MULTI-PLATFORM MODEL-DRIVEN SOFTWARE DEVELOPMENT OF WEB APPLICATIONS

Ulrich Wolffgang
Department of Information Systems, University of Muenster, Leonardo-Campus 3, Muenster, Germany

Keywords: Model-driven web engineering, MDA, MDSD, CIM, PIM, PSM, WASL.

Abstract: We present the generator framework WASL, which supports model-driven development of web applications. It enables the modeling of data, navigation, business logic, and presentation logic. WASL supports the MDA by providing transformation definitions from a computation-independent model (CIM) to a platform-independent model (PIM) and then to a platform-specific model (PSM). An important aspect of our approach is that we provide a separate DSL for each target platform in order to separate concerns between PIM and PSM and to be able to address platform-specific aspects. Currently, the frequently used platforms Java, PHP, and Python are supported.

1 INTRODUCTION

With the rise of the World Wide Web (WWW) in the past fifteen years the environment for software applications as well as for software development has changed fundamentally. Along with the WWW, new forms of feature-rich web applications have emerged becoming a ubiquitous part of business and everyday life. But the continuous usage of increasingly complex and widespread web applications induces demands for security, reliability and availability. Recent analysis has shown that often those goals are unrealized. In the last years a large portion of cybercrime attacks on computer systems could be conducted even by basic SQL injection attacks (Maple et al., 2010), taking advantage of poor coding practice in applications and leading to substantial losses of sensitive data (O'Dell, 2009).

As a result of the continuing software crisis in the web context, the discipline of web engineering has emerged from traditional software engineering. Web engineering is intended to provide web developers with a sound methodology, a disciplined and repeatable process, better development tools, and a set of good guidelines for developing web applications (Ginige and Murugesan, 2001).

In the field of web engineering the model-driven software development (MDSD) is gaining ground. Roughly, the idea of MDSD is to develop software on a raised level of abstraction by automatically transforming models of software systems in one or more steps to source code. Therefore technologies such as domain-specific modeling languages (DSL), code generators, and model-to-model transformation definitions are used (Stahl et al., 2006). The Object Management Group (OMG) envisioned with the model-driven architecture (MDA) a standardization of MDSD in which models are distinguished to be computation independent (CIM), platform independent (PIM) or platform-specific (PSM) (Miller and Mukerji, 2003). The Unified Modeling Language (UML), the meta-object facility (MOF) and transformation languages such as QVT are centerpieces for the MDA approach. As with MDSD source code is mostly created automatically, the mentioned typical security vulnerabilities can be avoided by using high quality code generators.

This paper introduces an MDSD generator framework that enables the automatic generation of typical data-oriented web applications (Figure 1). The generator framework realizes the MDA vision by supporting CIM, PIM and PSM modeling and by providing modeling languages and transformation definitions for the target web platforms Java EE, PHP, and Python. Thereby it is shown that multiple web platforms with heterogeneous characteristics can be covered by one single MDSD generator framework. Part of the framework is a family of domain-specific modeling languages consisting of the five languages WASL Data, WASL Abstract, WASL JavaEE, WASL Php and WASL Python covering the different abstraction layers and target platforms.

Wolffgang U..
MULTI-PLATFORM MODEL-DRIVEN SOFTWARE DEVELOPMENT OF WEB APPLICATIONS.
DOI: 10.5220/0003500501620171
In *Proceedings of the 6th International Conference on Software and Database Technologies* (ICSOFT-2011), pages 162-171
ISBN: 978-989-8425-77-5

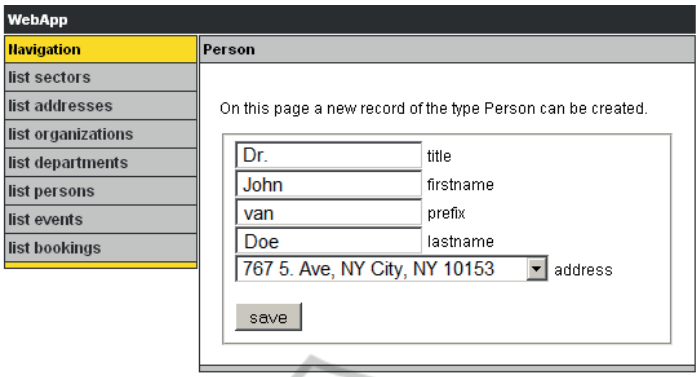

Figure 1: Representative GUI of a generated web application.

The rest of the paper is structured as follows. In Section 2 related work in the field of model-driven web engineering is discussed. Then in Section 3, metamodels of the modeling languages WASL Data, WASL Generic and partly WASL Php are introduced. Section 4 describes an exemplary transformation definition for model-to-model transformations. In Section 5, we conclude and point out future work.

