AutoCRUD
Automating IFML Specification of CRUD Operations

Roberto Rodriguez-Echeverria, José M. Conejero, Juan C. Preciado and Fernando Sanchez-Figueroa
School of Technology, University of Extremadura, Av. Universidad, Caceres, Spain

Keywords: Model Driven Web Engineering, Cost Reduction, CRUD, Model Driven Software Development.

Abstract: Development and deployment technologies for data-intensive web applications have considerably evolved in the last years. Domain specific frameworks or Model-Driven Web Engineering approaches are examples of these technologies. They have made possible to face implicit problems of these systems such as quick evolving business rules or severe time-to-market requirements. Both approaches propose the automation of redundant development tasks as the key factor for their success. The implementation of CRUD operations is a clear example of repetitive and recurrent task that may be automated. However, although web application frameworks have provided mechanisms to automate the implementation of CRUD operations, Model-Driven Web Engineering approaches have generally ignored them and its automation has not been properly faced yet. This paper presents AutoCRUD, a WebRatio plug-in that automates the generation of CRUD operations in OMG IFML (Interaction Flow Modelling Language) standard. The suitability of this tool has been evaluated by its application into several real projects developed by a software company specialized in model-driven web application development. The results obtained present evidences of the significant productivity improvement obtained by the tool, which almost completely removes the developer time dedicated to CRUD operation implementation.

1 INTRODUCTION

Model-Driven Web Engineering (MDWE) (Koch et al., 2008) approaches provide methodologies and tools for the design and development of most kinds of web applications. They address different concerns by using separate models (navigation, presentation, data, etc.), and are supported by model compilers that automatically produce most of the application's Web pages and logic code. The benefits of using MDWE are clear from different points of view such as team productivity, software quality or adaptation to evolving technologies (Rossi et al., 2007) (Vuorimaa et al., 2015).

Among the different MDWE approaches, it is worth mentioning IFML (Interacting Flow Modelling Language) (Brambilla and Fraternali, 2015), an OMG standard for the development of data-intensive applications that has become a reference in industrial developments (Casteleyn et al., 2014); (Toffetti et al., 2011). Its successful development tool, WebRatio, allows the edition and validation of IFML models but also, and even more important, allows the generation of the final application code for a specific technological deployment platform, reducing the time-to-market and the development effort for these applications.

Focusing on the development effort, one of the most redundant tasks in data-intensive web application development is the implementation of CRUD operations. As Martin Fowler argued, "disappointing as it is, many of the use cases in an enterprise application are fairly boring 'CRUD' (create, read, up-date, delete) use cases on domain objects" (Fowler, 2002).

However, and surprisingly, while several frameworks such as Ruby on Rails (http://rubyonrails.org/), Django (https://www.djangoproject.com/), MonoRail (http://www.castleproject.org/projects/monorail/), or Catalyst (http://www.catalystframework.org/), just to cite a few, have adopted solutions to optimize CRUD operations, MDWE approaches (i.e., IFML/WebRatio) have not provided yet an automatic tool to perform these tasks; even though there are works claiming a significant productivity gain (more than 90%) when these tasks are automated by model-driven techniques (Mbarki and

Erramdani, 2008); (Papotti et al., 2013).

This paper presents AutoCRUD, a WebRatio plug-in developed for the automatic IFML specification of CRUD operations. The objective of this paper is twofold. On one hand, presenting the development of the plug-in and its main features, and, on the other hand, showing how it impacts on the effort optimization in the development of real projects. For the latter purpose, we have relied on the collaboration of an external company specialized in the development of data-intensive web applications by means of WebRatio, obtaining improvements of 95% and more in the time dedicated to CRUD specification. The AutoCRUD plug-in can be freely downloaded at http://www.homeria.com/autocrud. WebRatio can be freely downloaded at http://www.webratio.com.

The rest of the paper is organized as follows. Section 2 presents motivation and related work, highlighting both, studies about optimization in MDWE and proposals for automatic generation of CRUD operations. Section 3 introduces AutoCRUD development, its main features and its main use cases. Section 4 shows the main results obtained when applying the plug-in to real projects. Finally, Section 5 summarizes the main contributions of this work.

