

# Agile Development of PHP Websites: A Model-Aware Approach

Carmen Ioana Gog\*

Faculty of Economics and Business Administration, Babeş-Bolyai University,  
Teodor Mihali Street Nr. 58-60, Cluj-Napoca, 400591, Romania

[carmen.gog@yahoo.com](mailto:carmen.gog@yahoo.com)

**Abstract.** The Design Science research method was hereby employed to develop an artifact that demonstrates the experimental “model-aware” software engineering methodology in the context of PHP Web development – a “low code” development approach with code templates generated from technology-specific models. The proof-of-concept consists of two interacting components: a custom diagrammatic modeling environment and model-driven generated PHP pages. The interaction between the two components conforms the engineering method labelled as “Model-aware software engineering” (MASE) – a flavor of model-driven engineering recently introduced in research projects as a hybridization of the Agile Modeling Method Engineering (AMME) framework and the Resource Description Framework (RDF). The experimental MASE method is employed here to demonstrate its feasibility for the common Model-View-Controller (MVC) website development pattern, thus showing potential to support common Web development work.

**Keywords:** Agile Modeling Method Engineering, Model-Aware Software Engineering, Resource Description Framework, Model-Driven Webpages.

## 1 Introduction

Recent research in model-driven engineering introduced two novel notions: (a) “technology-specific” modeling languages, a kind of domain-specific modeling languages where, instead of domain concepts, the language must assimilate technology-specific constructs as first order modeling citizens [1] and (b) “model-aware software engineering” (MASE), a flavor of model-driven engineering where models interact with running software through semantic technology [2].

The work at hand combines the two notions through a Design Science approach in order to demonstrate the feasibility of MASE for engineering tasks as common as Model-View-Controller (MVC) based Web development in PHP. The artifact reported by this article comprises two components: (a) a technology-specific modeling method for PHP Web

---

\* Corresponding author

© 2020 Carmen Ioana Gog. This is an open access article licensed under the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0>).

Reference: C. I. Gog, “Agile Development of PHP Websites: A Model-Aware Approach,” *Complex Systems Informatics and Modeling Quarterly*, CSIMQ, no. 25, pp. 19–31, 2020. Available: <https://doi.org/10.7250/csimq.2020-25.02>

Additional information. Author’s ORCID iD: C. I. Gog – <https://orcid.org/0000-0001-7358-069X>. PII S225599222000144X. Received: 15 November 2020. Accepted: 9 December 2020. Available online: 31 December 2020.

development and (b) demonstrative pages created by applying MASE with input from diagrams created with that method.

Design Science [3] was engaged as a research method and MASE [2] as an engineering method, combining Agile Modeling Method Engineering (AMME) [4] (used for the implementation of the diagrammatic modeling tool) with Resource Description Framework (RDF) [5] as semantic bridge between the PHP development environment and the technology-specific models.

The remainder of the article is structured as follows: Section 2 provides an overview of the used methods and languages for the entire artifact development process, together with the requirements for both the custom-made diagrammatic modeling tool and the model-driven webpages. Section 3 comments on the related literature. Section 4 briefly presents the used research method together with the development method. Section 5 draws up the design decisions made for both components. Section 6 details some of the implementation details. The article ends with a concluding evaluation and outlook.

## 2 Problem Statement and Solution Architecture

The targeted beneficiaries are Web developers aiming for a low-code development approach for common webpage patterns such as form management and CRUD operations. Templating systems are already well supported by MVC frameworks such as CakePHP [6] – however, this is still done in a codebase manner, whereas the artifact reported by this article does it with a diagrammatic method that incorporates semantic constructs of the CakePHP framework (actually, of most MVC frameworks) meaning that the proposal can act as a low-code development approach.

Website developers are thus supported with a diagrammatic panel to define the usage flow of a website, the order of screens and operations in the user experience.

Conceptual modeling is the key to designing information systems, as it provides a graphical representation of human comprehension of systems or processes, possibly specialized according to a domain's semantics (in domain-specific languages) – a notion that in the case of this work translates to technology-specific semantics pertaining to PHP Web development.

This specificity is reflected in modeling requirements and must be assimilated with metamodeling means in the modeling language – i.e., modeling is seen as knowledge representation. Therefore, AMME was employed to provide the needed language customizations.

The work at hand applies the MASE method to transferred technology-specific diagrams to the website skeleton that thus is informed at build-time about the order of CRUD and front-end operations that should be exposed to an end-user. This is mediated by a Graph DB management system [7] acting as the model repository for the MASE approach. The diagrammatic modeling tool was implemented using ADOxx [8]. This architecture is depicted in Figure 1.