2 RELATED WORK

There exist several methods and languages for web engineering, which differ w.r.t. the level of abstraction and notation. These approaches have in common, that they capture the hypertextual character of web applications. According to the separation of concerns, web applications are commonly represented and structured by data, navigation, and presentation models (Schwinger and Koch, 2006). Some of the approaches offer a modeling language, others a language based metamodel plus a process model forming a complete method. Most of them provide a code generator for generating source code from models, thus implementing the idea of model-driven web engineering.

The *Object Oriented Web Solutions (OOWS)* method is a modification of the *OO-method* (Pastor et al., 2001) and provides language elements for modeling data, navigation and presentation aspects of web applications (Fons et al., 2003). It follows the MDSD approach by generating source code from conceptual models through intermediate models and transformation definitions.

UML-based Web Engineering (UWE) is a method for the development of web applications that covers a process model and a modeling language in the form of UML profiles. UWE supports data, navigation and presentation models of web applications as well as models describing user interaction (Koch, 2001; Koch et al., 2008). Further development of the method covers modeling rich internet applications (RIA) (Koch et al., 2009) and the application of UWE for MDSD (Kraus et al., 2007; Kroiss and Koch, 2009).

The *Web Modeling Language (WebML)* is a domain-specific modeling language suited for data-oriented web applications that provides language elements for data, navigation, presentation and user modeling (Ceri et al., 2002). In WebML data-oriented functionality is represented on a high level of abstraction by so-called units. In addition to a CASE tool for creating WebML models further efforts have been made for supporting MDSD with the specification of an UML-based metamodel (Moreno et al., 2007; Moreno et al., 2006) as well as by a domain-specific MOF-compatible metamodel (Schauerhuber et al., 2007).

The *Web Application Extension (WAE)* is a platform-near modeling language based on UML, whose metamodel is specified as an UML profile (Conallen, 2002; Conallen, 1999). WAE provides language elements for modeling content and presentation models in which navigation structures are specified implicitly in the form of pages and links in the presentation model.

3 WASL LANGUAGE FAMILY

Despite conjoint commitment (Vallecillo et al., 2007), MDSD and generally web engineering is not established in the different web development communities widely, yet. From the authors' experience this is due to following factors.

Recent approaches for modeling web applications are typically using the profile mechanism of UML or a single DSL. However in contrast to classical software engineering, using UML for web engineering

leads to mapping defects. UML is not suited well for representing concepts such as configuration files, relational schemata, web service definitions, GUI engine templates, and notation format such as YAML (Ben-Kik et al., 2009). Programming languages and frameworks provided for web development are so diverse that one single modeling language just cannot cover all of them. Object-oriented programming (OOP) languages (e.g. Java) differ fundamentally from script languages such as PHP that are widely used in web development, allowing procedural programming without object-orientation at all. In contrast others are utilizing high-level frameworks (e.g. Ruby on Rails) that use configuration files extensively. This leads to modeling defects when pursuing the goal of PSMs that are intended to represent the implementation as close as possible. The abstraction from target platform concepts leads to a flat learning curve for new users of those generators as they have to understand which source code fragments are generated from which model elements. Furthermore the missing concepts hardly can be added to UML through the lightweight extension mechanism of UML by profiles, stereotypes and tagged values because UML does not offer elements whose semantic is generic enough to justify a specialization. Heavyweight UML extensions are not considered to be applicable because of missing tool support.

Besides appropriate modeling languages another requirement for an establishment of model-driven web engineering is that implementations of MDSD generators can be downloaded on the internet and integrated into current web development frameworks. However until now this is not the case with the described prominent web engineering tools and generators.

The MDSD generator presented in this paper contributes to the field of model-driven web engineering as it (1) targets multiple relevant and widely used web platforms by one generator framework, (2) provides a close representation of those platforms by multiple platform-specific DSLs, (3) supports CIM, PIM, and PSM modeling, and (4) uses standardized meta metamodels, transformation languages and code generator frameworks. Additionally (5) all platform-specific modeling languages are derived from their corresponding target platforms and are themselves used as the source for deriving higher-level platform-independent modeling languages. This guarantees that CIM and PIM models always can be transformed to source code through underlying PSM models.

3.1 WASL Framework

The Web Application Specification Language (WASL) family is a set of semi formal modeling languages for the model-driven development of data-oriented web applications. All languages are interconnected by transformation definitions, which support the transformation of platform-independent WASL models to platform-specific WASL models and from platform-specific WASL models to source code (Figure 2).