2 MOTIVATION AND RELATED WORK

Optimization of development effort in the Web Engineering domain has been addressed by several works. In (Fatolahi and Somé, 2014) the authors focused on the assessment of the impact of using a Model Driven Web methodology with respect to traditional web developments. They observed an important productivity gain by using their Model Driven approach. In (Martinez et al., 2014) the authors also compared the use of a Model-Driven Web Engineering approach - OOH4RIA (Melia et al., 2008) - in web information systems development with a code-centric one (implementation in .NET). In particular, they focused on maintainability characteristics of these systems. This work was an extension of the work presented in (Martinez et al., 2011), where authors performed a similar study but focusing on WebML (Ceri et al., 2000); (Ceri et al., 2002) as MDWE approach and PHP as code-centric alternative. In both works, authors observed that the utilization of the MDWE approach provided important maintainability improvements with respect to the code-centric implementation, e.g., OOH4RIA improved the actual efficiency of the changeability tasks in 317 times and also improved the effectiveness of the changeability by up to 27%. Nevertheless, although all these works reveal a clear optimization of development effort, they do not address CRUD operations specifically, which is the main objective of other works such as (Mbarki and Erramdani, 2008) and (Papotti et al., 2013).

In (Mbarki and Erramdani, 2008) the authors presented an approach based on model transformations that automatically generates the CRUD operations for a web system taking as input class diagrams based on UML profiles. In (Papotti et al., 2013) the authors also evaluated the productivity improvements obtained by a model-driven approach that automatically generates the CRUD operations source code for a Web information system. This approach also takes as input the UML class diagrams for the system. By using the model-driven approach, the authors observed an important development time reduction (up to 90,98%). They also surveyed the developers about the difficulties found compared to the manual coding approach and obtained better results for the model-driven approach. However, neither (Papotti et al., 2013) nor (Mbarki and Erramdani, 2008) do integrate with an MDWE approach.

Finally, the automatic generation of CRUD operations has been the focus of different works at different levels of abstraction: code (e.g., grocery CRUD for PHP, http://www.grocerycrud.com) or frameworks (Ruby on Rails, Django, MonoRail, or Catalyst). These frameworks provide specific tools to optimize CRUD operations, like software scaffolding toolkits that allow generating the structural parts of the applications expressed in some simplistic specification language (normally XML or YAML). Once the code is generated, it has to be manually refined by developers, discarding the initial specifications. Such approaches force a specific platform and architecture with the advantages of automatic code generation to speed-up the initial stages of the development. However, these proposals are at a lower level of abstraction than Model-Driven approaches, losing the benefits obtained by these ones, such as the independence of specific platforms and, in general, the optimization of development efforts outlined at the beginning of this section.

To our knowledge, and although the advantages are clear, there is not a MDWE approach incorporating the automatic generation of CRUD operations and this is the rationale for this work.

3 AutoCRUD

In this section, we first briefly provide an overview of the OMG IFML standard and, then, AutoCRUD is introduced by means of illustrative examples resembling its behaviour.

3.1 IFML Overview

IFML is a modelling language standardized by the OMG (Object Management Group) to represent an application front-end independently of the implementation technology or target device. Basically, IFML defines a set of visual elements to represent the user interaction and the front-end behaviour. WebRatio has led the language definition and WebML has been used as the conceptual base, which benefited its definition thanks to the experience of WebRatio using WebML. The language was adopted as a standard by the OMG in March 2013 (changing its name to IFML). And in March 2014, OMG Architecture Board formally adopted the specification of IFML 1.0 (Brambilla and Fraternali, 2015). Among other improvements, it is worth to note that the binding with the business and content models has been generalized to allow the usage of non-UML models. The IFML language defines the following core elements: View Container, View Component, Binding, Parameter, Event, Action, Navigation Flow, and Data Flow.

3.2 AutoCRUD Overview

Herein, based on the main functionality of scaffolding tools in web application frameworks, a new tool has been developed, as a plug-in for WebRatio, to considerably reduce the specification effort of CRUD operations from data entities in IFML. This plug-in provides the engineer with the proper functionality to automatically generate any CRUD operation in IFML from concrete data entities. Actually AutoCRUD has been defined to behave as an orchestrator of the different functionalities of WebRatio by calling the right components in the right moment for such generation. The benefits of that approach are two-fold: (1) it allows a more durable integration with WebRatio/IFML functionalities; and (2) the engineer can use AutoCRUD in any moment of the development lifecycle seamlessly.

