Mapping Pragmatic Class Models to Reference Ontologies

Heiko Paulheim [#1], Roland Plendl [#2], Florian Probst [#3], Daniel Oberle [*4]

[#]*SAP Research*
Bleichstrasse 8, 64283 Darmstadt, Germany
[1]heiko.paulheim@sap.com
[2]roland.plendl@sap.com
[3]f.probst@sap.com

[*]*SAP Research*
Vincenz-Priessnitz-Strasse 1, 76131 Karlsruhe, Germany
[4]d.oberle@sap.com

***Abstract*— Capturing a domain of discourse in an object-oriented class model and in a reference ontology leads to different results. On the one hand, modeling decisions for class models are motivated by pragmatic and efficiency-related choices because modeling decisions are motivated by different choices. However, information integration scenarios require a coexistence between reference ontologies and class models at runtime. When implementing an integration scenario, objects need to be transformed between both types of models, thus requiring expressive, executable, and bidirectional mappings. In this paper, we present a novel approach for a bidirectional mapping between a class model and an ontology. The approach is both non-intrusive and dynamic. That means it can be integrated in existing IT systems without having to change the class model and it does not rely on static 1:1 mappings, respectively.**

I. Introduction

The recent W3C Semantic Web recommendations, such as OWL, are increasingly used to specify reference ontologies. The idea of a reference ontology is to be a generic, commonly agreed upon conceptual model of a domain. To this end, modeling decisions are taken in a way such that the semantics of domain terms are captured as precisely as possible. As an example, consider the ISO 15926 Oil and Gas ontology standard whose purpose is to provide a lingua franca for Oil and Gas IT systems [1].

One goal of reference ontologies is to facilitate and enable information integration across IT systems. For example, the declared goal of the Norwegian Oil and Gas industry is to enable information integration based on ISO 15926 to support, e.g., the integration of different data sources and sensors, the validation of information delivered by different sources, and the exchange of information within and between companies via web services [2]. This requires that IT systems share information according to the reference ontology, and that they can interpret messages using that ontology.

However, *existing IT systems* are based on database schemas or class models that historically deviate from the reference ontology. In addition, we argue that even for *new IT systems*, it is likely that their class models or database schemas will deviate from the reference ontology as demonstrated by [3], [4]. The reason is that class models are task-specific, with the focus on an efficient implementation of an application. In contrast to reference ontologies, modeling decisions are geared towards a pragmatic and efficient model. Due to those differences, one often faces the situation where class models and reference ontologies are incompatible in the sense that a 1:1 mapping between them does not exist. Yet, it is required to use both the pragmatic class model and the reference ontology in parallel and to transform objects between class models and ontologies as illustrated by the Oil and Gas integration scenario. Practically, due to the dominant paradigm of object-oriented programming, this means that programming models supporting arbitrary mappings between class models and ontologies are needed. Despite the existence of sophisticated solutions for model mapping [5] and database integration [6], the existing approaches for mapping class models to ontologies are still limited with respect to supporting arbitrary mappings.

In this paper, we contribute a novel approach for bidirectionally mapping between reference ontologies and class models. As required for integrating *existing* IT systems, the approach is *non-intrusive*, i.e., it can be implemented without having to change the class model. The approach is *dynamic*, i.e., it does not rely on static 1:1 mappings between the class model and the reference ontology. Instead of using static mappings between the class model and the ontology, our approach uses rules which are dynamically evaluated at run-time.

We start by introducing a scenario of information integration in the area of emergency management in Section II. This scenario originates from building a prototype in the context of the German lighthouse project called *SoKNOS*.[1] The section also highlights typical deviations between class models and reference ontologies which go beyond a 1:1 mapping. Section III discusses how mappings for both directions are specified in the form of rules, followed by a discussion of the architectural design in Section IV. A scalability and performance evaluation can be found in Section V. Finally,

[1]Service-Oriented Architectures Supporting Networks of Public Security, www.soknos.de

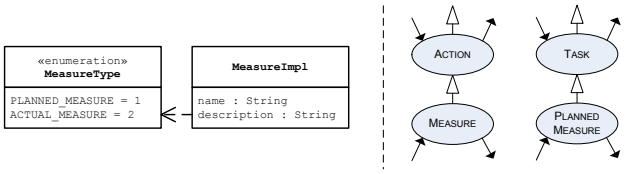

Fig. 1. A typical deviation between a class model (left) and a reference ontology (right) is a multi purpose class, i.e., a class used for different categories in the ontology.

we discuss related work and give a conclusion in Sections VI and VII, respectively.

