XV Jornadas de Ingeniería del Software y Bases de Datos
JISBD 2006
José Riquelme - Pere Botella (Eds)

BUILDING TOOLS FOR MODEL DRIVEN DEVELOPMENT. COMPARING MICROSOFT DSL TOOLS AND ECLIPSE MODELING PLUG-INS

Vicente Pelechano, Manoli Albert, Javier Muñoz and Carlos Cetina

Department of Information Systems and Computation
Technical University of Valencia
Camí de Vera S/N 46022 Valencia, Spain
e-mail:{pele, malbert, jmunoz, ccetina}@dsic.upv.es, web:http://oomethod.dsic.upv.es

Keywords: Model Driven Development, Code Generation, Domain Specific Languages

Abstract. *The development and wide adoption of CASE Tools that provide Code Generation from Models is one critical requirement to confirm the success of the Model Driven Software Development (MDSD). Main actors in the Software Development Industry are aware that they must provide technology and tools that allow building and/or adapting advanced CASE Tools to completely support MDSD. In this context, IBM and Borland, and other companies, (on the one hand) and Microsoft (on the other hand) are building advanced tools (Eclipse and DSL Tools respectively) that are trying to make easy the development and deployment of tools that support MDSD. Many academics and practitioners are thinking in adopting these tools but they do not have enough information to select one of them. In this paper we present an empirical comparison of both tools. In order to evaluate them, an experiment was developed with 48 last year undergraduate students of computer science engineering. The students were divided into 2 groups for developing a DSL (including code generation) where each group was using a different tool. At the end of the course, the students answered several questions about their experiences. This paper presents some conclusions about the current state of the tools that are extracted from the students' answers.*

1. INTRODUCTION

Currently, Model Driven Software Development (MDSD) seems to be a dream that comes true. Several signs point out that, in a near future, the use of this approach is going to rapidly increase. First, the significant support that MDSD has received from both the MDA, which is promoted by the OMG [1], and from the Software Factories [2], which are promoted by Microsoft. Second, the proliferation of CASE tools that support MDSD-based approaches which claim to be "*MDA compliant*", like ArcStyler [3], OptimalJ [4], Together [5],

AndroMDA [6], Poseidon [7] and ONME [8]. And third, the proliferation of technologies and tools for developing "*your own*" DSDM tools (graphical editors, model transformers, code generators, etc.). Inside this category of tools we can find the projects inside the Eclipse Modeling Project[1] (EMF, GMF, GMT, etc.) and the DSL Tools[2] that are integrated with MS Visual Studio 2005. In this context, companies and research groups evaluate the possibility of developing their own CASE tool for supporting their own DSDM-based approach (following the MDA, Software Factories, Product Lines, Generative Programming or whatever other more *specific* proposal) using one of these tools. We think that a comparative study of the current state of both technologies could be very useful for the community that is working on DSDM.

In this work we present a comparative qualitative study of the Microsoft DSL Tools (February 2006 CTP version) and some Eclipse Modeling Plug-ins (Eclipse 3.2M4, EMF 2.2M4, GMF 1.0M4 and MOFScript 1.1.4). The study has been carried out during the course "*Component Technology, Design Patterns and Code Generation*" (05/06 edition) in the last semester of a 5 years Computer Science degree at the Universidad Politécnica de Valencia (Spain). The experiment involved 48 students of this course which where divided in 2 laboratory groups for developing a domain specific language (including a code generator) where each group was using a different tool. At the end of the course, the students answered a survey that included several questions about their experiences. This paper presents some conclusions about the current state of the tools that are extracted from the students' answers.

The paper is structured as follows: first, both the Microsoft DSL Tools and the Eclipse tools for developing modelling tools are briefly introduced. Next the empirical comparison is presented introducing the experiment design, the questions under study and the analysis of the gathered data. Finally, some conclusions are presented.

2. MICROSOFT DSL TOOLS AND ECLIPSE MODELING TOOLS.

This section presents a general overview of the Microsoft DSL Tools and some of the Eclipse tools (plug-ins, in Eclipse jargon) that provide modelling-related functionality. In order to structure the analysis, our first step is to identify which are the basic facilities that a tool for building MDSD-based tools must provide. We consider that these functionalities are the following:

- Metamodeling Facilities
- Model Persistency
- Graphical Notation Development Tools
- Model to Model Transformation Tools
- Model to Text Transformation Tools

[1] http://www.eclipse.org/modeling/
[2] http://msdn.microsoft.com/vstudio/DSLTools/

2.1 Microsoft DSL Tools

Microsoft DSL Tools is a suite for defining Domain Specific Languages, building a graphical designer and defining code generators in Visual Studio 2005. This toolset is a key part of Microsoft's strategy for model-driven and its Software Factories proposal. DSL Tools, in its February 2006 version, provide a *project wizard* to create a DSL:

- Allows defining and editing a domain specific language through a graphical designer using a proprietary notation. Figure 1 shows a screenshot of this designer.

