A Meta-Framework for Generating Ontologies from Legacy Schemas*

Manuel Wimmer
Vienna University of Technology
Austria
wimmer@big.tuwien.ac.at

***Abstract*—A huge amount of schemas are expressed within outdated schema languages which are restricted concerning expressiveness, extensibility, readability, and understandability. Consequently, the actual intention of the schema developers is hard to grasp. Reverse engineering approaches try to tackle this problem by automatically transforming legacy schemas into ontologies, but rarely enhance the semantics of the schemas by exploiting the higher expressiveness of modern schema languages.**

Therefore, we propose a meta-framework for generating ontologies from legacy schemas going beyond existing approaches. This meta-framework is instantiable for various schema reverse engineering scenarios and allows to generate ontologies with improved structures and semantics compared to the original legacy schemas by exploiting the advanced expressiveness of modern schema languages. Finally, this meta-framework allows for the automatic migration of data from the legacy schemas into instances of the generated ontologies.

I. INTRODUCTION

Schemas are the first choice for describing data structures of applications since the introduction of database systems. The first generation of languages used for describing schemas, such as the relational model [1], was typically focused on the efficient implementation of data structures. However, soon it has been realized that also a conceptual view is necessary [2] to develop larger schemas, thus various conceptual schema languages have been proposed which influenced not only data engineering but also software engineering in general. Based on the origins and evolution, huge differences concerning the expressiveness, extensibility, readability, and understandability between practically used schema languages exist. Summarizing, modern schema languages offer various benefits for developing large schemas. For example, schemas expressed in UML class diagrams [3] provide sophisticated reuse mechanisms based on inheritance and composition, rich type systems, explicit relationships between elements, and furthermore, complex constraints are described with a dedicated constraint language [4], thus schemas become more and more like ontologies [5]. In the context of this paper, we use the term ontology for schemas which provide an explicit representation of all involved concepts, relationships, and constraints. In contrast, the term legacy schema is used for schemas which lack such an explicit representation.

In practice, explicit representations in form of ontologies are highly needed for integration concerns such as unification of schemas, for migrating data from one schema to another, and for extending schemas. For all these tasks, an explicit and precise representation is necessary in order to reconstruct the intentions of the schema developers. For this, switching the schema language and exploiting a visual representation is in general not sufficient, especially for large schemas. However, most current reverse engineering approaches support only such representation switches, which are necessary as a first step but are not sufficient as a final goal. Furthermore, the approaches are limited to a specific combination of schema languages, e.g., relational schemas to UML class diagrams, but no general reverse engineering methodology exists, which can be applied for several scenarios.

To tackle these problems, we propose a meta-framework which can be applied to various schema reverse engineering scenarios. The meta-framework provides a semi-automatic reverse engineering process consisting of two general phases. The first phase concerns the automatic transition from the legacy schema language to the ontology language, whereas the second phase supports the user by the semantic enrichment of the initial ontology in order to resolve deficiencies of the legacy schema language. This meta-approach has been extracted from several successful reverse engineering activities of the ModelCVS project [6] and the MDWEnet project [7]. The benefits of applying this meta-framework are: (1) once the legacy schema language is aligned with the ontology language, each schema conforming to the schema language is representable as an ontology, (2) the semi-automatic generation process allows not only a switch in the representation, but also an improvement of the structure and of the semantics of schemas, and finally, (3) the systematic generation of the ontology allows that data of the legacy schemas is automatically migrated into ontology instances of the generated ontology.

The remainder of this paper is structured as follows. Section 2 proposes the conceptual meta-framework by discussing the steps required to come from an implicit legacy schema to an explicit ontology. Section 3 elaborates on architectural concerns for a reverse engineering tool support-

* This work has been partly funded by the Austrian Science Fund (FWF) under grant P21374-N13.

ing the conceptual meta-framework. Section IV investigates related work and finally, Section V concludes this paper with an outlook on future work.

II. A Meta-Framework for Reverse Engineering of Legacy Schemas

In this section the meta-framework for generating ontologies from legacy schemas is presented on a conceptual level.

A. *Two-phase Generation Process at a Glance*

For producing semantic-rich ontologies from legacy schemas, we propose a semi-automatic process for overcoming the limitations of existing automatic generation approaches. The process encompasses two phases as illustrated in Figure 1. During the first phase a preliminary version of an ontology is automatically generated from the available legacy schema, while in the second phase this preliminary version is semantically enriched according to constraints not captured by the legacy schema as well as structurally improved by using dedicated features of the ontology language.