II. Motivating Scenario

Emergency management is a domain where semantically unambiguous information exchange between organizations is crucial. Fighting large incidents like floodings or earthquakes requires close cooperation between a large number of organizations. If such an incident occurs, an emergency headquarters is put into place. The command staff operating the headquarters has the task to plan counter measures and to coordinate the joint efforts of all involved organizations. Since more and more organizations use IT systems, the challenge of system interoperability arises.

In the project SoKNOS, a prototype of an emergency management infrastructure has been developed, which integrates both newly developed and existing IT systems, ranging from geographical information providers to databases for managing resources. The SoKNOS system enables the command staff to quickly integrate new information services, such as map overlays or sensor information, but it also enables the integration of complete modules from other systems, such as modules displaying the currently available resources [7], [8].

For facilitating information exchange between the IT systems, we have developed a reference ontology of the emergency management domain [9], which defines the categories and relations of relevant terms in the emergency management domain, based on the formal top level ontology DOLCE [10].

The SoKNOS prototype includes a number of system components developed in Java, which use a simplified, pragmatic class model for information processing. Several deviations do occur between such a class model and a reference ontology. For example, *measures* in emergency management, such as extinguishing a particular fire, have to be planned and carried out. Therefore, both planned measures as well as measures that are actually performed have to be stored by an emergency management system. Fig. 1 shows a *multi purpose class*, a typical deviation between a pragmatic class model and a reference ontology: The class `Measure` contains a flag `type` indicating whether an instance is an actual or a planned measure. In the ontology, both belong in different categories which do not even have a meaningful common super category.

Information exchange in the SoKNOS prototype requires IT systems to represent and share information according to the reference ontology but residing in the corresponding object. Therefore, a mapping from objects to the ontology has to be put in place and vice versa. However, objects of a class cannot be mapped statically at design time. In our example, mapping an object of a class with either Measure or Planned Measure can only be performed at run-time by inspecting the `Measure` object and *dynamically* assigning a mapping, based on the value of the `type` flag.

Besides multi-purpose classes, there are a number of other deviations that can occur due to the differences of class models and reference ontologies:

- *Multi purpose relations*, just as multi purpose classes, are relations between objects that are used for expressing different relations in the ontology, distinguished by a flag or by implicit background knowledge, such as naming conventions of the related object. A typical example would be a String attribute `contactData`, which, if it contains an `@` symbol, is interpreted as an e-mail address, otherwise as a telephone number.
- *Artificial classes* are classes that do not have a corresponding category in the ontology, but hold some meaningful information, such as a class `CustomerData` containing both address and payment information.
- *Relation classes* are classes that do not correspond to a category, but to a relation in the ontology, e.g. a `Marriage` class corresponding to a MARRIED TO relation in the ontology.
- *Shortcuts* are direct relations between objects that only exist as indirect relations via intermediate categories in the ontology, e.g., a `Customer` object having a `ZIP code` attribute, while the connection between the Customer and the ZIP Code category involves the intermediate category City in the ontology.
- *Conditional objects* are objects whose existence is determined by a flag. Most commonly, this occurs as a `deleted` flag indicating that the object has been deleted.
- *Object counting* occurs if a set of relations to other objects is represented by the number of those objects, e.g., using an attribute `numberOfDamages` instead of a set of damage objects.
- *Non atomic attributes* are attributes that contain information belonging to several categories in the ontology. A typical example are attributes such as `address`, which contains both the street name and the street number. Those are often mapped to two different categories in a reference ontology.