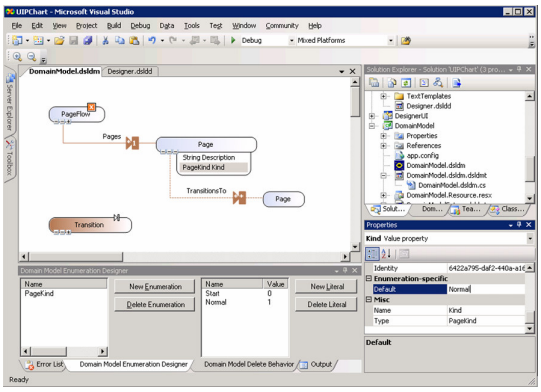

Figure 1. Domain Model Editor supplied by the Microsoft DSL Tools

- Allows defining designer definitions using a proprietary XML format which is the source to generate the code (without any manual programming) that implements the graphical modelers of the DSL.
- Includes a framework to define code generators based on template languages that take an instance of a domain model and generates code based on the template. Figure 2 shows an example of such a template.

```
<#@ template inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransformation"#>
<#@ output extension=".htm" #>
<html>
<body>
. . . . . .
>
<p>Name: <#=this.PageFlow.Name#></p>
<#                                          Sentences
    foreach (Page page in this.PageFlow.Pages )
    {
#>                                          Text Injection
    <p>Page: <#=page.Name#></p>
    <ul>
<#
                    foreach (Page linkedTo in page.TransitionsTo)
                    {
#>
                            <li>Linked to: <#= linkedTo.Name#>
<#
                    }
#>
```

Figure 2. Template Language (Sentences and Code Injection)

2.2 Eclipse Tools

Eclipse was initially the IBM IDE for Java development, which was released as free software. Currently, it is the base platform for many other technologies and projects due to its very powerful modular structure. Most of these plug-ins are related to software development but not necessarily using Java. The Eclipse Modelling Project unifies the modelling related plug-ins that are directly developed by the Eclipse project. Inside this project we can find, among others:

- *Eclipse Modeling Framework* (EMF): a modelling framework and code generation facility for specifying metamodels and managing (creating/editing/saving/loading) models instances. EMF is a Java implementation of the Ecore metamodel. Ecore is a light version of MOF. Figure 3 shows in action an editor of Ecore metamodels.

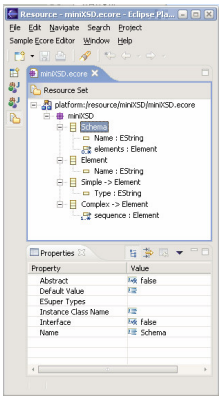

Figure 3. Building an Ecore Metamodel

- *Graphical Modeling Framework* (GMF): provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF. Figure 4 shows an editor that can be generated using the facilities provided by the GMF project.

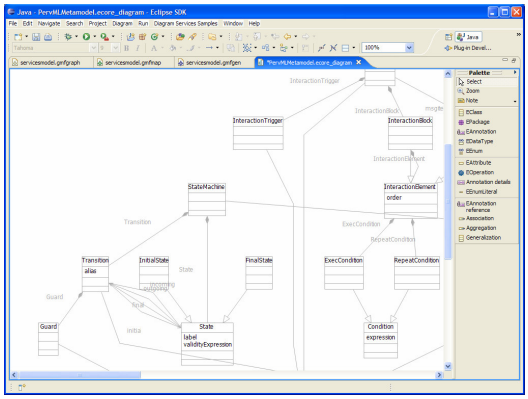

Figure 4. GMF-based editor of Ecore metamodels

- *MOFScript tool*: an implementation of the MOFScript model to text transformation language. This language was submitted to the OMG as response to the "*MOF Model to Text Transformation Language RFP"*. It is being developed in the MODELWARE European Project and it is available as open source. Figure 5 shows a screenshot of the rule editor and the additional Eclipse configuration added by this tool.

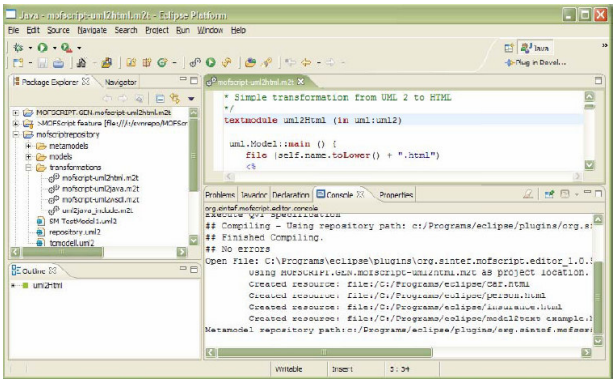

Figure 5. MOFScript Plugin

2.3 Microsoft DSL Tools vs. Eclipse Modeling Plug-ins

From an objective point of view, some clear differences can be identified from these two technologies. In this section we provide a description of these differences, which are summarized in the Table 1.