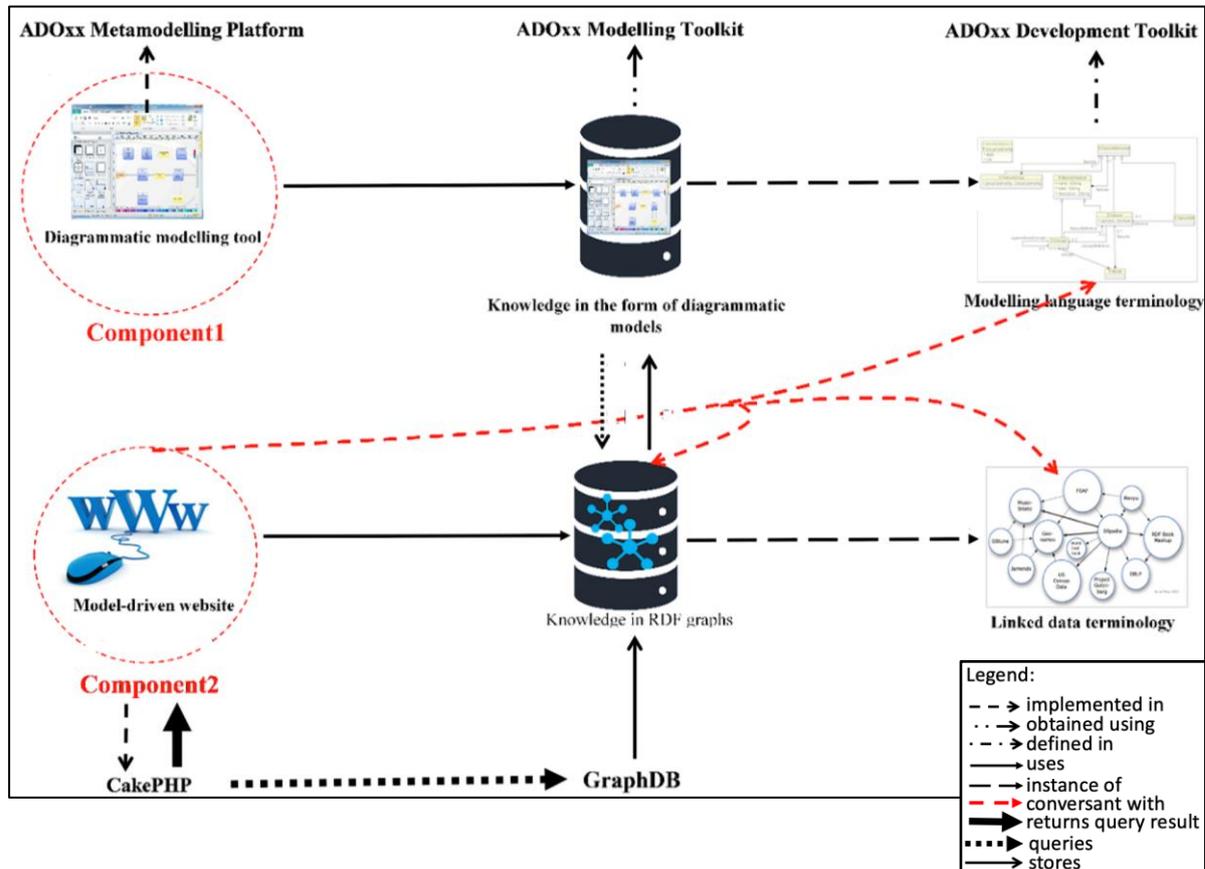
## 3 Literature Review

Diagrammatic models are a form of knowledge representation [9] presenting human-readable and linkable diagrammatic content that can be easily navigated by users, but at the same time they contain machine-readable knowledge [10], making possible the implementation of knowledge-driven functionality.

When creating diagrammatic models, the modeler must consider certain principles to effectively model, design and implement process-aware systems, gaining understanding regarding major concepts, languages and techniques, according to [11], and certain visualization principles must be considered, according to [9] and [10].

By enacting agile modeling principles combined with business modeling practices the intersection of practices in software development and model-driven engineering naturally lead to Agile Modeling [11]. Agile principles, as encouraged by [12], are necessary to be coupled with

Modeling Method Engineering, considering method development backlogs based on the building blocks defined in [4]. OMiLAB (Open Models Laboratory) was introduced there as a facilitator to domain-specific modeling methods development (acting as deployment environment and architecture) – with more details given in [13]. OMiLAB resources such as AMME and ADOxx and the ADOxx-to-RDF model export plug-in are employed in this work to crystalize the notion of “technology-specific modeling”.



**Figure 1.** Architecture and interactions between the artifact components

Technology-specific methods are a flavor of situation-specific methods as understood by [14], or as domain-specific methods as understood by the contributors of [15]. For the implementation of situation-specific or domain-specific methods several environments are available with their particular methodologies: [16] presents MetaEdit++; Eclipse-based Sirius [17] has recently become available; ADOxx was hereby employed due to its proven success in applying MASE.

A domain-specific method with RDF exporting capabilities was presented in [18] and is available as the Bee-Up tool at [19], also implemented on ADOxx. Bee-Up supports model-driven engineering through means such as SQL generation, HTTP requests and RDF; but it aims to comply with popular and standard modeling languages (UML, BPMN, ER) rather than taking the technology-specific modeling approach advertised in this article.

When taking into consideration standards, Interaction Flow Modeling Language (IFML) was “designed to express the content, user interaction and control behavior of the front-end software applications” [20]. The control of the behavior of front-end pages is presented by the authors of [21] through the generation of user interfaces from BPMN models and Class diagrams by identification and extraction of different rules and using stereotypes to extend BPMN notation. However, the work at hand proposes to generate user interfaces from an entirely customized modeling language.

Another method that established an RDF-based bridge between modelers and app developers is ComVantage [22], however, that was not intended to support common Web development tasks, mostly focusing on mobile app orchestration.

The literature also reports other approaches for managing model contents with semantic technology [23] – however the focus of that work is the ontological integration of models rather than generation of running webpage code.

## 4 Research and Engineering Method

Because it is a solution-oriented research method, Design Science has an engineering cycle driven by examining the existing requirements [3]. The cycle comprises five phases: Real-world problem investigation, Treatment design (interaction between the artifact and context), Design validation, Treatment implementation, and Evaluation. However, the identified requirements for a technology-specific language brings additional variability, which leads to the need for an agile methodology.

Because modeling requirements demand flexibility both for model contents and modeling method, the AMME framework was applied to develop the diagrammatic modeling tool. Agility from AMME’s point of view described in [4] materializes into *artifact agility* (facilitated by the division of a modeling method into building blocks) and *methodological agility* which “manifests in the engineering process itself, taking the form of an incremental and iterative spiraling development.”

The proposed artifact (i.e., modeling method) consists of a modeling language having notation (graphical symbols), syntax (rules for combining graphical symbols in each diagram type) and semantics (meaning of the modeling elements). Other components are the mechanisms, covering the functionality, and the modeling procedure which comprises the modeling actions. In addition, model-driven webpages were implemented for evaluation purposes.