III. Mapping Specification

Simply mapping one element from the class model to a category in the ontology is insufficient as we have learned in the previous section. Instead, arbitrary mappings between class models and ontologies are required which in turn necessitate an expressive representation formalism to specify the mappings. Therefore, some more versatile solution is needed, which allows for declaratively expressing dynamic mappings and interpreting those mappings at run-time.

We use *rules* for expressing dynamic mappings between class models and ontologies (and vice versa). The rules can

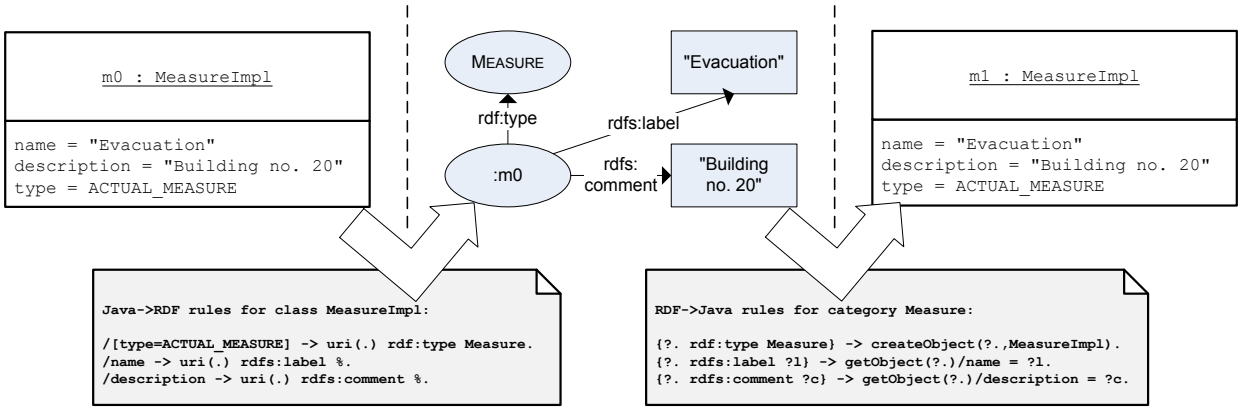

Fig. 2. Example for mapping rules between a class model and an ontology on a Java object.

be evaluated at run-time on objects to create the desired RDF graph describing the object, and on an RDF graph to create the corresponding set of objects. Fig. 2 shows how the mapping rules are used to transform a Java object into an RDF graph and back, using the example depicted in Fig. 1. The next sections explain the different rule syntaxes in detail.

A. Mapping Class Models to Ontologies

Our mapping approach uses tests on the objects to be mapped as rule bodies, and a set of RDF triples to be produced as rule heads. For each class, one rule set is defined. In cases where objects have relations to other objects, the rule sets of the corresponding related classes are executed when processing a related object. Rule sets defined for super classes are inherited to sub classes, however, the developer may also override inherited rules explicitly.

As the example depicted in Fig. 2 shows, the tests on the objects are defined using XPath expressions. The result of the expression can be used as a variable in the RDF triple to be generated (referred to with the `%` sign), as well as the current object (referred to with the `.` sign). The `uri` function used in a triple generates a unique URI for an object. The triples may also contain blank nodes, which are needed, e.g., to cope with shortcuts.

The XPath expressions may also contain regular expressions to deal with implicit background knowledge and non-atomic data types. Those can be used for conditions (e.g., does the `contactData` attribute contain an `@` symbol?) or for splitting data values, e.g., separating street names from numbers. A `repeat` function can be used to cover object counting deviations (e.g., produce a set of n blank nodes for an attribute value of n).

B. Mapping Ontologies to Class Models

The mapping rules from ontologies to class models are similar. Again, rule sets are defined per ontology category, are inherited to sub categories, and rules can be explicitly overridden by the developer.

For rule bodies, SPARQL expressions are used. In the rule heads, objects are created by using `createObject`, and attribute values for created objects are set (by using `getObject` and an XPath expression identifying the attribute to be set). The execution order of rules is defined such that all `createObject` statements are executed first, assuring that all objects are created before attempting to set attribute values.