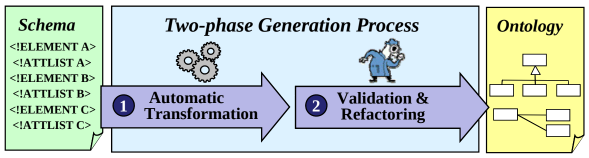

Figure 1. Two-phase generation process

As running example throughout this paper, the generation of *UML class diagrams* from *Document Type Definitions* (DTDs) is used to exemplify the steps of the proposed meta-framework. But before we are actually going into details about the meta-framework, the involved artifacts of the reverse engineering process are described and aligned based on the OMGs meta-layer architecture [8].

B. *Meta-Layers Involved*

In Figure 2, the meta-layers of the reverse engineering process are shown. On the lowest level, we have the so-called *Instance Layer* (cf. I in Figure 2) where the instances, i.e., the data, conforming to schemas resides. The middle layer is the so-called *Schema Layer* (cf. S in Figure 2) where schema definitions are located. And finally, on top we have the *Language Layer* (cf. L in Figure 2) where the schema language definitions reside. Please note that each artifact on layer *N* has to conform to a specification on layer *N+1*, e.g., the schema has to conform to the schema language, and that any number of artifacts on layer *N* may be instantiated from a specification on layer *N+1*, e.g., data conforming to a schema may be instantiated as often as needed.

In this meta-layer architecture, the artifacts of the reverse engineering process are arranged. On the left hand side of Figure 2, the legacy schema on layer S, its corresponding

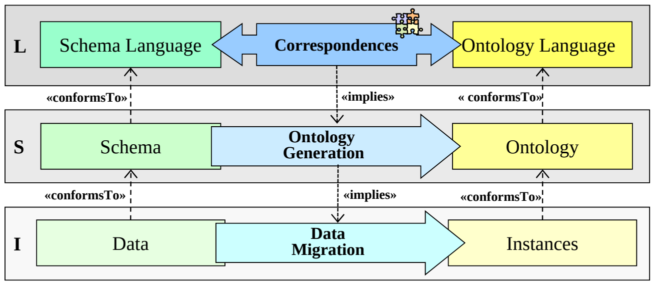

Figure 2. Reverse Engineering Meta-Layers

schema language on layer L, and its instantiated data on layer I are shown. On the right hand side of Figure 2, we have only one given artifact, namely the ontology language on layer L. The rest of the right hand side artifacts need to be generated by the ontology generation process and data migration processes located in the middle of Figure 2. Before these artifacts can be generated, the correspondences between the schema language and the ontology language have to be defined. By bridging the language definitions on layer L, it is possible to derive a transformation which is capable of generating initial ontologies from any legacy schema conforming to the bridged schema language. Furthermore, the correspondence definitions on layer L allow to migrate arbitrary data conforming to already transformed schemas as instances conforming to the generated ontologies. This means, the correspondences are the key for the required transformations on the layers below.

Considering our running example, when we bridge the DTD language with the UML language on layer L, by defining correspondences, e.g., between *Entity Type* from DTD and *Class* from UML, a DTD, e.g., the (X)HTML recommendation or an application-specific schema, can be automatically transformed into an UML class diagram. Furthermore, an XML document conforming to the DTD, e.g., an (X)HTML document or a document of the application-specific schema, can be transformed into instances of the UML class diagram called object diagrams [3].

C. *From Implicit to Explicit Representations*

The correspondences on layer L are the first ingredients for the reverser engineering process. After defining how each language element of the legacy schema language corresponds to an ontology language element, the schema definitions are transformable into initial ontologies. Correspondences are for example defined in a correspondence table and then implemented as transformation rules.