As depicted in Figure 1, basically, the engineer must simply follow the next 3 steps: (1) selecting a concrete entity from the data model; (2) choosing the desired CRUD operations (create, read, update, delete or all-in-one); and (3) providing the proper binding for the parameters of the CRUD operation (e.g., what is the element to be deleted).

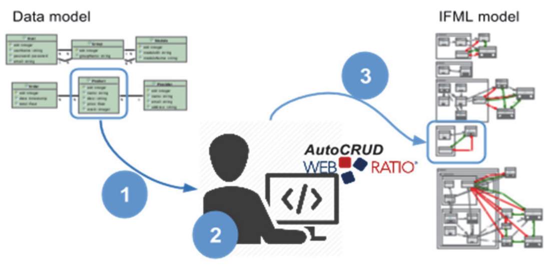


Figure 1: AutoCRUD functionality.

3.2.1 AutoCRUD Architecture

As Figure 2 presents, AutoCRUD is a tool built on top of WebRatio. According to traditional 3-layer software architecture, the tool has been defined inside domain and presentation layers. Concerning the presentation, the user interface is basically organized in a few dialogs allowing data entity and site view selection, on one hand, and CRUD operation management, on the other hand. Regarding the domain layer, a facade layer has been defined on top of WebRatio API to orchestrate its functionality in order to generate the IFML specification of the CRUD operations. This facade is formed by two main components: the manager and the engine. The manager is responsible for storing the configuration of the CRUD operation to be generated. The engine, invoked by the manager, is in charge of translating such configuration to a concrete sequence of WebRatio functionality calls that eventually generates the IFML specification.

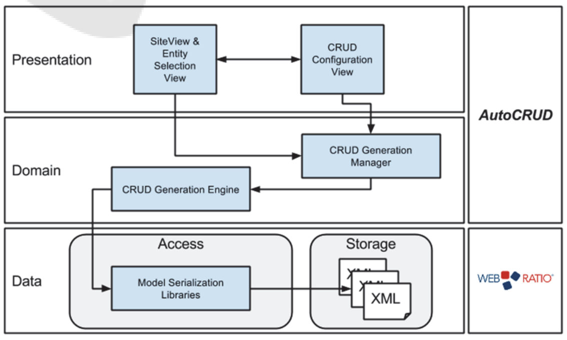


Figure 2: AutoCRUD architecture.

3.2.2 Categorization of Use Cases

In order to simplify the presentation of all the different use cases supported by the tool, a previous categorization has been defined based on two orthogonal dimensions:

- Type of CRUD operation whose IFML specification must be generated. This dimension has the following possible values: create, read, update, delete or all-in-one. The last one is a special compact case whose specification does not match exactly to the join of the specification of every CRUD operation individually.
- Cardinality of the relationship between the data entities involved in the CRUD operation. Although just one data entity is required to start the generation process, it may be interesting to consider other data entities related to the first one. This dimension represents then such possibility. The possible values of this dimension are the following: no relationship (the data entity is considered in isolation or it has not been defined any relationship), onetomany relationship (1-N), and manytomany relationship (N-M). Obviously, inside a single case, this dimension may get different values, e.g., a data entity A holds an onetomany relationship with a data entity B and, at the same time, it holds an onetomany relationship with a data entity C. In such cases, the tool allows selecting any of those relationships or both of them. In the rest of the paper, the tool behavior is illustrated just considering the single cases because the complex ones are just composition of those ones.

Table 1 shows the collection of use cases supported by the tool and organized according to the dimensions explained before. For the sake of brevity, just one of them is next detailed in order to illustrate the actual extension of the approach, which appears in Table 1 in bold font (in the first row C stands for Create, R for Read, U for Update and D for Delete). The interested reader may find the details of every specific case in http://www.homeria.com/autocrud.

Table 1: AutoCRUD use cases.