For determining which categories an RDF instance belongs to, and for executing SPARQL statements, reasoning on the RDF graph and the domain ontology can be used. To cover non-atomic data types, the rules may use a `concat` function for concatenating different results of a SPARQL query. For coping with object counting, a `count` function can be used to produce the corresponding attribute values (once SPARQL version 1.1, which supports counting, becomes a standard, this will be obsolete).

Although the rules for both directions look similar, there is one subtle difference. The XPath expressions used on the object model are executed with closed world semantics, while the SPARQL expressions used on the RDF model are executed with open world semantics. The rationale is that the set of objects in an information system is completely known (and therefore forms a closed world), whereas an RDF graph representing a set of objects will typically only represent a subset of the original information.

IV. Architecture

Fig. 3 shows the architecture of our implemented prototype with two IT systems involved in an information exchange. As a first step, *Java→RDF* and *RDF→Java* mapping engines have to be put in place. As a second step, *Java→RDF rules* for mapping each object and *RDF→Java rules* for mapping each ontology category have to be specified, as discussed in the previous section. The rules have to registered in the corresponding mapping engine. At runtime, the IT systems are handled as black boxes with an API, following the paradigm of non-intrusiveness. When an object from an IT system is retrieved via its API and needs to be exchanged, an annotation in RDF is generated which conforms to reference ontology.

The exchange of objects between IT systems is controlled by *object exchange* components. They retrieve objects from and hand over objects to the IT systems' APIs, invoke the