In terms of the employed instruments the following were ingredients of the engineering cycle:

- ADOxx [8] the metamodeling platform which was used in the development of the modeling toolkit;
- RDF [5] used for representing model contents as graphs, with ADOxx’s RDFTransformer to convert the metamodel and models in a way that can be managed by GraphDB;
- SPARQL [24] as semantic query language for models;
- GraphDB [7] for model repository management;
- CakePHP [6] the MVC framework for website development, and MySQL [25] for the website data;
- EasyRDF [26] the programming library to manage RDF graphs in PHP.

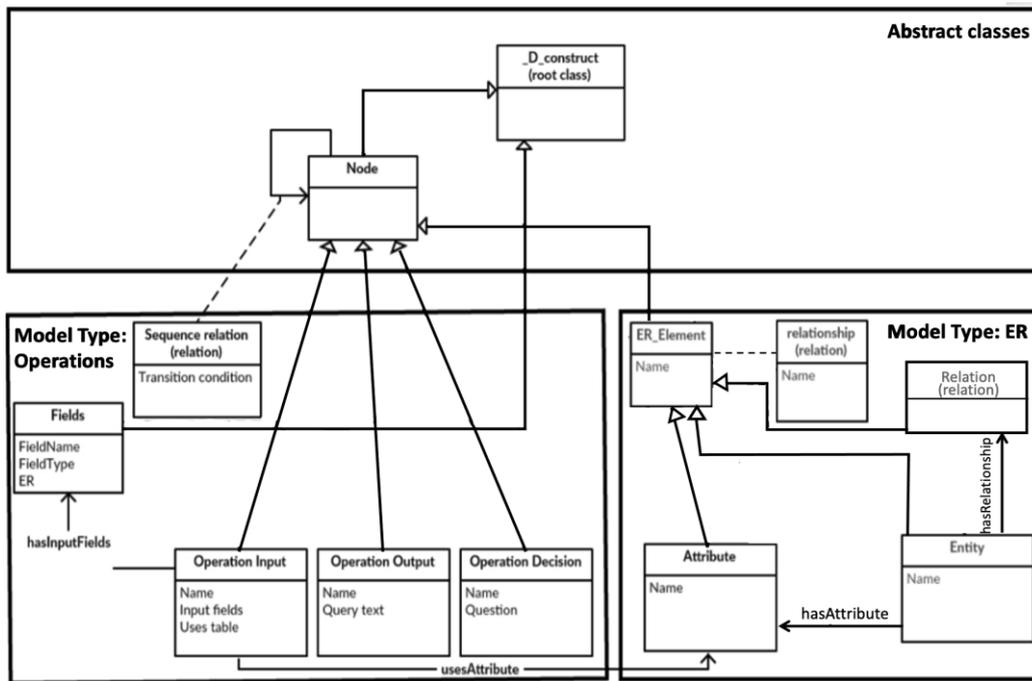
## 5 Design Decisions

In designing the metamodel of the proposed modeling software the slicing technique presented by the authors of [27] was employed.

Figure 2 presents the metamodel governing the proposed modeling software. The Operations Model type allows to model three types of operations (input, output and decisions), and the ER Model type is a simplified flavor of ER to describe data entities subjected to CRUD operations.

Semantic links are enabled between the two diagram types. The **abstract classes** comprise the ADOxx-specific `_D_` construct and Node concepts. **Model Type: Operations** forms the slice for the UI microflow and comprises one relation class which is Sequence relation and four node classes: Fields, Operation Input, Operation Output, Operation Decision. **Model Type: ER** is a slice describing the data to be manipulated by front-end forms with a simplified approach to ER modeling, adopted for demonstration purposes (the ER part will be oversimplified, since it is

well-known and not the focus of this work only its relevant elements will be brought into discussion).



**Figure 2.** The metamodel

The concepts which form the Operations Model are: OperationInput (O.I.), OperationOutput (O.O.) and OperationDecision (O.D.). The relation between them is called “Sequence relation” and it shows the order of operations as defined in a diagrammatic way. Table 1 presents them as they are defined graphically on the ADOxx platform.

**Table 1.** Metamodel concepts and their graphical symbols

Model Type	Concepts and their graphical symbols	Relations between concepts and their graphical symbols
Operations	OperationInput (O.I.) 	Sequence relation 
	OperationOutput (O.O.) 	
	OperationDecision (O.D.) 	
ER	Attribute 	hasAttribute / hasRelationship 
	Entity 	
	Relation 	

With respect to machine-readable semantics, for O.I. an attribute is “Input fields” (of type RECORD) which has the form of a table containing the fields “FieldName” (of type STRING), “FieldType” (of type ENUMERATION), and “ER” (of type INTERREF, a kind of hyperlink). Modelers are thus allowed to define form field names, their type and the corresponding ER attribute to be referenced by the current field. The field types are the ones predefined by HTML.

The attribute “uses table” permits the modeler to select the corresponding attribute from an ER diagram to be used by the current operation making possible the navigation across diagrams.

For O.O. an attribute named “Query Text” was created to allow the modeler to type in a query in SQL format in order to perform an operation on the MySQL database. The MySQL database must be created and populated in advance.

For O.D. an attribute is “Question”, which allows modelers to define the question answered by the form. Sequence relation semantics provide a “Transition condition” attribute to describe the condition which determines the advancement from current operation to the next operation.

PHP pages are built using the MVC framework CakePHP. The model-driven webpages perform operations on a MySQL database according to the diagrammatic design.

In Figure 3 the MVC architecture is shown, as applied on the model-aware website together with the existing connections. The architecture connects to both MySQL and GraphDB databases. While on MySQL database the SQL queries retrieve data to support the webpage functionality, on GraphDB the SPARQL queries act as “model queries” to fill the PHP/HTML code templates.