The WASL language family is an implementation of the MDA envisioned by the OMG, which classifies models as CIM, PIM and PSM (Miller and Mukerji, 2003). As a common metamodeling language the OMG proposes the meta-object facility (MOF), for which an implementation exists with the Eclipse Modeling Framework (EMF) and its meta metamodel Ecore that slightly differs from MOF in some details (Gerber and Raymond, 2003). All WASL languages are based on the Ecore meta metamodel because it provides large tool support through the Eclipse Modeling Framework (EMF), several model-to-model transformation languages, code generators and standardized file formats such as XMI. In the following we assume some familiarity with those prominent MDSD technologies, particularly with Ecore.

In WASL, the web application development process starts by creating a conceptual data model, which represents the structure of the domain as a CIM. For this step the modeling languages ERM, UML, or WASL Data can be used. As WASL Data (Sec. 3.2) is the source language for further transformations, models written in ERM or UML have to be transformed to WASL Data models first. This is achieved with a transformation definition written in the standardized transformation language QVT Operational (QVTO). A redundant implementation based on QVT Relational (QVTR) has not proved to be applicable because of performance problems.

PIM layer models are specified with the semi formal modeling language WASL Generic (Sec. 3.3), which offers language elements for modeling the data structure, navigation structure, business logic, and presentation structure of a web application. WASL Generic models can be generated from WASL Data models based on a QVTO transformation definition. In the transformation step, the data model is copied from the WASL Data model to the WASL Generic model as both languages represent data models in the same way. Additionally the contents of the navigation, business logic, and presentation models are generated from the contents of the WASL Data model automatically. The generated business logic and presen-

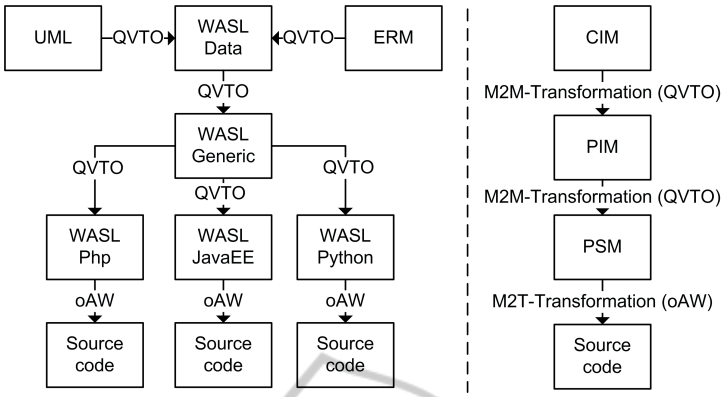

Figure 2: WASL MDSD framework.

tation aspects represent the functionality of the web application for create, read, update and delete (CRUD) operations on the entities described in the data model. The generated WASL Generic model serves as an initial point for modifications on the generated model elements and for enrichments by adding new model elements for additional functionality.

In contrast to the PIM language WASL Generic, platform-specific implementation details are modeled with the three semi-formal modeling languages WASL Php (Sec. 3.4), WASL Python and WASL JavaEE, covering the three widely established target platforms PHP, Python and Java EE. WASL Generic models can be transformed to models written in these three languages by three QVTO transformation definitions, which map the WASL Generic PIM concepts to detailed technical PSM realizations. Finally the transformation from the PSM level to the source code level is implemented with three individual code generators based on the openArchitectureWare (oAW) code generator framework and its transformation languages Xpand and Xtend.

As mentioned, because there is only a small overlap between domain concepts covered by UML and those within the target web platforms, custom DSLs are favored instead of cutting UML down into needed parts. However a transformation definition exists between UML and WASL Data providing a bridge between both worlds. The use of intermediate WASL languages for modeling platform-specific details offers multiple advantages:

1. The complexity of bridging the gap between the PIM and the final source code is shifted from the source code generator to the model transformation engine. The advantage is that QVTO offers type safety on the source model as well as target model in contrast to code generator frameworks such as openArchitectureWare, which only ensure type safety on the source model. Also preconditions can be checked as part of the model-to-model transformation such that the code generator can focus on creating source code exclusively.
2. Due to the extraction of condition checks, the code generator engine needs only a minimal feature set. This allows the generator engine to be exchanged against most other Ecore-based code generator frameworks.
3. It is considered reasonable to represent the implementation directly as a model on the lowest level of the MDSD stack. By using intermediate WASL languages these PSM aspects do not pollute the more abstract language WASL Generic, which is intended to conceal those platform-specific details.
4. PSM languages have proven to be a good foundation for developing the PIM language WASL Generic as they offer a structured reference the PIM can be built against. This allows a bottom-up research process developing WASL Generic as an abstraction from the platform-specific WASL languages which by that are a co-product on the search for an optimal PIM language for web applications.
5. The consistency of a PSM can be checked more easily than the consistency of generated source code with standard check frameworks such as OCL or oAW Check. This is due to the fact, that a consistency check on the source code level introduces the need for a adaptable parser which is not available for every target programming language.