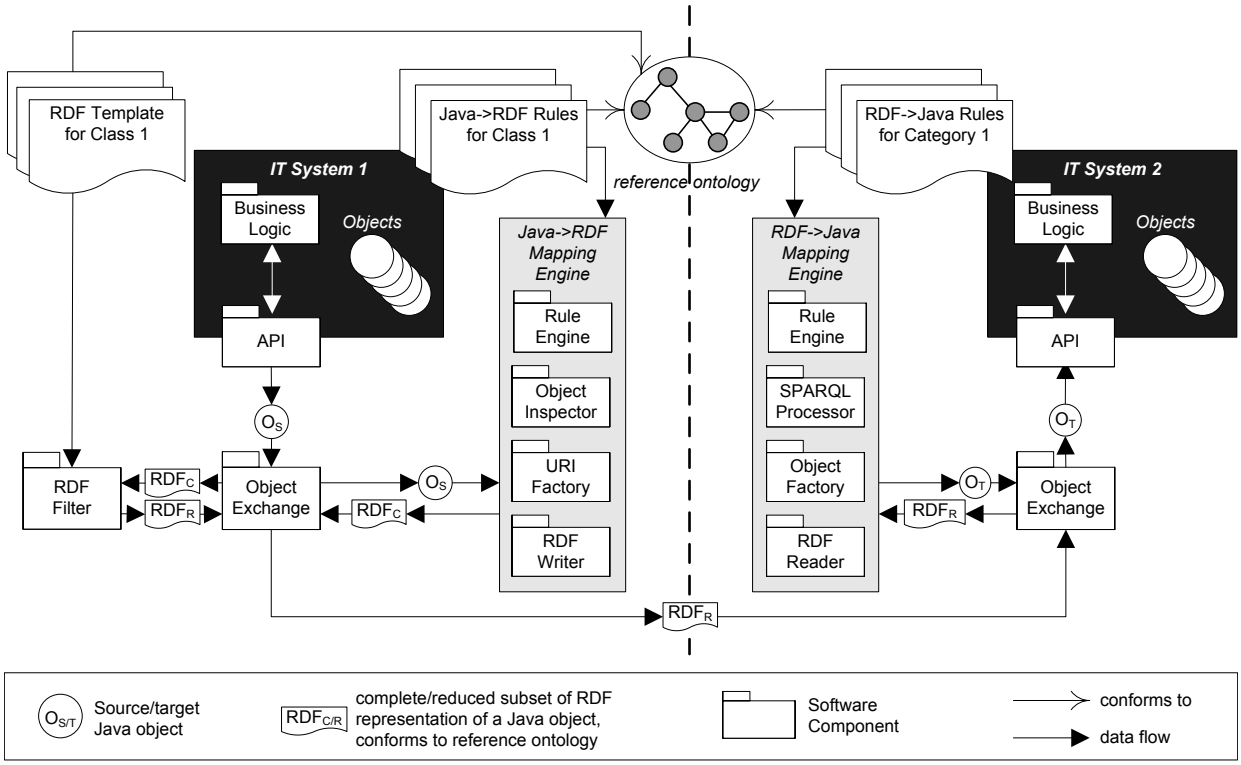

Fig. 3. Architecture of the implemented prototype, showing the object exchange between a sending (left) and a receiving (right) IT system.

corresponding mapping engines, and control the transfer to other object exchange components.

Let us assume *IT System 1* needs to exchange information with *IT System 2*. For generating the RDF representation, rules are processed by a *Rule Engine*. For executing rules, the *Object Inspector* evaluates conditions, and a *URI Factory* creates unique URIs for each object. An *RDF Writer*, unifies the outcomes of all relevant rules and creates the RDF annotation.

On the side of *IT System 2*, an object has to be created from the received RDF annotation. The *RDF Reader* first reads the received annotation, and another *Rule Engine* processes the mapping rules. A *SPARQL Processor* evaluates conditions, and an *Object Factory* eventually creates objects. The objects can then be used for calling IT System 2's API.

When exchanging objects, it may be desirable not to transfer all RDF statements that can be generated for a particular object, but only a subset. Since connected object graphs can be fairly large, the resulting RDF annotation can grow considerably in size, thus slowing down the transmission as well as the transformation back to objects. Therefore, an *RDF Filter* can be applied for reducing the set of RDF statements to be transmitted. The filter uses RDF templates which consist of blank nodes and labeled relations, according to the reference ontology. When applying a template to an RDF graph, only those resources are kept which are contained in the template.

Although the filtering could also be done by defining a subset of rules, we have chosen to keep both issues separated. The rules define *all* possible annotations that can be generated for a particular object and are therefore universal. The templates define the subset that is needed for a particular use case, e.g., transmission to another IT system. It is possible to use the same set of rules with a different set of templates for different use cases. For example, information exchange between systems owned by the same company might include more information, while the information exchanged with systems outside of the company is restricted.

We have implemented the solution sketched above in a Java-based prototype, using JXPath[2] for evaluating the XPath expression on Java objets and Jena [11] for RDF processing. The parsers for the rules were generated with JavaCC.[3] By using Java's reflection API [12] and relying on the Java Beans specification [13] (a naming convention for object constructors and for methods for accessing property values), the implementation is non-intrusive and does not require changes to the underlying class model.

V. Evaluation

We have evaluated our mapping prototype with the class model used in the SoKNOS prototype and the reference ontology of emergency management described in [9]. The SoKNOS class model consists of 58 classes with 30 attributes and 25 relations. The ontology consists of 156 categories and 136 relations. The number of occurrences of the deviations of the mapping between both are depicted in Table I. Thus, about 16% of all model elements (classes, attributes, and relations)

[2] http://commons.apache.org/jxpath/
[3] https://javacc.dev.java.net/

TABLE I
DEVIATIONS IN THE SOKNOS PROTOTYPE.

Deviation type	No. of Occurrences
Multi purpose class	3
Multi purpose relation	2
Artificial class	0
Relation class	1
Shortcuts	3
Conditional objects	2
Object counting	3
Non atomic attributes	4

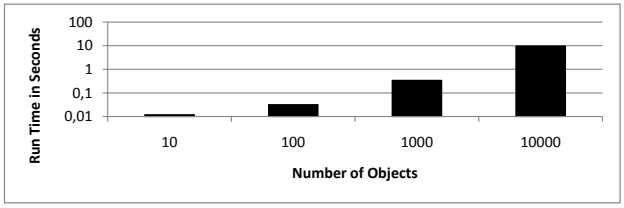

Fig. 4. Run time for creating the RDF annotation for a set of Java objects

and 6% of all ontology elements (categories and relations) are involved in some of the deviations.