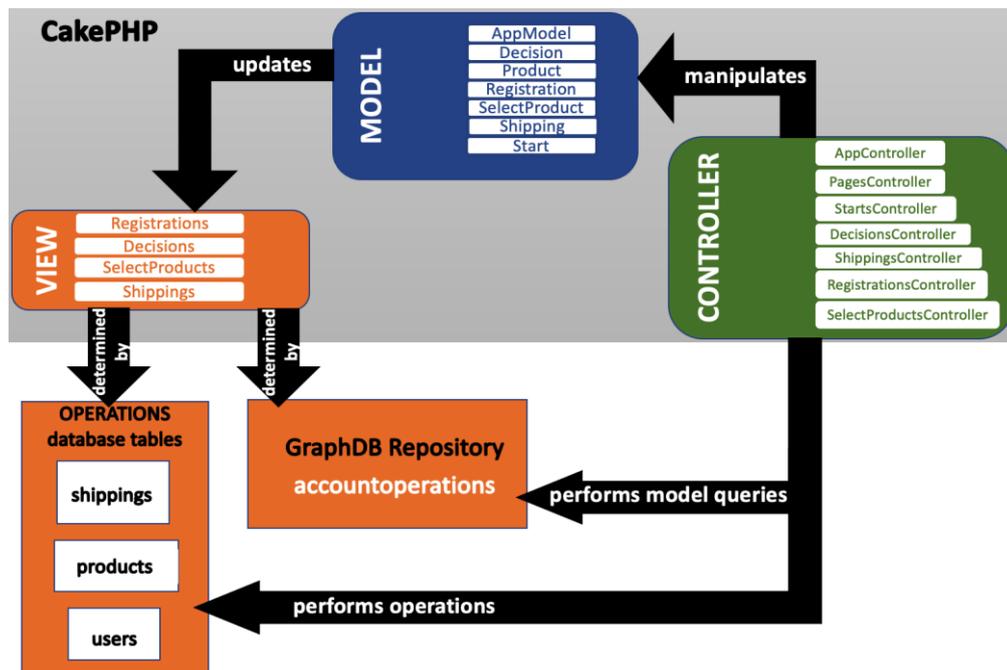


Figure 3. CakePHP architecture connected to MySQL database and GraphDB

## 6 Implementation Details

### 6.1 Implementation of the Technology-Specific Modeling Tool

An example of O.I. named “Registration” can be seen in Figure 4 together with all its possible annotation options that lead to the generation of an HTML input form.

An example of O.O. named “Products” can be seen in Figure 5. This will trigger the execution of a MySQL query on the database attached to the model. Figure 6 shows an exemplification of the operation type Decision which it will manifest as a webpage asking the user whether he/she wants to take a particular path in his front-end interaction.

Sequence Relations connect these Operation types, with a “Transition condition” that will provide options to be followed by the website user through the usage flow.

An exemplification of described operation types can be seen in Figure 7 where operations of type input are “Registration”, “Select product”, and “Shipping”; an operation of type output is “Products” and an operation of type decision is the displayed question. Transition conditions that can redirect the front-end usage flow are “A. buy product” and “B. store personal data for future operations”.

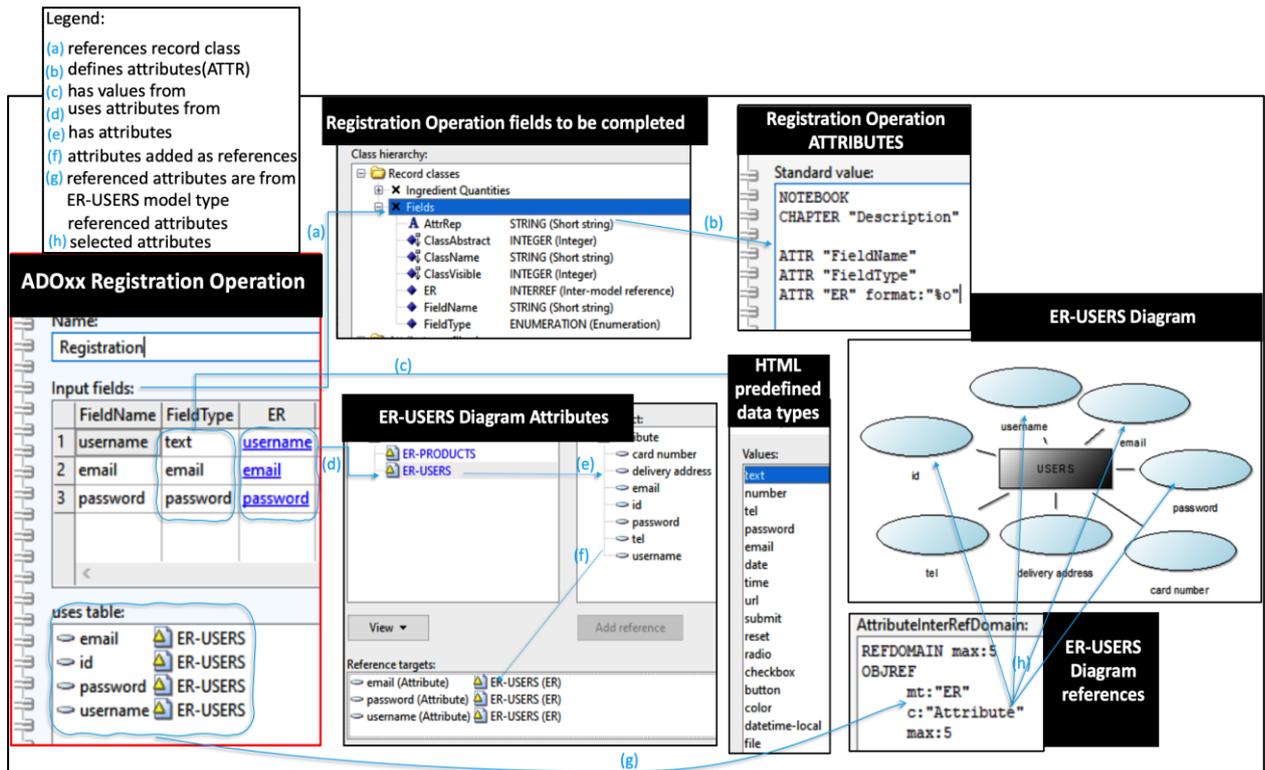


Figure 4. Exemplification of an Input Operation and its mapping on ER elements

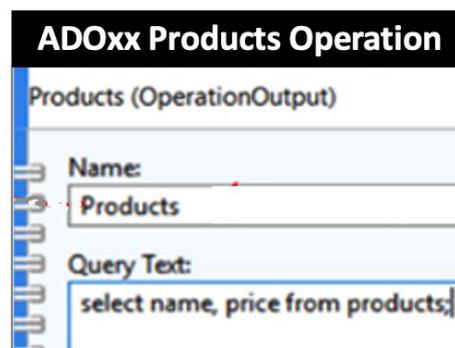


Figure 5. Attaching queries to diagram elements that represent database Operations