	C	R	U	D	All
No	C-no	R-no	U-no	D-no	All-no
1-N	C-1N	R-1N	U-1N	D-1N	All-1N
N-M	**C-nM**	R-nM	U-nM	D-nM	All-nM

Following the selected use cases are explained with detail. And, at the end of the section, a summary table collecting the total number of elements needed to specify all the use cases is presented.

3.2.3 Illustrative Example

Before describing the selected use cases, it is mandatory to explain the sample data model used, shown in Figure 3. As it can be noted, an excerpt of the data model of a real application has been used to illustrate the explanation of the different use cases. This sample data model contains common entities of an e-commerce application: The document margins must be the following:

- Product. It holds one relationship with each of the other two entities.
- Provider. It holds an onetomany relationship with Product, hence each product has only one provider while a provider may provide several products.
- Order. It holds a manytomany relationship with Product, hence an order may contain several products and a product may appear in different orders.

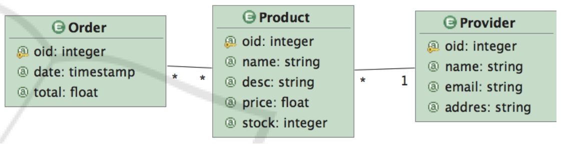
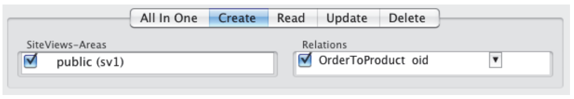


Figure 3: Sample data model.

3.2.4 C-nM Case

This case allows illustrating the elements involved in a Create operation, defined over a data entity (Order), which keeps a manytomany relationship with another data entity (Product) and this relation must be considered in the specification of such operation.

Figure 4 shows the tool configuration for this particular case. Once the Order data entity has been selected, the user just needs to navigate to the Create tab to generate the corresponding operation. This tab allows the user to select the SiteView or Area in which the create operation specification for the Order entity will be generated. Moreover, the OrderToProduct relationship may be selected by the user to be considered by the tool in the specification generation. Note that the involved OrderToProduct relationship is derived from the data model (see Figure 3).



SiteViews-Areas
☑ public (sv1)
Relations
☑ OrderToProduct oid

Figure 4: AutoCRUD set-up for C-nM case.

The IFML specification generated by the tool for this case is shown in Figure 5. As main container, a new Page (CreateOrder) has been generated. This page contains 3 different IFML units: a selector unit (EntityOrder), an entry unit (FormOrder) and a message unit (MessageOrder). The form (entry unit) collects the data of every product involved in a specific order, as well as other general attributes as the date and the total price. Note that the form is connected to a selector unit that provides the form with the data of the available products. Once all the data is collected the form submission may trigger the execution of a create operation unit (CreateOrder) and a connect operation unit (OrderToProductOrder) that stores the new order in the database. Additionally, the message unit is used to show the user any message steaming from the process.

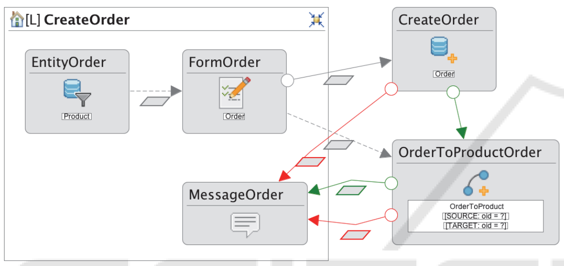


Figure 5: IFML specification of the C-nM case.

Table 2 shows the number of every type of IFML elements generated for the specification of this operation in this specific case. For the sake of this work, IFML elements have been classified into three main groups: links, units and couplings. The tool uses 15 different IFML elements for this case. As it may be observed, most of the considered IFML elements have been used for the specification of the Create operation.

Table 2: Aggregated numbers of IFML elements for the C-nM case.

	Links	Units	Couplings
C-nM	7	5	3

3.2.5 Summary of IFML Generation

Concerning the rest of the previously identified cases, the generation of the IFML elements follows a similar process. Although concrete details of their generation are not discussed in this work, Table 3 presents a summary of the total number of IFML elements generated for the specification of each CRUD operation in every use case considered. A developer has to use more than 18 elements in average to specify a CRUD operation with IFML. In other words, AutoCRUD is saving him the average cost of 18 elements per CRUD operation in the specification of a web application with IFML. Additionally, as shown, leaving apart AllinOne cases, create and update operations present a more complex specification than remove and display operations. In fact, AutoCRUD always produce the same specification for remove and display operations independently of the cardinality dimension, basically, because relationships are not relevant for those kind of operations at this level.