By defining only unambiguous correspondences, one only gain a switch in the representation which often allows for a graphical representation of the schemas, but the quality of the schema in terms of structural properties or semantic enhancement is not achieved. Therefore, in addition to unambiguous transformation rules, we need further mechanisms to improve the generated ontology. At this point, it has to be decided between automatic and manual improvements. Considering our running example, UML offers inheritance between classes which is not available in DTDs. For enhancing the schema definition, we would like to introduce inheritance to gain a more compact representation. Now, we have to decide: is it better to automatically explore possible inheritance structures or should the user decide where to introduce inheritance? The answer to this question may dependent on the specific schemas. When a schema is used where classes which have overlapping attributes should also have a taxonomic relationship, this may be automatically generated. However, cases exist where this approach would lead to unintended schema specifications. For example, if we have a class *Professor* and a class *Bottle*, which both have an attribute *name* and an attribute *age*. Although the subclasses do not have a taxonomic relationship, a common superclass would be generated. For deciding between automatic or manual enhancements, the ratio between true positives and false negatives of automatic improvement rules has to be considered.

For supporting both possibilities, the proposed meta-framework provides two steps for enhancing the quality of the schemas. First, heuristics are introduced which tackle deficiencies of the legacy schema language by automatically improving the generated initial ontology. The heuristics are automatically applied and are therefore located in the first phase of the meta-framework as is also indicated in Figure 3. Applied heuristics should be documented in the initial ontology, e.g., by using annotation mechanisms, in order to give the user the chance to validate the correctness of the heuristic applications. Considering again our running example, a possible heuristic is to transform each attribute of type *Enumeration* with values *on* and *off* from the DTD into a Boolean attribute of the corresponding UML class diagram with an annotation ≪*Boolean attribute identified*≫.

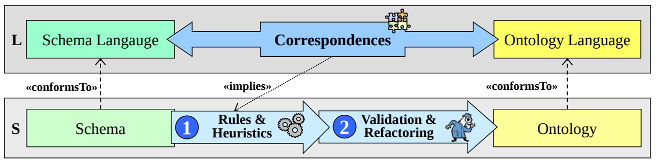

Figure 3. Operationalization of the Two-phase Process

On the basis of the transformation rules and heuristics, in the second phase, the user needs to manually validate the heuristics applications within the generated ontology and refactor it accordingly in order to resolve the remaining deficiencies which cannot be resolved by heuristics. For cases where a high amount of false positives of heuristic applications is expected or in cases where user interaction is necessary to enhance the semantics and structure of the initial ontology, refactoring patterns should be provided. The first source for refactorings are features of the ontology language which are missing in the legacy schema language. Considering the running example, one may define refactorings for introducing inheritance relationships between classes. Therefore, refactoring operations can be introduced which origin from object-oriented refactoring patterns [9] such as introduce abstract superclass, shift attributes to superclass and so on. The second source for refactorings are missing constraints which have not been expressed in the legacy schema. Such cases are typically hard to identify when examining only the schemas. Therefore, often additional information resources have to be inspected, e.g., additional declarative specifications or source code of programs which use the schemas. However, in general it is tedious to derive the necessary constraints using such a white-box approach. A black-approach would be to test the applications in an explorative manner or to study documentations such as user handbooks.

In order to verify that the ontology is properly semantically enriched, a list of deficiencies of the legacy schema language as well as lists of defined heuristics and refactoring patterns should be established. Then, each entry of the deficiencies list must be at least mapped to a heuristic or to a refactoring pattern. This is required to verify that no deficiencies of the legacy schema remain in the generated ontology.

III. Tool Support

Because reverse engineering is a tedious and error-prone task, tools support is inevitable. Therefore, we discuss in this section, how tools are constructed to support the proposed meta-framework. First, we present a tool architecture for reverse engineering of schemas, and second, for supporting also cases where it is necessary to migrate data from legacy schemas to ontology instances, we elaborate on a tool architecture for automatically migrating instances based on the schema transformation.

A. *Ontology Generation Architecture*

In this subsection, we elaborate on the core components for the reverse engineering of schemas. Figure 4 shows the details of a possible conceptual architecture. The architecture is divided into two areas according to the two-phase generation process. In a first step a specific parser, e.g., a DTD parser in the running example, builds an object graph of the legacy schema. Then each node in the object graph is visited and transformed according to the transformation rules and heuristics. As soon as the complete object graph of the ontology has been generated, the default serializer of the ontology language is activated in order to serialize the ontology, e.g., as an XML file. This file may be loaded into existing graphical ontology editors. In the second phase, the annotations which indicate the application of a heuristic are validated by the user and the ontology is refactored accordingly. The editor should provide additional functionality such as the direct navigation to the heuristic annotations as well as their convenient acceptance and rejection. The refactoring patterns should be available as refactoring operations within the editor to allow their convenient application and their propagation to the instance level which is elaborated in the next subsection.