Our mapping prototype was able to handle most of the deviations. The only problematic cases are non-atomic data types. In cases where a convention exists that can be formalized, such as "a street name never contains numbers, and the street number always starts with a digit between 1 and 9," our approach can handle them by using regular expressions. Cases where such a formalization cannot be found, such as for separating first and last names, cannot be handled fully automatically without human intervention.

We processed different, artificially created sets of objects from 10 to 10000 connected objects in order to test the performance and scalability of our approach.[4] Fig. 4 shows the processing times for producing RDF annotations for Java objects. The processing times scale linearly, the average time for processing one object is below one millisecond.

Fig. 5 shows the corresponding processing times for creating Java objects from RDF annotations. For processing SPARQL statements, different reasoning options can be used. We tested our approach with three built-in reasoners of Jena [11]: the *transitive reasoner*, which only uses `rdfs:subClassOf` and `rdfs:subPropertyOf` statements, the *RDFS reasoner*, and the *OWL reasoner*, which covers OWL Lite reasoning. Except for the latter, which creates some overhead for larger sets of objects, creating an object from RDF takes less than ten milliseconds, and the processing times scale linearly.

VI. RELATED WORK

In this paper, we have discussed an approach for creating annotations for Java objects, and for re-constructing Java objects from such annotations. Such approaches can be roughly categorized into *generative*, *intrusive*, and *non-intrusive* approaches. The latter are the only approaches that

[4]The tests were run on a Windows PC with a Pentium 3.6GHz dual core processor and 2GB of RAM.

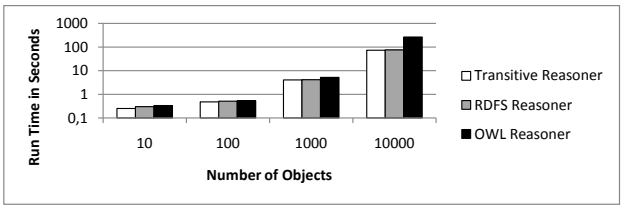

Fig. 5. Runtime for creating Java objects from a set of annotations, using different types of reasoning

can be applied when the developer cannot or must not change the class model, be it for technical or for legal reasons.

Generative approaches create a class model from an ontology. They can thus be seen as a special case of model driven architecture (MDA) [14]. Examples for generative approaches are *RDFReactor* [15], *OWL2Java* [16], *OntoJava* [17], and *agogo* [18]. As a special case, the class model may also be generated on-the-fly at run-time when using dynamically typed scripting languages, as shown with *ActiveRDF* [19] for Ruby and *Tramp*[5] for Python. For the developer, such approaches are more comfortable, since they do not require additional tools such as a code generator, but work directly out of the box by invoking an API. In general, generative approaches usually create one Java class per ontology category, therefore, a 1:1 mapping exists by definition. Generative approaches are a suitable solution for developing *new* IT systems, but not for integrating *existing* ones.

Intrusive approaches do not generate new Java classes from an ontology, but modify (i.e., intrude into) an existing class model by adding additional code fragments, such as Java annotations[6] or additional methods and/or attributes. Examples for intrusive approaches are *otm-j* [20], *Sommer*,[7] *Texai KB* [21], *RDFBeans*,[8] *Jenabean*,[9] and *P-API* [22]. All those approaches require a 1:1 mapping between Java classes and categories in the ontology.

Non-intrusive approaches can produce annotations without altering the class model. The only examples we found are *ELMO*[10] and the EMF-RDF transformations introduced in [23]. Both approaches require that the "the domain model should be rather close to the ontology" [23] and cannot cope with most of the deviations discussed above.

While some of the approaches also cover both directions of conversion – from Java to RDF and from RDF to Java – and can cope with *some* of the deviations discussed above, to the best of our knowledge, there are no solutions that can cope with all the deviations and can therefore support *arbitrary* mappings between class models and reference ontologies.

