as

COMA++ [2], FOAM [20], CROSI [9], Alignment API [8], which produce alignments in the INRIA alignment format. Hence, we can use them to increase the results of the initial matching phase. In contrast to given instance-based approaches, we do not use given instances and try to find the similarities in their values, as described for example in [5, 7, 11]. We use dedicated real world examples to define instances. Hence, we can be sure the same terms denote the same concepts.

Schema heterogeneities: We use a dedicated mapping language which can explicitly bridge the heterogeneity gap for common heterogeneity problems. *Clio* [12] is a matching tool from IBM which is also able to bridges heterogeneity problems, but in an implicit manner. They use value correspondences (a kind of simple alignments) for mapping the schemas and in a second step they interpret sets of value correspondences and compute from these sets complex mappings for the most common schema heterogeneities known from the database field. *QuicMig* [6], is another matching approach which bridge schema heterogeneities to fulfill the transformation scenario. Similar to the *SmartMatching* approach, *QuicMig* makes use of manually created example instances. But the processing of these instances differs, *QuicMig* uses existing instance-based matchers to generate alignments. The *SmartMatching* approach calculates the differences between manually created instances, which are transformed to adopt the mappings between the schemas to be integrated.

Machine learning: The *SmartMatcher* is also related to supervised machine learning [1]. In general, supervised machine learning tries to find a functional relationship model to get the output instance from the input instance with the help of a set of training instances. The actual and targeted output are compared and the differences are propagated back to adopt the functional relationship model. The resulting functional relationship model is evaluated with a set of evaluation examples. In case of the *SmartMatcher* the functional relationship model is the mapping model between the LHS and RHS schema. The prepared instances which describe the real world example serve as input and output. The *Fitness Function* is responsible for calculating differences and feeding them back to adapt the mappings. At least one pair of instances describing the same real world domain is used to train the *SmartMatcher*. Further pairs can be used to evaluate the mappings if they hold for different kind of instances.

5. Conclusion and Further Steps

In this paper we have presented a generic approach for improving the completeness and correctness of automatically produced mappings for the transformation scenario. Our approach is orthogonal to existing schema-based, instance-based, and hybrid matching approaches and can be tailored to different kinds of integration scenarios, schema languages, and mapping languages. The contribution of this paper is the usage of instances representing the same real world domain and an initial mapping as a starting point, combined with a feedback-driven iterative approach. This iterative approach is used for improving the initial mappings based on the differences between the user defined target instances and the generated actual instances. Furthermore, this test-driven approach enables the automatic evaluation of the generated mappings. Due to the permanent evaluation of the *SmartMatching* process, the final mapping result can be treated as correct for the defined set of example instances. Hence, the mapping results of existing matching tools can be improved. More specifically, wrong mappings can be eliminated and missing mappings can be explored. However, the side condition and critical part of the approach is the definition of the instances which describe the real world example.

In contrast to other automatic matching approaches, the *SmartMatching* approach needs more work in the preparation phase, i.e., the establishment of the example instances, but less in the reworking phase, i.e., the manual evaluation and the adjustment of the mappings. Hence, our hypothesis is that the effort building the instances for the real world example is less than the evaluation and reworking phase of other approaches. Furthermore, the real world examples can be used in other integration scenarios. For example, if *schema A* and *B* have been already integrated with the *SmartMatching* approach, this means the real world example instances have been created and therefore can be reused when integrating *schema A* and *B* with *schema C*.

The *SmartMatcher* is already implemented prototypically for schemas defined in *Ecore*, the metamodelling language of the Eclipse Modeling Framework (EMF) [4]. In particular, we are using *CAR* as mapping language, and we have implemented a simple initial matcher component. In addition, we provide an import functionality for alignment models based on the INRIA alignment format. Furthermore, we have implemented a *FitnessFunction* which compares the target model with the transformed actual model. The differences between target models and actual models can be propagated back to the *MappingEngine* which produces *CAR* mapping models which can be transformed into transformation definitions based on colored petri-nets [15].

Further steps are improving our current prototype, and the evaluation of the aforementioned hypothesis that the preparation phase is less work than the reworking phase. Finally, we have to evaluate our matching approach extension regarding completeness and correctness of the mappings with empirical experiments.

References

[1] E. Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, October 2004.

[2] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908, New York, NY, USA, 2005. ACM Press.

[3] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[4] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose. *Eclipse Modeling Framework (The Eclipse Series)*. Addison-Wesley Professional, August 2003.

[5] A. Doan, Y. Lu, Y. Lee, and J. Han. Object matching for information integration: A profiler-based approach. In *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, pages 53–58, 2003.

[6] C. Drum, M. Schmitt, H.-H. Do, and E. Rahm. QuickMig - automatic schema matching for data migration projects. In *Proceedings of the Sixteenth Conference on Information and Knowledge Management (CIKM2007)*, Lisabon, November 2007.

[7] D. Engmann and S. Massmann. Instance matching with COMA++. In *Workshop: Model Management und Metadaten-Verwaltung (BTW)*, pages 28–37, 2007.

[8] J. Euzenat. An API for ontology alignment. In *Proceedings of the third International Semantic Web Conference (ISWC 2004)*, volume 3298 of *LNCS*, pages 698–712, Hiroshima, Japan, November 7-11 2004. Springer Berlin / Heidelberg.

[9] Y. Kalfoglou and B. Hu. Crosi mapping system (cms) results of the 2005 ontology alignment contest. In *Proceedings of the 3rd International Conference on Knowledge Capture (KCap 2005) workshop on Integrating Ontologies*, Banff, Canada, Oct. 2005.

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

[11] T. Kirsten, A. Thor, and E. Rahm. Instance-based matching of large life science ontologies. In *Proceedings of Data Integration in the Life Sciences, 4th International Workshop (DILS 2007)*, pages 172–187, 2007.

[12] R. J. Miller, M. A. Hernandez, L. M. Haas, L. Yan, H. C. T. Ho, R. Fagin, and L. Popa. The Clio project: managing heterogeneity. *SIGMOD Rec.*, 30(1):78–83, March 2001.

[13] C. Parent and S. Spaccapietra. Issues and approaches of database integration. *Commun. ACM*, 41(5es):166–178, 1998.

[14] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal — The International Journal on Very Large Data Bases*, 10(4):334–350, December 2001.

[15] T. Reiter, M. Wimmer, and H. Kargl. Towards a runtime model based on colored petri-nets for the execution of model transformations. In *3rd Workshop on Models and Aspects - Handling Crosscutting Concerns in MDSD*, Berlin, Germany, July 2007.

[16] G. Salton and M. J. McGill. *Information Retrieval. Grundlegendes für Informationswissenschaftler*. McGraw-Hill, Maidenh, 1987.

[17] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.

[18] P. Shvaiko. et. al. OpenKnowledge Deliverable 3.1.: Dynamic ontology matching: a survey. Technical report, Informatica e Telecomunicazioni, University of Trento, 38050 Povo - Trento (Italy), Via Sommarive 14, July 2006.

[19] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *Journal on Data Semantics IV*, pages 146–171, 2005.

[20] Y. Sure and M. Ehrig. FOAM - framework for ontology alignment and mapping - results of the ontology alignment evaluation initiative. In *Proceedings of the K-CAP 2005 Workshop on Integrating Ontologies*, 2005.

[21] C. J. van Rijsbergen. Information retrieval. Online, 1979.

[22] M. Wimmer, H. Kargl, M. Seidl, T. Reiter, and M. Strommer. Integration of ontologies with CAR mappings. In *First International Workshop on Semantic Technology Adoption in Business – STAB'07*, pages 27–38, Mai 2007.