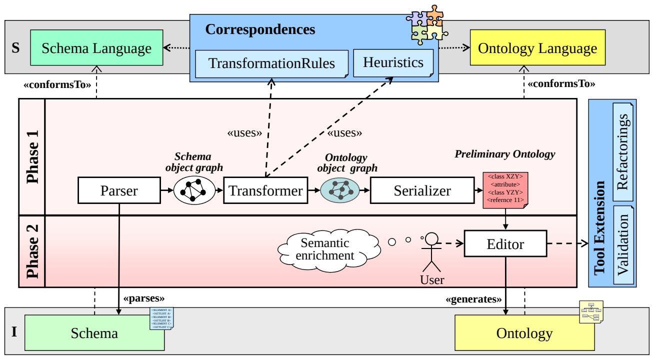

Figure 4. Ontology Generation Architecture

B. Data Migration Architecture

After presenting the tool architecture for the ontology generation, we have now the background to discuss the architecture for data migration which is also divided into two areas (cf. Phase 1 and Phase 2 in Figure 5). The prerequisites for migrating data from the legacy schema to ontology instances are the automatic transformation rules and heuristics (cf. Step 1 in Figure 5) as well as the validation of applied heuristics and the employed refactoring operations (cf. Step 2 in Figure 5). In particular, the automatic transformation of the schema implies how the data should be transformed into initial ontology instances (cf. Step 3). This dependency is not problematic, because the rules and heuristics are predefined which allows to provide a generic component for transforming data to ontology instances conforming to the automatically generated ontology. The more problematic dependency is the second one, namely between the validation and refactoring of the initial generated ontology. This dependency requires proper adaptation (cf. Step 4)) of the initial instances generated by Step 3 in order to conform to the reworked ontology which leads to the field of co-evolution [10].

In order to provide co-evolution rules for instances, it must be ensured that the generated initial ontology is systematically changed by predefined refactoring patterns and not by arbitrary modifications. The reason for this restriction is the fact that for predefined refactoring patterns, rules can be derived for co-evolving instances. Regarding to their impact on the instances, two main categories of refactoring patterns can be distinguished.

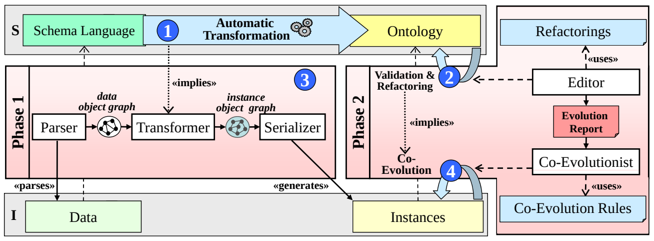

Figure 5. Data Migration Architecture

(1) Refactorings without impact on instances: Several refactorings are only restructuring the ontologies without influencing their corresponding instances. For example, introducing an abstract class for collecting common properties has no impact on the automatically generated ontology instances. Therefore, no co-evolution rules have to be provided.

(2) Refactorings with impact on instances: Of course, some refactorings have an impact on the instance level. For example, if new ontology elements are introduced which can be instantiated (in contrast to the previous example of the abstract class), or if ontology elements are replaced, co-evolution rules are necessary. This category can be further divided.

(2a) Automatically derivable: Some co-evolution rules are automatically derived from the refactoring patterns and specified as transformation rules. Consider our running example, when an ID/IDREF attribute pair of a DTD is replaced by a typed relationship between two classes within the corresponding UML class diagram, a general rule can handle the co-evolution of the instances by replacing the attribute values by a link between the instances.

(2b) User defined: Some co-evolution rules have to be at least partly defined by the user. For example, if an attribute named *mark* of type *Integer* is refactored as an attribute of type *Enumeration* with values {*A,B,C,...*} then the user has to provide value mappings, e.g., in the form *if 1 then A*, to ensure an appropriate co-evolution of the ontology instances.