In the following, the metamodels of WASL Data and WASL Generic are detailed. Subsequently a representative subset of the metamodel of WASL Php is described for illustrating the platform-specific character of the PSM modeling languages WASL JavaEE, WASL Php and WASL Python. As the metamodels of WASL PSM languages are substantially larger and

more complex than WASL Data and WASL Generic only a subset is covered in this paper.

3.2 WASL Data

WASL Data is a semiformal modeling language for CIM modeling in the WASL generator framework and offers language elements for representing the structural domain aspects of a web application in the form of a data model.

The metamodel of WASL Data (Figure 3) resembles the UML metamodel for class diagrams and generally offers language elements for specifying packaged entity types, typed properties and associations between these properties. The central language element *Entity* represents an entity type of the conceptual domain. A type hierarchy can be built with the EReference *generalization* for defining the super type of an entity and with the EReference *specializations* for multiple opposite subtypes. An entity can possess properties, which are subdivided into *ValueProperties* and *ReferenceProperties*. The super type *Property* allows merging collections of both *Property* types into a single collection. This is important for specifying an ordered set of properties mixed of *ReferenceProperties* and *ValueProperties*, which for instance is represented in the resulting web application as an ordered list of input fields and selection boxes contained in a HTML form.

Each *ValueProperty* is typed with a value from the EEnum *PrimitiveDataTypes*, which is a list of all provided platform-independent primitive data types. For example the type *PrimitiveDataType::UnsignedInteger* can be assigned to a *ValueProperty* instance named *age* for modeling the age of a person. Using an enumeration of fixed primitive types offers the advantage of type-specific and type-safe mappings in subsequent model-to-model-transformations. E. g. it can be specified that both the primitive PIM types *Integer* and *UnsignedInteger* should be transformed to the PSM primitive SQL type *BIGINT*.

In WASL Data in contrast to ERM entities can be packaged and properties are typed with a preset primitive type system. In contrast to the UML metamodel, the WASL Data metamodel is reduced to a minimal feature set and provides special EAttributes for some metamodel elements such as the *Entity's nameProperties*. Additionally in UML the primitive types are not preset by the metamodel but are specified on the model layer with instances of the metamodel element *PrimitiveType* (OMG, 2007). With this approach a *ValueProperty* instance would be typed by assigning one of those *PrimitiveType* instances to the *ValueProperty* instance. The UML approach has the advantage of higher flexibility as the language can be configured while modeling by modifying the primitive type system. The downside of the approach is the ambiguity of primitive types that leads to the problem that primitive types can only be identified through textual comparison by their name. Thus they cannot not be transformed type-safely in a type-to-type-mapping. Because of this issue the approach of using an enumeration of fixed primitive types is chosen for WASL Data.

Each *Entity* is contained in a *Package* which itself can be nested into a super package. This allows structuring *Entities* in a hierarchy grouped by domain concepts. Associations between *ReferenceProperties* are modeled by the metamodel element *Association*, which provides bidirectional navigable relationships between *Properties*. An association can reference the same *Entity* or two different *Entities*, such that the question arises where to store the association instance in the model. Similarly to UML an *Association* instance can be contained in any *Package* instance without any restriction regarding the packages, which are containing the *Entity* instances referenced by the *Association*.

3.3 WASL Generic

WASL Generic is a platform-independent semiformal modeling language, which offers language elements for modeling data structures, navigational structures, presentation structures, and business logic inducing a separation of concerns following the Model-View-Controller (MVC) pattern (Reenskaug, 1979). The language elements for the data model comply with WASL Data such that WASL Generic can be interpreted as an extension of WASL Data.

The language elements for the navigational model are contained in the EPackage *navigation* (Figure 3) and represent the web application's navigation structure in the form of a directed graph. Web applications consist of pages or respectively views between which the user interactively can navigate with the browser e.g. by clicking links and buttons. In WASL Generic, the term *Node* is defined as a part of a web application covering delimited specific functionality, addressed by a unique URL and navigable by the user with the browser's controls such as back and forth buttons as well as bookmarks. Normally a *Node* corresponds to a single web page; however other implementations with multiple pages are possible as long as only one URL is presented to the user for these pages. Also platform-specific technologies such as AJAX allow the content of a web page to be replaced after