[5]`http://www.aaronsw.com/2002/tramp/`

[6]Java annotations are a language construct in Java and must not be confused with *semantic* annotations, as used throughout this paper. See `http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html`.

[7]`https://sommer.dev.java.net/`

[8]`http://rdfbeans.sourceforge.net/`

[9]`http://code.google.com/p/jenabean/`

[10]`http://www.openrdf.org/doc/elmo/1.5/`

VII. Conclusion and Outlook

We have discussed a number of typical deviations between class models and reference ontologies. In a use case from the emergency management domain, we have shown that the deviations do occur in actual IT systems and ontologies. We argue that also other domains are affected by such deviations, e.g., the Oil and Gas domain.

We have introduced an approach for mapping class models to ontologies. Our approach uses rules both for creating RDF representations of objects as well as vice versa, thus supporting information exchange between Java objects as RDF between potentially heterogeneous class models.

A prominent area of using ontologies is facilitating interoperability between heterogeneous information systems by allowing for semantically unambiguous information exchange. As this may involve IT systems that cannot or must not be altered for technical or legal reasons, our approach has been implemented in a *non-intrusive* way. It supports *arbitrary mappings* between class models and ontologies. We have further evaluated the performance and scalability of our approach, showing that the transformations are performed within milliseconds, growing linearly with the number of objects.

The paper has focused on the use case of information integration and exchange. However, the mechanism of mapping different class models to one common ontology may also be used to make the data in different applications available as a unified linked data set, allowing for reasoning and for unified visualization [24].

Currently, the developer has to develop the mapping rules by hand. In the future, techniques developed, e.g., in the field of ontology matching [25] may be employed to assist the user, however, the state of the art in ontology matching is most often focused on discovering 1:1 mappings.

References

[1] J. W. Kluewer, M. G. Skjæveland, and M. Valen-Sendstad, "ISO 15926 templates and the Semantic Web," Position paper for W3C Workshop on Semantic Web in Energy Industries; Part I: Oil and Gas, http://www.w3.org/2008/11/ogws-papers/kluwer.pdf, 2008.

[2] F. Verhelst, F. Myren, P. Rylandsholm, I. Svensson, A. Waaler, T. Skramstad, J. Ornæs, B. Tvedt, and J. Høydal, "Digital Platform for the Next Generation IO: A Prerequisite for the High North," in *SPE Intelligent Energy Conference and Exhibition*, 2010.

[3] R. Credle, V. Akibola, V. Karna, D. Panneerselvam, R. Pillai, and S. Prasad, "Discovering the Business Value Patterns of Chemical and Petroleum Integrated Information Framework," IBM, Red Book SG24-7735-00, August 2009.

[4] F. Myren, U. Pletat, and J. W. Kluewer, "Model driven integration architecture for IO G2 information – Reference Semantic Model alignment to ISO 15926," Semantic Days 2009, 18 - 20 May, Clarion Hotel Stavanger, https://www.posccaesar.org/wiki/PCA/SemanticDays2009, Session 6: Semantic technology for IO Generation 2, 2009.

[5] P. A. Bernstein and S. Melnik, "Model Management 2.0: Manipulating Richer Mappings," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 1–12.

[6] A. Doan and A. Y. Halevy, "Semantic Integration Research in the Database Community: A Brief Survey," *AI Magazine*, vol. 26, no. 1, pp. 83–94, 2005.

[7] S. Döweling, F. Probst, T. Ziegert, and K. Manske, "SoKNOS - An Interactive Visual Emergency Management Framework," in *GeoSpatial Visual Analytics*, ser. NATO Science for Peace and Security Series C: Environmental Security, R. D. Amicis, R. Stojanovic, and G. Conti, Eds. Springer, 2009, pp. 251–262.

[8] H. Paulheim, S. Döweling, K. Tso-Sutter, F. Probst, and T. Ziegert, "Improving Usability of Integrated Emergency Response Systems: The SoKNOS Approach," in *Proceedings "39. Jahrestagung der Gesellschaft für Informatik e.V. (GI) - Informatik 2009"*, ser. LNI, vol. 154, 2009, pp. 1435–1449.