In summary, for migrating data, first a component for supporting Step 3 is required in order to generate initial ontology instances which is again a transformation component based on the correspondences between the legacy schema language and the ontology language. Second, a component supporting Step 4, namely the co-evolution of the generated initial instances, must be available which takes a summary of the applied refactorings (cf. Evolution Report in Figure 5) as input and activates the necessary co-evolution rules on the ontology instances.

IV. Related Work

Due to space limitations, only the most related work to the presented meta-framework is discussed, although we are aware that a huge amount of reverse engineering approaches exist which are bound to specific schema language combinations, e.g., cf. [11], [12], [13] for approaches and cf. [14], [15], [16] for surveys.

The most closely related work is the *ModelGen* operator proposed by Bernstein [17] in the field of model management. The basic idea of model management is to provide a high-level language which is based on a set of generic operators such as *ModelGen*, *Match*, *Merge*, or *Diff*. These operators can be composed in so-called scripts in order to solve more complex integration scenarios. In particular, the ModelGen operator is used when a source schema *S1* defined in terms of the data model *M1* is given, as well as a target data model *M2*. The ModelGen operator is capable of producing a target schema *S2* defined in terms of data model *M2* which semantically corresponds to the source schema *S1*. Furthermore, also instances of source schema *S1*, i.e., the data, is transformed to instances of source schema *S2*. Atzeni et al. [18], [19] provided an implementation of the ModelGen operator regarding commonly used data models such as ER, UML class diagrams, and the relational data model. For the implementation of ModelGen, the authors decided to use a so-called supermodel, a model that integrates the modeling constructs of commonly used data models and acts as a "pivot" model. In addition, adapters are used to transform ER, UML class diagrams, and relational models into the supermodel formalism. If one wants to transform an ER model into a relational model, then the ER model is transformed with simple one-to-one translations into a supermodel model, then a set of generic operators is applied on the model as long as it only uses concepts of the relational data model, e.g., many-to-many relationships are transformed into additional relations, and finally, the model may be translated with a simple one-to-one transformation into a relational model. Although our meta-framework also exploits the fact that correspondences on the language layer can be used for transforming schemas on the next lower layer as well as for transforming instances of those schemas, we are aiming at the semantic enrichment of the schemas. In particular, in the second phase of our meta-framework, the user has to incorporate constraints in the schemas which have not been captured in the original schema definitions. Therefore, in addition to unambiguous transformation rules, we are using heuristics and refactorings to improve the design and precision of the schemas.

V. Conclusion and Future Work

In this paper we have presented a meta-framework for reverse engineering of schemas into ontologies, which is applicable in several scenarios. The presented meta-framework has been proven useful in various integration scenarios [20], [6], [21]. Promising results have been achieved for reverse engineering of languages supported by CASE tools and standards such as (X)HTML, but also for schemas of legacy web applications. One of the most comprehensive case study was the creating a metamodel for WebML from the accompanying tool WebRatio, where the language has been implemented with several DTDs [22]. By following the presented meta-framework, major improvements of the quality of the language description have been achieved. For example, the overall amount of elements used to describe the language WebML has been reduced from 707 to 487. This quality improvements have been useful in subsequent projects in which WebML has been integrated with other web modeling languages [23] and extended with aspect-oriented modeling concepts [24].

Several issues remain open for future work. We plan to establish a generic tool platform which provide adapters for several schema languages, a high-level transformation language for implementing the schema transformations as well as the refactoring patterns, and finally a generic co-evolution tool should be integrated which allows to define the co-evolution rules with an appropriate domain-specific language. For developing this platform, we plan to employ model-based technologies which show a high unification potential [25] as well as an appropriate abstraction level.

References

[1] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[2] P. P. Chen, "The entity-relationship model - toward a unified view of data," *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, 1976.

[3] OMG, "UML Specification: Superstructure Version 2.0," http://www.omg.org/docs/formal/05-07-04.pdf, August 2005.

[4] ——, "OCL Specification Version 2.0," http://www.omg.org/docs/ptc/05-06-06.pdf, June 2005.

[5] T. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[6] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer, "Lifting Metamodels to Ontologies - A Step to the Semantic Integration of Modeling Languages," in *Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, 2006.