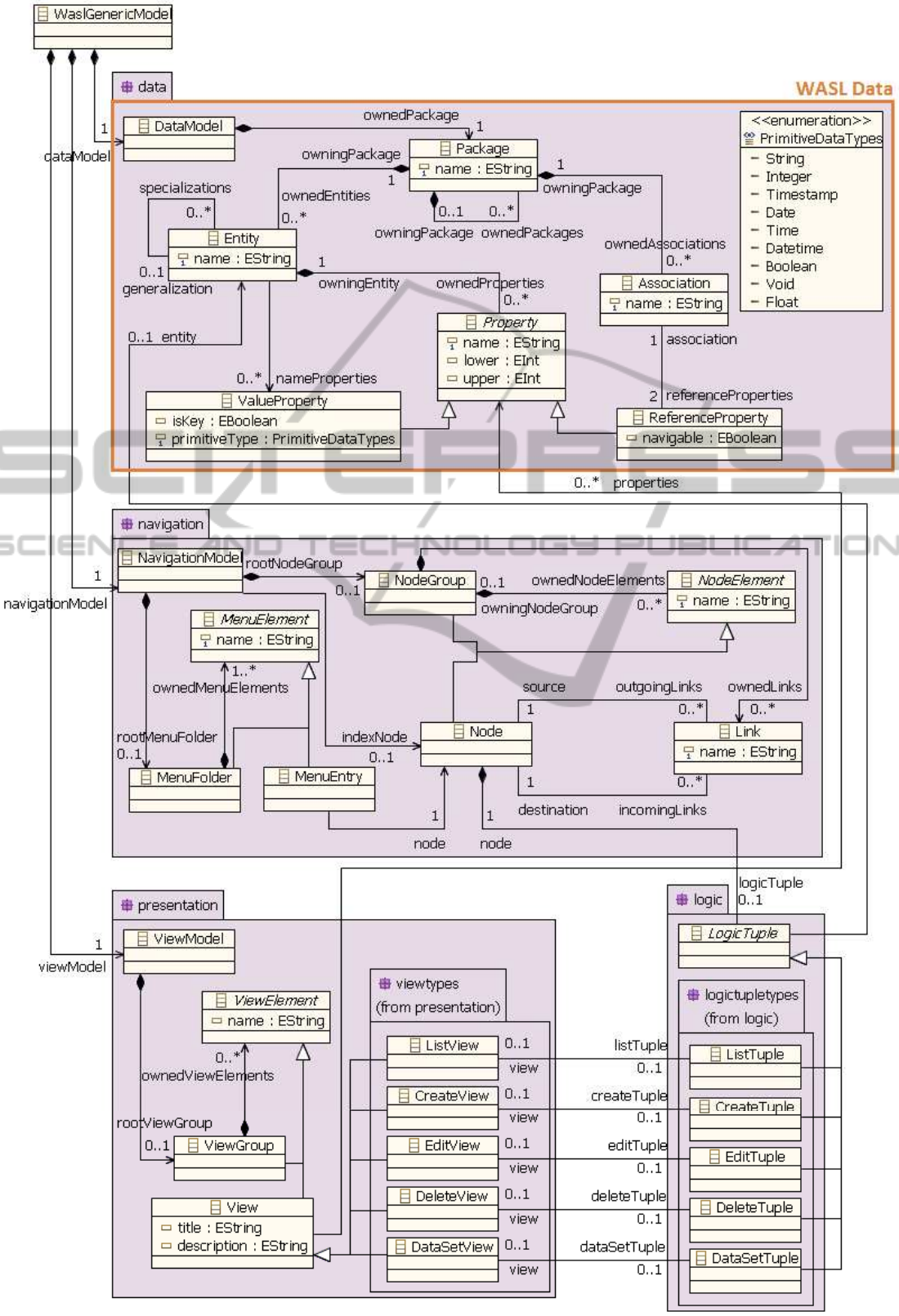

Figure 3: Metamodel of WASL Data and WASL Generic.

initially rendering it within the browser, leading to a multi-page approach based on a single page and URL.

The navigation path between two *Nodes* is expressed by a *Link*, which corresponds to a directed edge of the page graph and is implemented platform-specifically e.g. by a hypertext anchor or a HTML form's action attribute. Each *Node* is contained in a *NodeGroup*, which itself can be nested in a super *NodeGroup* thus forming a hierarchy of *NodeGroups*. Usually the structure of the hierarchy is structured along the functional areas of the modeled web application.

The menu structure of the web application is modeled without using *Links* with the purpose to reduce the number of links in the navigation model and thus to reduce the complexity. Modeling the menu structure in the navigation model by links with a number of n nodes would lead to a number of n^2 links as for each node a link to all nodes would have to be added. This does not comply with the requirement of clarity a model should satisfy. Also typically the menu of a web application is visualized and implemented separately from the rest of the page contents (Figure 1), such that a separated representation in the model simplifies the implementation of transformation definitions. Corresponding to *NodeGroups*, the menu can be structured hierarchically by *MenuFolders*.