[9] G. Babitski, F. Probst, J. Hoffmann, and D. Oberle, "Ontology Design for Information Integration in Catastrophy Management," in *Proceedings of the 4th International Workshop on Applications of Semantic Technologies (AST'09)*, 2009.

[10] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari, "WonderWeb Deliverable D18 – Ontology Library (final)," 2003, http://wonderweb.semanticweb.org/deliverables/documents/D18.pdf. Accessed August 2nd, 2010. [Online]. Available: http://wonderweb.semanticweb.org/deliverables/documents/D18.pdf

[11] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: Implementing the Semantic Web Recommendations," in *Proceedings of the 13th international conference on World Wide Web - Alternate Track Papers & Posters*, S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, Eds. ACM, 2004, pp. 74–83.

[12] I. R. Foreman and N. Forman, *Java Reflection in Action*, ser. In Action. Manning Publications, 2004.

[13] R. Englander, *Developing Java beans*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1997.

[14] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, "Model-driven architecture," in *Advances in Object-Oriented Information Systems, OOIS 2002 Workshops, Montpellier, France, September 2, 2002, Proceedings*, ser. LNCS, vol. 2426. Springer, 2002, pp. 290–297.

[15] M. Völkel and Y. Sure, "RDFReactor - From Ontologies to Programmatic Data Access," in *Posters and Demos at International Semantic Web Conference (ISWC) 2005, Galway, Ireland*, 2005. [Online]. Available: http://www.xam.de/2005/11_voelkel_iswc2005-rdfreactor3-demo.pdf

[16] A. Kalyanpur, D. J. Pastor, S. Battle, and J. A. Padget, "Automatic Mapping of OWL Ontologies into Java," in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004), Banff, Alberta, Canada, June 20-24, 2004*, F. Maurer and G. Ruhe, Eds., 2004, pp. 98–103.

[17] A. Eberhart, "Automatic Generation of Java/SQL Based Inference Engines from RDF Schema and RuleML," in *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, ser. Lecture Notes in Computer Science, I. Horrocks and J. A. Hendler, Eds., vol. 2342. Springer, 2002, pp. 102–116.

[18] F. S. Parreiras, C. Saathoff, T. Walter, T. Franz, and S. Staab, "APIs à gogo: Automatic Generation of Ontology APIs," in *Proceedings of the International Conference on Semantic Computing*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 342–348.

[19] E. Oren, R. Delbru, S. Gerke, A. Haller, and S. Decker, "ActiveRDF: object-oriented semantic web programming," in *WWW*, C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, Eds. ACM, 2007, pp. 817–824.

[20] M. Quasthoff and C. Meinel, "Design Pattern for Object Triple Mapping," in *2009 IEEE International Conference on Services Computing (SCC 2009), 21-25 September 2009, Bangalore, India*. IEEE Computer Society, 2009, pp. 443–450.

[21] S. Reed, "Semantic Annotation for Persistence," in *Proceedings of AAAI2007's Workshop on Semantic e-Science*, 2007.

[22] J. Wagner, F. Babi, and P. Bednar, "Java RDF framework for knowledge repository," in *7th International Symposium on Applied Machine Intelligence and Informatics (SAMI 2009)*, jan. 2009, pp. 99 –102.

[23] G. Hillairet, F. Bertrand, and J. Y. Lafaye, "Bridging EMF applications and RDF data sources," in *Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering (SWESE)*, E. F. Kendall, J. Z. Pan, M. Sabbouh, L. Stojanovic, and K. Bontcheva, Eds., 2008.

[24] H. Paulheim and L. Meyer, "Ontology-based Information Visualization in Integrated UIs," in *Proceedings of the 2011 International Conference on Intelligent User Interfaces (IUI)*, 2011, to appear.

[25] J. Euzenat and P. Shvaiko, *Ontology Matching*. Berlin, Heidelberg, New York: Springer, 2007.