Table 3: Number of IFML elements for all the cases.

	Links	Units	Couplings	Total
C-no	3	4	1	8
R-no	1	3	1	5
U-no	5	6	3	14
D-no	3	4	1	8
All-no	16	19	6	37
C-1N	4	5	2	11
R-1N	1	3	1	5
U-1N	6	7	4	18
D-1N	3	4	1	8
All-1N	19	16	9	44
C-nM	7	5	3	17
R-nM	1	3	1	5
U-nM	13	10	7	30
D-nM	3	4	1	8
All-nM	27	18	13	58

4 INDUSTRIAL VALIDATION

In order to validate the utility and applicability of the plug-in presented, this section shows an evaluation of its application to different projects developed by a Spanish software company, Homeria SL. This company is an official partner of WebRatio (http://www.webratio.com/site/content/es/partners) and has developers certified in the development of web applications by using the technologies and methodologies provided by IFML and, in particular, WebRatio. It has developed more than 100 projects in the last 9 years and it relies on an important set of clients.

The main goal of this evaluation is to analyse the cost saving obtained by incorporating the tool presented here into the development process of real projects implemented by the company. In this analysis, the developer time has been considered as the cost measurement unit so that the minor the development time is, the lower cost the Web system has. In order to obtain some base time measurements, the next procedure has been followed:

- A group of developers of the company was selected for the analysis. All these developers had a similar experience in developing data intensive web applications with WebRatio.
- The developers were responsible for developing the CRUD operations over different data entities (and in different situations) in the projects that the company was involved at the moment of performing this study.
- The time spent in the specification of the needed IFML entities for each CRUD operation was measured.
- Considering similar development times, as aforementioned, the IFML elements were classified into three main groups: links, units and couplings.

The average of the development time for the different groups was obtained. These values are shown in Table 4 and are used as a reference in the measurements performed during the study. Note that, although these values could vary in terms of developers' expertise, it has been estimated that the proportions among them would keep constant. Thus, this study could be replicated in other companies just by adapting the base time values for the development time to the expertise of the particular development team.

Table 4: Developer time average per group of IFML elements.

Group	Link	Unit	Coupling
Dev. Time (secs.)	24	112	66

According to these times and the values shown in Table 3, Table 5 presents, on one hand, the costs related to the manual specification of each case considered for the CRUD operations, and on the other hand, shows the time (in average) spent by a developer in specifying each case for the CRUD operations by means of the plug-in. Note that the times spent by the plug-in in automatically generating the final code are not shown since they are insignificant (once the operations have been specified).

Once the base time measurements were obtained, the study was focused on the analysis of projects previously developed by the company and with the source models available. To this purpose, a quantitative analysis was carried out where the needed IFML elements for the specification of the CRUD operations in the different projects were identified.

As an example of the obtained data, Table 6 shows an excerpt for a reduced set of 6 real projects recently developed in the company. This table shows for each project: 1) the size; 2) the number of CRUD operation cases; 3) the total number of CRUD operations; 4) the total number of data entities in the data model; 5) the total number of IFML elements used to specify all the CRUD operations (classified as links, units and couplings); 6) the total time (in hours) dedicated to the project. Based on row 5) (number of elements needed for representing the CRUD operations) and the time spent in specifying each operation (shown in table 7), the total time spent in manually implementing these operations has been calculated for each project (7). Based on row 6) and 7) the percentage of time dedicated to the CRUD operations for each project is shown in row 8). Likewise, considering the time spent in specifying each CRUD operation by using the plug-in, the time employed in automatically generating all the CRUD operations for each project has been calculated (9). Finally, the table shows the difference (10) between both costs -manually developed vs. automatically generated- showing the benefits in both hours and percentage.