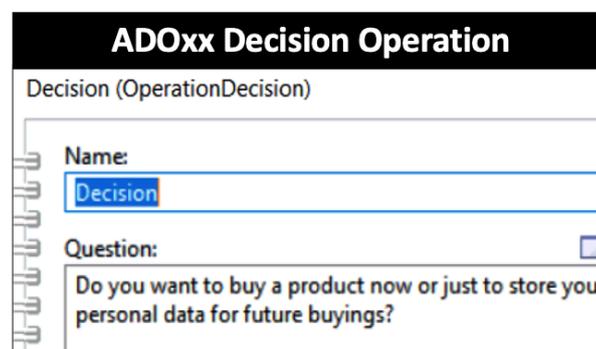
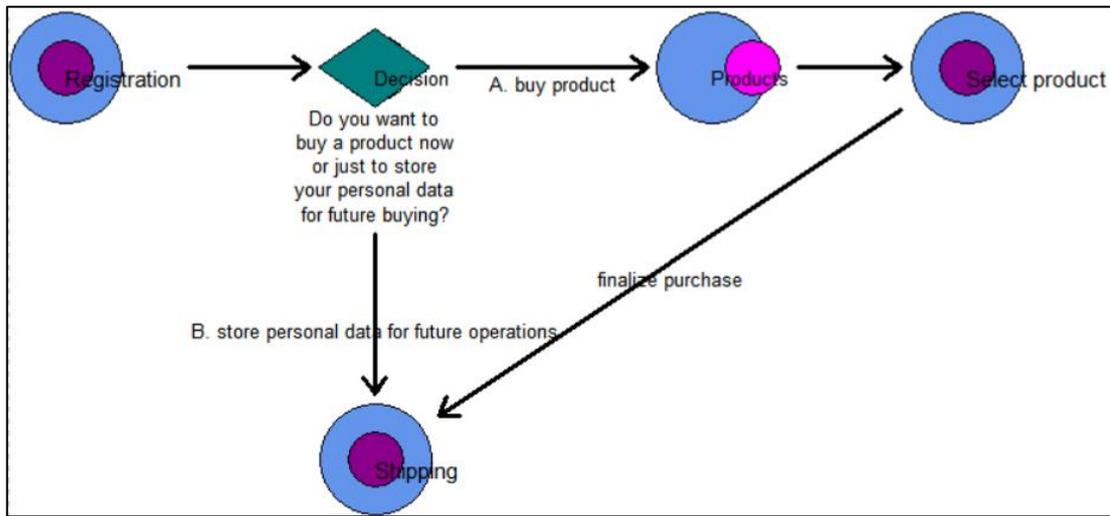


Figure 6. Attaching question text to diagram elements identified as Decision



**Figure 7.** Diagram sample created within the proposed diagrammatic modeling tool

Models are translated to RDF graphs with ADOxx’s RDFTransformer plug-in. After the RDF export is completed, GraphDB will allow the extraction (using queries) of information from the models. Some query examples can be seen in Figure 8, detecting the operations connected by an arrow (left) and the fieldnames attached to an operation (right).

It detects the direction of the relations and prints the source operation and the target operation.

```
PREFIX : <http://carmen.org#>
PREFIX cv: <http://www.comvantage.eu/mm#>
select ?fromname ?toname
where {
  ?relation a :Sequence_relation.
  ?relation cv:from_instance ?from.
  ?from :Name ?fromname.
  ?relation cv:to_instance ?to.
  ?to :Name ?toname
}
```

	fromname	toname
1	Select product	Shipping
2	Decision	Shipping
3	Registration	Decision
4	Products	Select product
5	Decision	Products

It selects the fieldnames contained by Registration operation

```
PREFIX : <http://carmen.org#>
PREFIX cv: <http://www.comvantage.eu/mm#>
select *
where {
  :OperationInput-63007-Registration-Input_fields_1
  :FieldName ?fieldname;:FieldType ?fieldtype
}
```

	fieldname	fieldtype
1	username	text

**Figure 8.** Semantic query examples for retrieving model elements

## 6.2 Implementation of Model-Aware Webpages

Model-aware webpages were implemented using CakePHP (an MVC) framework. Their content has been generated automatically based on the created models. Each page of the website represents an operation specified in the modeling tool. The order in which webpages are displayed is dictated by the diagrammatic order in the page flow model (the Operations model type).

The Registration webpage (visible in Figure 9 and Figure 10) is generated based on the created “Registration” operation within the Operations model. After the user completes the form, data is stored in “operations” (a MySQL database). Another query checks which is the current operation generating the “NEXT” button to the next operation page.