In WASL Generic, standard business logic patterns are represented separately from the *Node* concept as specializations of the metamodel element *LogicTuple*. A *LogicTuple* is assigned to a *Node* and aggregates the information, which *Node* operates on which *Entity* and should be visualized by which *View*. The approach is flexible as additional business logic types can be appended to the metamodel of WASL Generic as additional specializations of the element *LogicTuple*.

The business logic is represented to the user by one or more *Views* allowing to model simple CRUD operations as well as for example more complex multi-page wizards. In contrast to *Nodes*, multiple *Views* owned by one *LogicTuple* must not be distinguishable for the user by individual URLs and as described thus do not have to be navigable by the browser's back and forward buttons or bookmarks. As navigation and presentation structure are separated from each other and AJAX functionality is represented only in the views of the presentation model, the page-driven paradigm typical for web applications is ensured. AJAX introduces the problem that the functionality of the browser's page-driven URL addressing mechanism is broken due to the fact, that with AJAX the page content is just replaced and no transition between two web pages distinguishable by two URLs is made (Zucker, 2007). The separation of navigation and (AJAX-based) presentation in WASL Generic assures that web applications derived from WASL Generic do not suffer typical AJAX symptoms such as a missing browser history, broken back and forward buttons and indistinguishable URLs prohibiting the bookmarking of specific parts of the web application.

Current web engineering research addresses issues of modeling process-oriented web applications (Brambilla et al., 2006). An advantage of WASL Generic in separating *Nodes* from *LogicTuples* originates from the fact, that complex process-oriented logic can be modeled by *LogicTuples* separately from the node-based navigation model. Thereby process-oriented web applications can be modeled based on a chain of *LogicTuples*, which for example can rely on web service calls and interacting with each other. The user can interact with the process by respective *LogicTuples*, addressed by *Nodes* and presented by *Views*.

3.4 WASL Php

The metamodel of WASL Php for the scripting language PHP is organized according to the specific language features of PHP and provides language elements for modeling entity classes and data access objects (DAO) for a object-relational mapping (ORM) framework such as Doctrine. For describing the platform-specific character of WASL Php a subset of the metamodel of WASL Php is detailed in this section. The full metamodel as well as all other WASL metamodels can be found in (Wolffgang, 2009).

The main container of a WASL Php model is the EClass *PhpModel* that contains the class model, a page model and a relational schema (Figure 4). With the Doctrine ORM framework a relational schema is represented by a set of PHP classes, which map the schema metadata for the database tables managed by Doctrine. These classes and database tables can be generated by Doctrine automatically from an YAML schema file. Metamodel elements for describing such a schema are provided with the EPackage *doctrine* and its EClass *Schema*. An YAML Schema contains *Models* that represent database tables and *Connections* that can be associated to *Models* to specify, how Doctrine should access the database. With the option *detect_relations* Doctrine can be instructed to guess relationships between Models based on Models' column names.

As described a *Model* is implemented by a PHP entity class as well as a database table and can contain multiple *Columns* for representing value properties, multiple *Relations* for referencing other *Mod-*

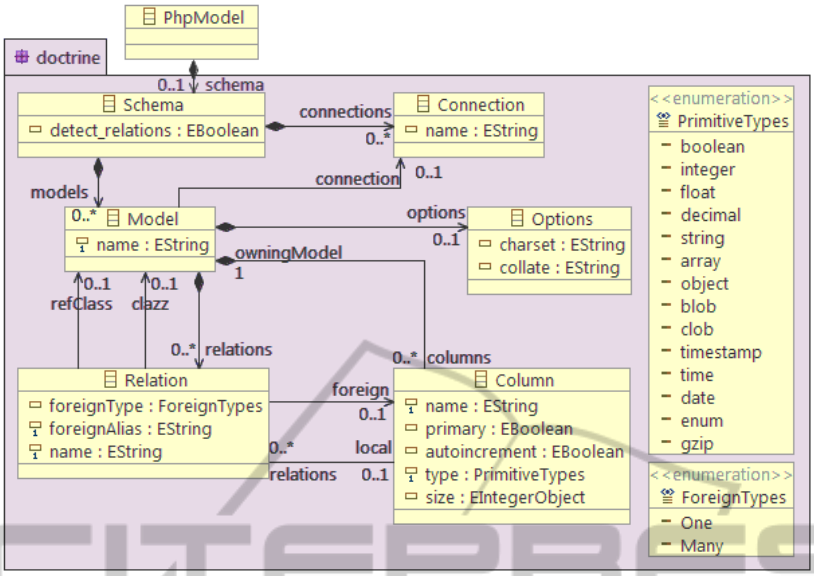

Figure 4: Subset of the metamodel of WASL Php.