As it may be observed in Table 6, the benefit (and, thus, the saved time) by using the plug-in is higher than 95% of the time dedicated to CRUD operations in all the projects. These data show a clear evidence of the great productivity improvement provided by the tool presented here, which almost completely removes the time dedicated to the specification of CRUD operations from the projects. Moreover, Table 6 shows how this improvement seems to be correlated with project size. In other words, the bigger the project is, the higher the improvement obtained. As an example, the improvement obtained in the biggest project is 99,88% that suggests that the time dedicated to CRUD specification is insignificant with respect to the total time of the project.

However, obviously, these data must be considered also taking into account the percentage of time dedicated to CRUD operations in each project.

Observe that, in the projects analysed, these percentages range from 6,14% to 9,69% of the total time (row 8 in Table 6). That means that, for instance, the plugin is able to provide a reduction of the total time of the project of almost 10% in the biggest project. In addition, this conclusion becomes even more important if we consider that the total time of the project involves many tasks that are not directly related with the IFML specification, such as user interface design or scripting implementation. To put in another way, the total time of CRUD specification saved in the project may be compared with the total time dedicated to IFML specification and, then, the results obtained are even more promising. In particular, the percentage of time dedicated to the IFML specification in each project has been presented in Table 7. This table also shows the percentage of the IFML specification dedicated to CRUD operations so that the reduction of the time dedicated to IFML in each project has been calculated. This reduction varies from 14,99% to 28,15%. That implies that, considering that the time dedicated to CRUD specification is practically removed from the projects, an average reduction of the 19,75% of the time dedicated to IFML specification is achieved in these projects.

5 CONCLUSIONS

This paper has presented AutoCRUD, a tool that automates the specification of CRUD operations in IFML. The tool has been developed as a WebRatio plug-in so that it may be easily integrated into industrial developments, bridging a gap that MDWE approaches had not deal with yet, i.e., the optimization of the specification of repetitive and recurrent CRUD operations. The benefits obtained by the tool have been evaluated by applying it to real projects developed by an external software company. By this analysis, we observed important evidences of the optimization gain obtained by the tool but also of its scalability since the results are even better when biggest projects are considered. Moreover, the number of errors usually introduced during the specification of these operations has been dramatically reduced.

As further work, we plan to follow several research lines. Firstly, we want to apply a similar approach to other repetitive development tasks that are being identified in WebRatio. Secondly, we are working on the development of some heuristics to guide an algorithm on the automatic generation of the most likely useful CRUD operations for every data entity.

ACKNOWLEDGEMENTS

This work has been funded by Junta de Extremadura (Consejería de Economía e Infraestructuras - GR15098) and European Regional Development Fund (ERDF).

REFERENCES

Brambilla, M, Fraternali, P., 2015. Interaction Flow Modeling Language – Model Driven UI Engineering of Web and Mobile Apps with IFML, *Morgan Kaufmann / OMG Press*.

Casteleyn, S., Garrigós, I., Mazón, J., 2014. Ten Years of Rich Internet Applications: A Systematic Mapping Study, and Beyond. In *ACM Transactions on the Web* 8, 3, 1-46.

Ceri, S, Fraternali, P, Bongio, A, Brambilla, M, Comai, S, Matera, M, 2002. Designing Data-Intensive Web Applications. *Morgan Kaufmann Publishers Inc*.

Ceri, S., Fraternali, P., Bongio, A., 2000. Web modeling language (WebML): a modeling language for designing Web sites. In the *International Journal of Computer and Telecommunications Networking*, 33, 1, 137–157.

Fatolahi, A, Somé, A., 2014. Assessing a Model-Driven Web Application Engineering Approach. In *Journal of Software Engineering and Applications 7*, 360-370.

Fowler, M., 2002. *Patterns of Enterprise Application Architecture*. Addison Wesley Signature Series.

Koch, N, Meliá-Beigbeder, S, Moreno-Vergara, N, Pelechano-Ferragud, V, Sánchez-Figueroa, F, Vara-Mesa, JM, 2008. Model Driven Web Engineering. In the *European Journal of the Informatics Professional*, 9, 2, 40-45.

Martinez, Y, Cachero, C, Matera, M, Abrahao, S, Luján, S. 2011. Impact of MDE approaches on the maintainability of web applications: an experimental evaluation. In *Lecture Notes in Computer Science*.

Martínez, Y, Cachero, C, Meliá, S. 2014. Empirical study on the maintainability of Web applications: Model-driven Engineering vs Code-centric. In *Empirical Software Engineering*.