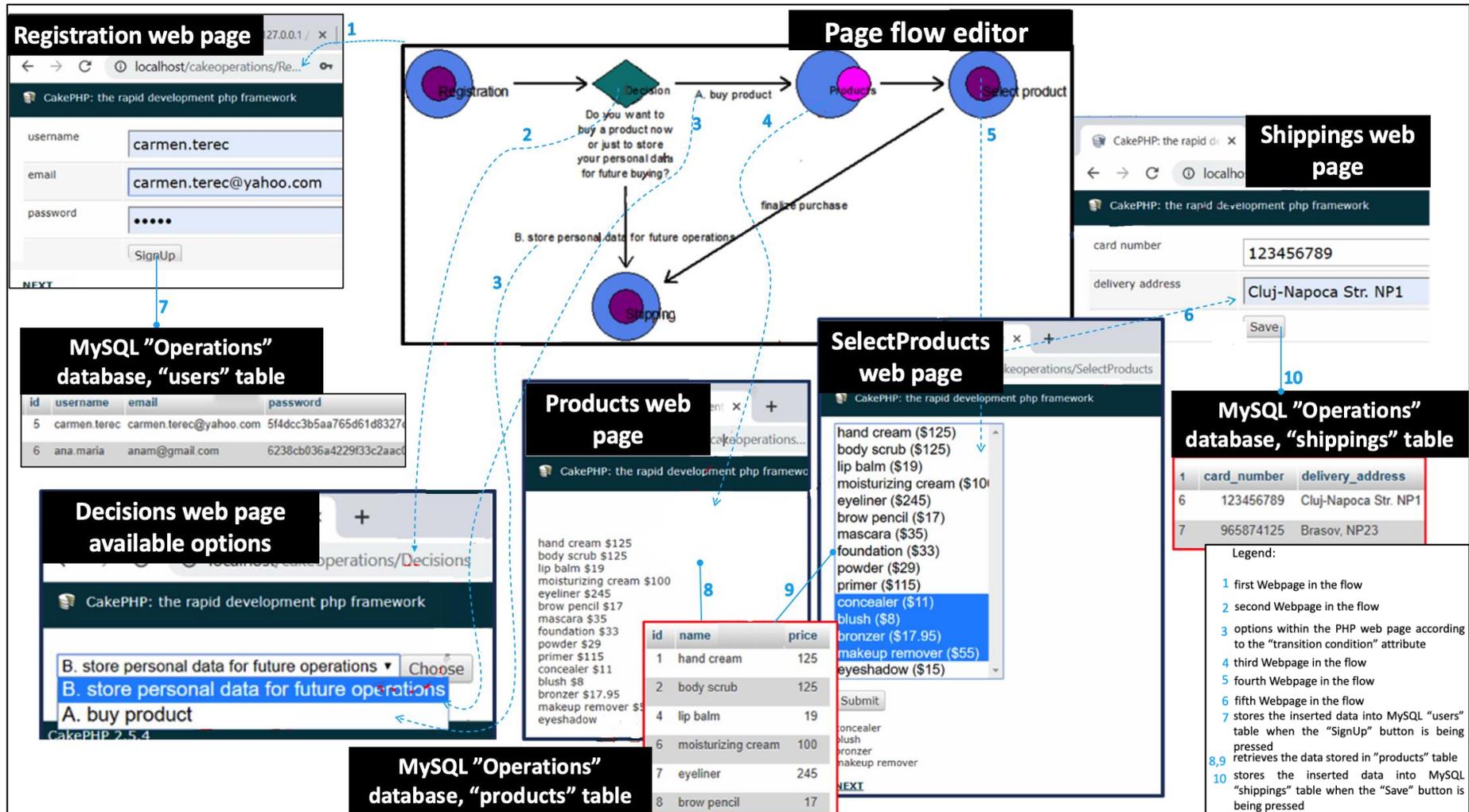
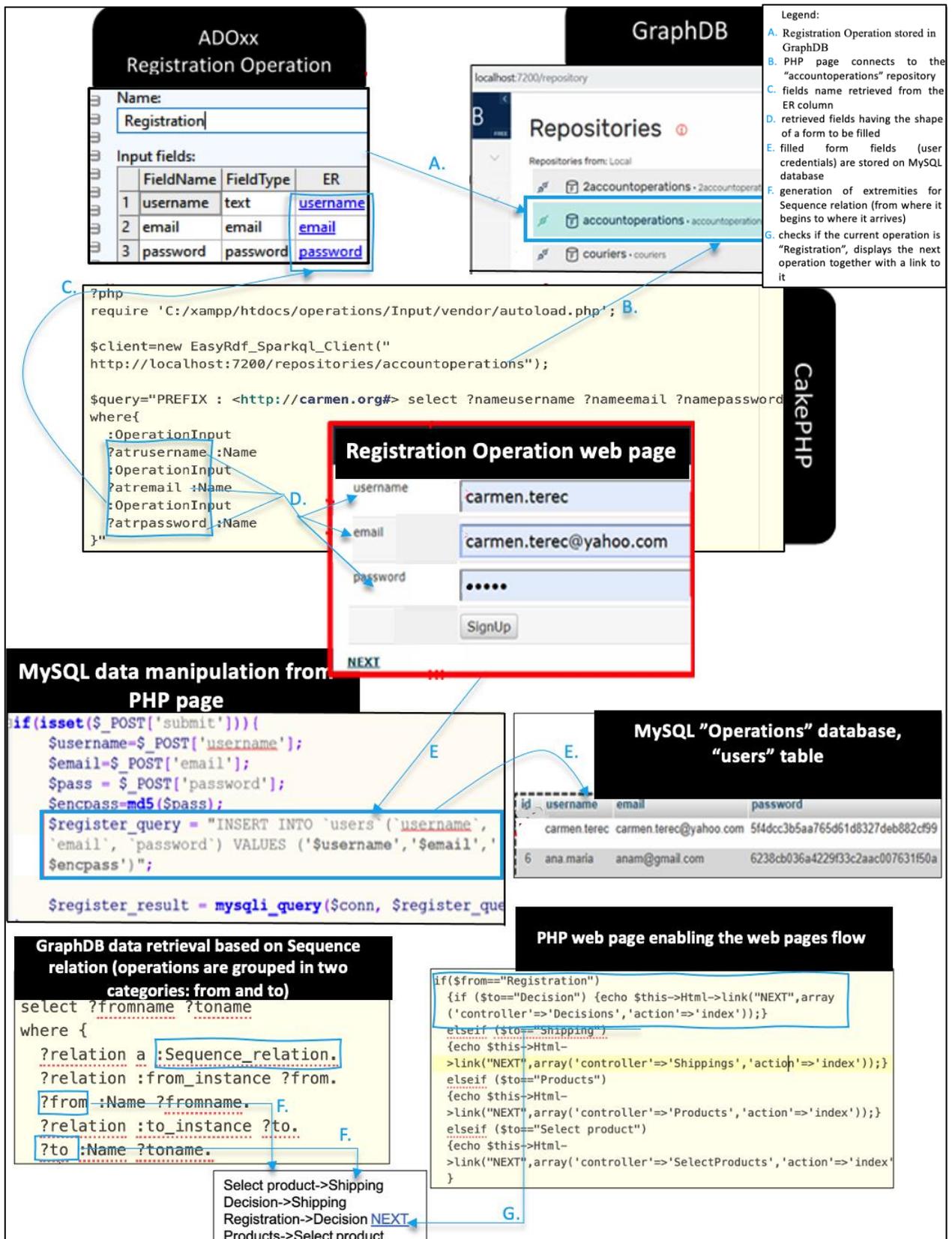