els, a database *Connection* and multiple *Options* for charset and collate settings. A value attribute is expressed as a single *Column* correspondingly to the element *ValueProperty* of WASL Generic. In contrast, corresponding to a *ReferenceProperty* an association between a Model A and a Model B is expressed through a *Column* owned by A plus a *Relation* owned by B referencing the *Column* in A. Each *Column* can be marked as a primary key, whose value can be generated automatically when inserting a new row. The column's type has to be selected from the EEnum *PrimitiveTypes*. Corresponding to the Doctrine documentation each relation references its local column as well as a key column of the referenced model. With the EAttribute *foreigntype* the cardinality of the relation is specified, allowing the combinations *One:One*, *Many:One*, *One:Many* and *Many:Many*. In the case of a relation with a cardinality of *Many:Many* a cross table *Model* has to be added, which is referenced by the two associated Models through the EReference *refClass*.

For each *Model* the corresponding *DoctrineClass* can be specified additionally. WASL Php offers an explicit approach of modeling and generating both the low-level database YAML schema file as well as the high-level Doctrine model classes for reasons of (1) complete explication, (2) integration into the build process and (3) controlling how the generated source code should look. Another approach not chosen in WASL would be to model just the YAML schema file, to generate it with the MDA generator workflow and separately to generate the Doctrine PHP classes based on the YAML file with the generator script provided by Doctrine. Such an approach has the problem, that (1) in the model other PHP classes cannot reference methods of the model classes as they do not exist in the model, (2) the build process is split into two generator steps and (3) the external generator process cannot be customized.

4 TRANSFORMATION

As described the transformation definitions of the WASL generator framework are implemented in QVTO and Xpand (Figure 2). The following example is intended to illustrate how the plattform-independent language element *Entity* from WASL Generic can be transformed with QVTO to the low-level WASL Php language element *Model* representing a Doctrine ORM model element. The generated elements conform to the typical doctrine YAML file format and represent the low-level concepts of doctrine in the MDSD modeling environment.

```
mapping GENERIC::data::Entity::toModel
    (in model : GENERIC::WaslGenericModel)
    : PHP::doctrine::Model{
  name := self.name.firstToUpper();
  columns := self.getValueProperties()
    ->select(upper = 1).map toColumn()
    ->asSequence()
    ->union(self.getReferenceProperties()
    ->select(navigable)->select(upper = 1).map
    toColumn())->asOrderedSet();
  relations := self.getReferenceProperties()
    ->select(p | p.upper = 1).map
    toRelation(model)->asOrderedSet();
```

```
  options := object PHP::doctrine::Options{
    charset := 'utf8';
    collate := 'utf8_general_ci';
  }
}
```

When running the generator, for each element with the type *Entity* in a WASL Generic source model the mapping creates an element with the type *Model* in the WASL Php target model. Additionally the attributes *name*, *columns*, *relations* and *options* are filled with values. In the transformation the *Entity*'s name is copied to the *Model*'s name with the first character as upper-case, following the conventions of Doctrine. The ERefernce *columns* is filled with *Column* elements that are generated for all of the *Entity's ValueProperties* and navigable *ReferenceProperties* with a cardinality of 1. Additionally for each *ReferenceProperty* with a cardinality of 1 a relation is created that similarly to a foreign key points to the referenced *Model*. Additional platform-specific details such as the default encoding of the database table and selected charset are set by the transformation definition by default to UTF8 such that the developer does not have to add those settings to the target model after each transformation.

5 CONCLUSIONS

We have presented the WASL MDSD generator that allows fully automatic generation of web applications based on typical web platforms such as Java EE, PHP, and Python. The proposed approach contributes to the field of model-driven web engineering as it (1) realizes the MDA vision by platform-specific and platform-independent modeling, (2) supports multiple web platforms, (3) utilizes established MDA standards, and (4) represents platform concepts directly through platform-specific DSLs.

Future work includes efforts for extending the metamodel of WASL Generic with additional language elements for specifying additional presentation and logic concepts, e.g. for tethering the web application to service-oriented architectures by specific *LogicTuples*. Also a metamodel for the Google Web Toolkit (GWT) promises interesting possibilities in RIA modeling. Furthermore it is intended to decompose the platform-specific WASL languages to submetamodels with each of them representing exactly one web development framework such as Doctrine. This would allow composing project-specific modeling languages suited to the set of frameworks used in specific software development projects.

REFERENCES

Ben-Kik, O., Evans, C., and dt Net, I. (2009). *YAML Ain't Markup Language (YAML) Version 1.2*.