[7] A. Vallecillo, N. Koch, C. Cachero, S. Comai, P. Fraternali, I. Garrigós, J. Gómez, G. Kappel, A. Knapp, M. Matera, S. Meliá, N. Moreno, B. Pröll, T. Reiter, W. Retschitzegger, E. Rivera, A. Schauerhuber, W. Schwinger, M. Wimmer, and G. Zhang, "MDWEnet: A Practical Approach to Achieving Interoperability of Model-Driven Web Engineering Methods," in *Workshop Proceedings of 7th International Conference on Web Engineering (ICWE'07), Italy, July*, 2007.

[8] OMG, "MDA Guide Version 1.0.1," http://www.omg.org/docs/omg/03-06-01.pdf, June 2003.

[9] M. Fowler, *Refactorings*, 1st ed. Addison-Wesley, 1999.

[10] J. Banerjee, W. Kim, H.-J. Kim, and H. F. Korth, "Semantics and implementation of schema evolution in object-oriented databases," *SIGMOD Rec.*, vol. 16, no. 3, pp. 311–322, 1987.

[11] J.-L. Hainaut, C. Tonneau, M. Joris, and M. Chandelon, "Schema Transformation Techniques for Database Reverse Engineering," in *Proc. of the 12th International Conference on the Entity-Relationship Approach (ER'93)*. Springer, 1994, pp. 364–375.

[12] G. Booch, M. Christerson, M. Fuchs, and J. Koistinen, "UML for XML Schema Mapping Specification," Rational Software and CommerceOne, Tech. Rep., August 1999.

[13] M. Necasky, "Reverse Engineering of XML Schemas to Conceptual Diagrams," in *Proc. of the 6th Asia-Pacific Conference on Conceptual Modelling (APCCM'09)*, 2009.

[14] K. H. Davis and P. H. Aiken, "Data Reverse Engineering: A Historical Survey," in *Proc. of the 7th Working Conference on Reverse Engineering (WCRE'00)*. IEEE Computer Society, 2000, p. 70.

[15] M. Bernauer, G. Kappel, and G. Kramler, "Representing xml schema in uml - a comparison of approaches," in *Proc. of the 4th International Conference on Web Engineering (ICWE'04)*. Springer, 2004, pp. 440–444.

[16] A. Yu, "An Overview of Research on Reverse Engineering XML Schemas into UML Diagrams," in *Proc. of the 3rd International Conference on Information Technology and Applications (ICITA'05)*. IEEE Computer Society, 2005, pp. 772–777.

[17] P. A. Bernstein, "Applying model management to classical meta data problems," in *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR'03), California*, 2003.

[18] P. Atzeni, P. Cappellari, and P. A. Bernstein, "ModelGen: Model Independent Schema Translation," in *Proceedings of the 21st International Conference on Data Engineering (ICDE'05), Tokyo, Japan*, 2005.

[19] P. Atzeni, P. Cappellari, and G. Gianforme, "MIDST: model independent schema and data translation," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China*, 2007.

[20] A. Schauerhuber, M. Wimmer, and E. Kapsammer, "Bridging Existing Web Modeling Languages to Model-Driven Engineering: A Metamodel for WebML," in *Proceedings of the 2nd International Workshop on Model-Driven Web Engineering (MDWE'06), Palo Alto, California, USA*, July 2006.

[21] M. Wimmer and G. Kramler, "Bridging Grammarware and Modelware," in *Proceedings of Satellite Events at the MoDELS 2005 Conference*. Springer, 2005, pp. 159–168.

[22] A. Schauerhuber, M. Wimmer, E. Kapsammer, W. Schwinger, and W. Retschitzegger, "Bridging WebML to Model-Driven Engineering: From Document Type Definitions to Meta Object Facility." *IET Software*, vol. 1, no. 3, pp. 81–97, 2007.

[23] M. Wimmer, A. Schauerhuber, W. Schwinger, and H. Kargl, "On the Integration of Web Modeling Languages: Preliminary Results and Future Challenges," in *Workshop Proceedings of 7th International Conference on Web Engineering (ICWE'07)*, 2007.

[24] A. Schauerhuber, M. Wimmer, W. Schwinger, E. Kapsammer, and W. Retschitzegger, "Aspect-Oriented Modeling of Ubiquitous Web Applications: The aspectWebML Approach," in *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), Tucson, Arizona, USA*, Mar. 2007, pp. 569–576.

[25] J. Bézivin, "On the unification power of models," *Software and System Modeling*, vol. 4, no. 2, pp. 171–188, 2005.