Figure 9. Operations mapped on associated PHP pages



**Figure 10.** Operations within Registration Page

The Decision webpage is generated based on the created "Decision" operation within the Operations model. The page looks for Sequence relations outgoing from the operation Decision and their transition conditions are provided as a selection list Based on the selection, the button "NEXT" for the corresponding operation page is generated. As presented in Figure 9, if the user

chooses option “A. buy product” he is redirected to the Products page, otherwise to the Shipping page.

The Products webpage is generated based on the created “Products” operation within the Operations model. A variable takes the query which the modeler types into the “Query text” area and runs it on the Products table (from the MySQL “operations” database). The name of the products is concatenated with their price for a better view and the “NEXT” button is generated to direct the user from the current operation page to the following one.

The Select product webpage is generated based on the created “Selected product” operation within the Operations model. The PHP page retrieves the products and their price (from products table). Further on, they are grouped in a dropdown list where the user can perform selection of products. In the “Products web page” area, the products and their price are displayed, while in the “SelectProducts webpage” section the insertion of selected products is performed (into the “selections” table from MySQL) by clicking the submit button; After the selected products are stored into the corresponding table the button (NEXT) leading to the next operation page is generated.

## 7 Testing and Evaluation

The testing and evaluation was inspired by the artifact evaluation criteria discussed in [28].

A first evaluation was made by analyzing the usability. Because the operators are humans, usability informs on the complexity of the interfaces and how operators deals with them. The diagrammatic modeling tool was tested by the PHP software’s architect. Prior to giving the modeling software to the website architect in order to test it, the necessary instructions have been presented to him. The instructions include defining the types of operations which can be created, the purpose for which each operation type can be used and how operations can be connected in order to have a logical flow. For creating the diagram described above, the number of clicks given per each operation can be seen in Table 2, counting a number of 103 clicks in total. This gives an impression on the usability and low-code (diagrammatic) effort behind developing the sequence of shown webpages.

**Table 2.** Number of clicks given for each operation and relation creation

<b>Operation Name</b>	<b>Number of clicks given to create the operation</b>
Registration	21
Decision	6
Products	6
Select product	21
Shipping	21
Sequence relation (relation)	19
Transition condition(relation)	
<b>Total number of clicks</b>	<b>103</b>

The developers of the model-driven website tested it continuously all along the development process. In terms of limitations, because MVC connects to MySQL tables, in its Model section, the used database type was set to MySQL, but in parallel the connection to GraphDB had to be performed in each of the PHP files. This means that the approach would benefit from MVC frameworks that are capable of mixing object-relational database mappers with object-RDF mappers (much less common). As a synthetic evaluation, a SWOT analysis is summarized in Table 3.

**Table 3.** SWOT analysis for the proposed artifact

<b>STRENGTHS</b>	<b>WEAKNESSES</b>
<ul style="list-style-type: none"> <li>• Possibility to generate template-based webpages from diagrammatic descriptions</li> <li>• Low-code development of the website</li> <li>• Demonstrates the feasibility of the MASE engineering approach and an expository instance of “technology-specific” modeling</li> </ul>	<ul style="list-style-type: none"> <li>• Operations are of limited types and have limited field types</li> <li>• Website only partially generated, currently not including state management</li> <li>• The ER component of the metamodel is oversimplified to focus the current presentation on its original contribution</li> </ul>
<b>OPPORTUNITIES</b>	<b>THREATS</b>
<ul style="list-style-type: none"> <li>• Tailoring this work for more standard models (BPMN, Petri Nets, etc.)</li> </ul>	<ul style="list-style-type: none"> <li>• Low-code platforms are becoming successful without relying on conceptual modeling environments [29]; we consider, however, that the lessons learned over time in the metamodeling community can facilitate the development of such platforms</li> </ul>

## 8 Conclusions

This article presented the design and engineering details for creating a demonstrative artifact comprising a “technology-specific” modeling tool successfully applied for developing typical components in a PHP website. This showcases the applicability of the model-aware software engineering approach to common Web development tasks, assuming expertise with modeling method engineering and RDF information retrieval.

Future work will improve the generality of the proposal by identifying constructs common to multiple MVC frameworks regardless of language, so that not only PHP developers are supported. Since MVC development is a pattern-driven approach, it is adequate for a model-driven engineering approach where commonly used patterns can be configured on a diagrammatic level to enable a new approach to low code development.