Mbarki, S, Erramdani, M., 2008. Toward automatic generation of mvc2 web applications. In *Infocomp Journal of Computer Science*, 7, 4: 84-91.

Melia, S, Gómez, J, Pérez, S, Díaz, O., 2008. A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In *ICWE'08*, *8th International Conference on Web Engineering* (2008).

Papotti, P. E., do Prado, A. F., Lopes, W., Cirilo, C. E., Ferreira, L., 2013. A Quantitative Analysis of Model-Driven Code Generation through Software Experimentation. In *Lecture Notes in Computer Science*, 7908, 321-337.

Rossi, G, Pastor, O, Schwabe, D, Olsina, L., 2007. Web Engineering: Modelling and Implementing Web Applications. *Human-Computer Interaction Series*, Springer-Verlag, London.

Toffetti, G., Comai, S., Preciado, J. C., Linaje, M., 2011. State-of-the Art and trends in the Systematic Development of Rich Internet Applications. In *Journal of Web Engineering*, 10, 1, 70-86.

Vuorimaa, P., Laine, M., Litvinova, E., Shestakov, D., 2015. Leveraging Declarative Languages in Web Application Development. In *World Wide Web Journal*.

Table 5: Developer time (secs.) for the specification of every considered case in AutoCRUD.

Time (secs.)	C-no	R-no	U-no	D-no	All-no	C-1N	R-1N	U-1N	D-1N	All-1N	C-nM	R-nM	U-nM	D-nM	All-nM
Manual	586	426	990	586	2908	5496	788	426	426	1192	2842	5674	926	426	1894
AutoCRUD	1	1	1	1	2	2	1	2	1	3	3	1	3	1	5

Table 6: Results of the 6 projects under evaluation.

Projects		P1	P2	P3	P4	P5	P6
1) Size		Big	Medium	Medium	Medium	Small	Small
2) Different CRUD cases	C-no	115	8	5	3	2	2
	R-no	107	9	4	3	1	1
	U-no	116	7	5	4	2	1
	D-no	112	7	5	5	1	1
	All-no	116	9	5	3	2	2
	C-1N	97	5	4	4	2	1
	R-1N	94	7	4	3	2	1
	U-1N	92	4	4	3	2	1
	D-1N	95	6	3	3	1	1
	All-1N	97	10	4	4	2	2
	C-nM	26	6	3	4	3	3
	R-nM	42	8	2	2	2	2
	U-nM	42	9	6	4	3	2
	D-nM	24	6	3	3	1	1
	All-nM	83	7	6	5	4	2
3) Total CRUD operations		1.258	108	63	63	38	33
4) Entities in the data model		193	84	36	27	14	10
5) Total IFML units	Total Links	11.259	874	512	404	224	177
	Total Units	11.285	848	533	391	222	168
	Total Couplings	5.387	414	250	194	108	109
6) Project total time	Hours	5.380	650	350	230	125	80
7) CRUD operations cost (manually developed)	Secs	1.876.777	143.848	77.388	65.812	34.258	28.382
	Hours	521,33	39,96	21,5	18,28	9,52	7,88
8) Time dedicated to CRUD	%	9,69	6,15	6,14	7,95	7,61	9,85
9) CRUD operations cost (automated with plug-in)	Secs	2.225	1.370	1.339	1.330	1.317	1.307
	Hours	0,62	0,38	0,37	0,37	0,37	0,36
10) Benefit obtained in the time dedicated to CRUD (Manual - Automatic)	Hours	520,71	39,58	21,12	17,91	9,15	7,52
	%	99,88	99,05	98,27	97,98	96,16	95,39

Table 7: Time dedicated to IFML and CRUD specification in each project.

Projects		P1	P2	P3	P4	P5	P6
1) Size		Big	Medium	Medium	Medium	Small	Small
2) Project total time	Hours	5.380	650	350	230	125	80
3) Time for IFML Specification	%	51	41	40	40	36	35
4) Time for CRUD operations	%	9,69	6,15	6,14	7,95	7,61	9,85
5) Time for CRUD operations of the IFML specification	%	19	14,99	15,35	19,87	21,14	28,15