Brambilla, M., Ceri, S., Fraternali, P., and Manolescu, I. (2006). Process modeling in web applications. *ACM Transactions on Software Engineering and Methodology*, 15(4):360–409.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., and Matera, M. (2002). *Designing Data-Intensive Web Applications*. Morgan Kaufmann, San Francisco.

Conallen, J. (1999). Modeling web application architectures with uml. *Communications of the ACM*, 42(10):63–70.

Conallen, J. (2002). *Building Web Applications With Uml*. Addison-Wesley, Reading, USA, 2 edition.

Fons, J., Pelechano, V., Albert, M., and Pastor, O. (2003). Development of web applications from web enhanced conceptual schemas. In *Workshop on Conceptual Modeling and the Web, ER'03*, volume 2813 of *LNCS*, pages 232–245, Chicago, USA. Springer.

Gerber, A. and Raymond, K. (2003). Mof to emf: There and back again. In Burke, M. G., editor, *OOPSLA Workshop on Eclipse Technology eXchange (OOPSLA2003)*, pages 60–64, Anaheim, California. ACM-Press.

Ginige, A. and Murugesan, S. (2001). Web engineering - an introduction. *IEEE MultiMedia*, 8(1):14–18.

Koch, N. (2001). *Software Engineering for Adaptive Hypermedia Systems*. PhD thesis, LMU Mnchen.

Koch, N., Knapp, A., Zhang, G., and Baumeister, H. (2008). *Web Engineering: Modelling and Implementing Web Applications*, volume 12, chapter 7, pages 157–191. Springer, Heidelberg.

Koch, N., Pigerl, M., Zhang, G., and Morozova, T. (2009). Patterns for the model-based development of rias. In *Proc. 9th Int. Conf. Web Engineering (ICWE'09)*, volume 5648, pages 283–291, San Sebastian, Spain. Springer.

Kraus, A., Knapp, A., and Koch, N. (2007). Model-driven generation of web applications in uwe. In *Proc. MDWE 2007 - 3rd International Workshop on Model-Driven Web Engineering, CEUR-WS*, volume 261.

Kroiss, C. and Koch, N. (2009). Uwe4jsf: A model-driven generation approach for web applications. In *Proc. 9th Int. Conf. Web Engineering (ICWE'09)*, volume 5648 of *LNCS*, pages 493–496, San Sebastian, Spain. Springer.

Maple, C., Phillips, A., and Morris, B. (2010). Uk security breach investigations report - an analysis of data compromise cases 2010. Technical report, 7Safe.

Miller, J. and Mukerji, J. (2003). Mda guide. Technical report, Object Management Group.

Moreno, N., Fraternali, P., and Vallecillo, A. (2006). A uml 2.0 profile for webml modeling. In *Workshop on Model-Driven Web Engineering (MDWE2006)*, Palo Alto, USA.

Moreno, N., Fraternali, P., and Vallecillo, A. (2007). Webml modeling in uml. *IET Software*, 1(3):67 – 80.

O'Dell, J. (2009). Rockyou hacker - 30% of sites store plain text passwords. *The New York Times*.

OMG (2007). Uml 2.1.2 infrastructure. Technical report, Object Management Group.

Pastor, O., Gmez, J., Insfrn, E., and Pelechano, V. (2001). The oo-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. *Information Systems*, 26(7):507–534.

Reenskaug, T. (1979). Models - views - controllers. Technical report, Xerox Parc.

Schauerhuber, A., Wimmer, M., Kapsammer, E., Schwinger, W., and Retschitzegger, W. (2007). Bridging webml to model-driven engineering: From dtds to mof. *IET Software*, 1(3):81–97.

Schwinger, W. and Koch, N. (2006). *Web Engineering: The Discipline of Systematic Development of Web Applications*, pages 39–64. John Wiley and Sons.

Stahl, T., Voelter, M., and Czarnecki, K. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley and Sons, Chichester, England.

Vallecillo, A., Koch, N., Cachero, C., Comai, S., Fraternali, P., Garrigs, I., Gmez, J., Kappel, G., Knapp, A., Matera, M., Meli, S., Moreno, N., Prll, B., Reiter, T., Retschitzegger, W., Rivera, J. E., Schauerhuber, A., Schwinger, W., Wimmer, M., and Zhang, G. (2007). Mdwenet: A practical approach to achieving interoperability of model-driven web engineering methods. In Koch, N., Vallecillo, A., and Houben, G.-J., editors, *Proc. 7th Int. Conf. Web Engineering (ICWE'07)*, volume 261 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Wolffgang, U. (2009). Web application specification language (wasl). Technical report, ERCIS.

Zucker, D. F. (2007). What does ajax mean for you? *interactions*, 14(5):10–12.