## References

- [1] A. Harkai, M. Cinpoeru, and R. A. Buchmann, “The ‘What’ facet of the Zachman framework: a Linked Data-driven Interpretation,” *Proceedings of CAISE 2018 Workshops, Lecture Notes in Business Information Processing*, Springer, vol. 316, pp. 197–208, 2018. Available: [https://doi.org/10.1007/978-3-319-92898-2\\_17](https://doi.org/10.1007/978-3-319-92898-2_17)
- [2] R. A. Buchmann, M. Cinpoeru, A. Harkai, and D. Karagiannis, “Model-Aware Software Engineering – A Knowledge-based Approach to Model-Driven Software Engineering,” *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, ScitePress, vol. 1, pp. 233–240, 2018. Available: <https://doi.org/10.5220/0006694102330240>
- [3] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*, Springer, 2014. Available: <https://doi.org/10.1007/978-3-662-43839-8>
- [4] D. Karagiannis, “Agile modeling method engineering,” *Proceedings of the 19th Panhellenic Conference on Informatics*, ACM, pp. 5–10, 2015. Available: <https://doi.org/10.1145/2801948.2802040>
- [5] W3C, RDF. Available: <https://www.w3.org/RDF/>. Accessed on December 1, 2020.
- [6] CakePHP, CakePHP. Available: <https://cakephp.org>. Accessed on December 3, 2020.
- [7] Ontotext, GraphDB. Available: <https://graphdb.ontotext.com>. Accessed on December 3, 2020.
- [8] BOC, ADOxx Metamodelling Platform. Available: <https://www.adoxx.org/live/about>. Accessed on December 2, 2020.
- [9] D. Moody, “The physics of notations: Toward a scientific basis for constructing visual notations in software engineering,” *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009. Available: <https://doi.org/10.1109/TSE.2009.67>
- [10] D. L. Moody, P. Heymans, and R. Matulevičius, “Visual syntax does matter: Improving the cognitive effectiveness of the i\* visual notation,” *Requirements Engineering*, vol. 15, no. 2, pp. 141–175, 2010. Available: <https://doi.org/10.1007/s00766-010-0100-1>

- [11] S. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, John Wiley & Sons, Inc., 2002.
- [12] R. Hu, Z. Wang, J. Hu, J. Xu, and X. Jun, “Agile Web development with Web framework,” *4th International Conference on Wireless Communications, Networking and Mobile Computing, IEEE*, pp. 1–4, 2008. Available: <https://doi.org/10.1109/WiCom.2008.2960>
- [13] D. Bork, R. Buchmann, D. Karagiannis, M. Lee, and E. T. Miron, “An Open Platform for Modeling Method Conceptualization: The OMiLAB Digital Ecosystem,” *Communications of the Association for Information Systems*, vol. 44, pp. 673–697, 2019. Available: <https://doi.org/10.17705/1CAIS.04432>
- [14] K. Kumar and R. Welke, “Methodology Engineering: a proposal for situation-specific methodology construction,” *Challenges and strategies for Research in Systems Development*, John Wiley & Sons, Inc., pp. 257–269, 1992.
- [15] D. Karagiannis, H. Mayr, and J. Mylopoulos, *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, Springer, 2016. Available: <https://doi.org/10.1007/978-3-319-39417-6>
- [16] S. Kelly, K. Lyytinen, M. Rossi, and J. P. Tolvanen, “MetaEdit+ at the Age of 20,” *Seminal Contributions to Information Systems Engineering*, Springer, pp. 131–137, 2013. Available: [https://doi.org/10.1007/978-3-642-36926-1\\_10](https://doi.org/10.1007/978-3-642-36926-1_10)
- [17] Eclipse, Sirius. Available: <https://www.eclipse.org/sirius/overview.html>. Accessed on December 5, 2020.
- [18] D. Karagiannis, R. A. Buchmann, P. Burzynski, U. Reimer, and M. Walch, “Fundamental conceptual modeling languages in OMiLAB,” *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, Springer, pp. 3–30, 2016. Available: [https://doi.org/10.1007/978-3-319-39417-6\\_1](https://doi.org/10.1007/978-3-319-39417-6_1)
- [19] OMiLAB, BEE-UP. Available: <https://www.omilab.org/activities/bee-up.html> Accessed on December 5, 2020.
- [20] OMG, IFML standard. Available: <https://www.ifml.org>. Accessed on December 7, 2020.
- [21] E. Diaz, J. I. Panach, S. Rueda, and O. Pastor, “Towards a method to generate GUI prototypes from BPMN,” *Proceedings – International Conference on Research Challenges in Information Science (RCIS)*, IEEE, pp. 1–12, 2018. Available: <https://doi.org/10.1109/RCIS.2018.8406675>
- [22] R. A. Buchmann and D. Karagiannis, “Agile modelling method engineering: Lessons learned in the ComVantage research project,” *The Practice of Enterprise Modeling. PoEM 2015. Lecture Notes in Business Information Processing*, Springer, vol. 235, pp. 356–373, 2015. Available: [https://doi.org/10.1007/978-3-319-25897-3\\_23](https://doi.org/10.1007/978-3-319-25897-3_23)
- [23] E. Laurenzi, K. Hinkelmann, and A. van der Merwe, “An agile and ontology-aided modeling environment,” *The Practice of Enterprise Modeling. PoEM 2018. Lecture Notes in Business Information Processing*, Springer, vol. 335, pp. 221–237, 2018. Available: [https://doi.org/10.1007/978-3-030-02302-7\\_14](https://doi.org/10.1007/978-3-030-02302-7_14)
- [24] W3C, SPARQL. Available: <https://www.w3.org/TR/rdf-sparql-query/>. Accessed on December 1, 2020.
- [25] MySQL AB, MySQL. Available: <https://www.mysql.com>. Accessed on December 6, 2020.
- [26] N. J. Humfrey, EasyRDF. Available: <https://www.easyrdf.org>. Accessed on December 2, 2020.
- [27] D. Bork, D. Karagiannis, and B. Pittl, “A survey of modeling language specification techniques,” *Information Systems*, Elsevier, vol. 87, 2020. Available: <https://doi.org/10.1016/j.is.2019.101425>
- [28] N. Prat, I. Comyn-Wattiau, and J. Akoka, “Artifact evaluation in information systems design-science research – A holistic view,” *PACIS 2014 Proceedings*, AIS eLibrary, 2014. Available: <https://aisel.aisnet.org/pacis2014/23>
- [29] Mendix Platform. Available: <https://www.mendix.com>. Accessed on December 2, 2020.