	DSL Tools	Eclipse
Metamodelling	Proprietary Notation	EMF (eCore)
Repository	XML file	XML, XMI
Graphical Notation	Direct Manipulation of XML files	GMF
Model to Model	Lack of explicit technique	ATL, MTF, Viatra2, etc.
Model to Text	Proprietary Template Language	MOFScript, FreeMarker, Velocity, JET, etc.

Table 1. DSL Tools vs. Eclipse

Regarding **metamodelling**, Microsoft DSL Tools provide a proprietary notation and a graphical environment for specifying the language metamodel. Eclipse, in the Eclipse Modeling Framework (EMF), provides a complete metamodeling and model management environment, which uses Ecore as language for metamodel specification.
It is important to note that for the serialization of models (**model repository**) the Microsoft DSL Tools provide a XML proprietary format, whereas Eclipse/EMF supports the XMI standard or any XML-schema format defined by the users.
On the **graphical editor** definition, the Microsoft DSL Tools provided (in the 2006 February

version) a primitive mechanism consisting on the direct manipulation of XML files. Fortunately, the June version included a graphical mechanism for achieving this task. In the case of Eclipse/GMF, the environment for the definition of graphical editors is more complete than in the Microsoft DSL Tools, since it provides wizards, cheatsheets and other utilities that are integrated within GMF.

The Microsoft DSL Tools lack of an explicit technique for supporting **model-to-model** transformations. One possible trick to implement this kind of transformations is to use the templates mechanism for generating the XML files that represents the target models. Eclipse (still[3]) does not provide an implementation of the OMG standard (QVT), but many plug-ins exists for model-to-model transformation which manipulate Ecore models (ATL, MTF, Viatra2, etc.).

Finally, for the code generation (the **model-to-text** transformation) the DSL Tools provide a primitive template language that is integrated in the environment. This language enables the injection of C# or VB code. In the Eclipse world, all the Java-based templates languages (like FreeMarker, Velocity, JET,…) can be used, since the models can be manipulated as Java classes (this is a feature provided by EMF). From a pure model-to-text approach (in this context, ecore-to-text) we have introduced above the MOFScript plug-in.

3. DSL TOOLS VS ECLIPSE MODELING PLUGINS. EMPIRICAL EVALUATION

In this section we introduce the experiment that has been carried out in order to qualitatively compare the Microsoft DSL Tools and the Eclipse Modeling Plug-ins. First, we present how the experiment was designed; next we describe the questions that were formulated to the subjects and finally the results of the questionnaires are analyzed.

3.1. Experiment Design

In our study we have adopted two well known techniques for user research: focus groups [9] and survey. Focus groups are collective interviews and discussions involving a small set of target representatives per session (lecturers and researchers in this case, and usually from six to twelve persons). In our case, focus groups allow us to collect which research questions (qualitative information) should be formulated to the students in the survey. The survey questions focused on the qualitative characteristics of DSL Tools and Eclipse Modeling Plug-ins that emerged from the focus group sessions.

The experiment has been developed by 48 fifth year undergraduate students of computer science engineering. The students were between 22 and 24 years old and had similar backgrounds. The students were divided into 2 groups for developing a DSL including code generation using Microsoft DSL Tools[4] (20 students) and Eclipse Modeling Plug-ins[5] (28

[3] http://www.eclipse.org/proposals/m2m/

[4] February 2006 Version (currently there is a new version that was published in June 2006)

[5] April 2006 Version (currently there is a new version that was published in June 2006)

students).
The DSLs were proposed by the students under the advisor of the supervisors. Some of the projects that were developed during the course were:

- PervML Modeler and a Code Generator (Supporting MDD of Pervasive Systems).
- A J2EE Code Generator (Building a DSL and Code Generation).
- Definition of a Language for Specifying Project Plans and Code Generation of a tracking application.
- Definition of a Language for Modeling Agents and Code Generation in Jade.
- Generating Code in Hibernate from Class Diagrams. Making Objects Persistent.
- Specification and Automatic Generation of Unit Tests (JUnit) from the UML Class Diagram.
- Code Generation from BPMN Models using Toghether 2006 QVT.
- From the Class Diagram to the Relational Model. Generating SQL code.
- Definition of a Language for specifying Aspect Oriented Software Architectures.
- Definition of a State Transition Diagram Modeler and Code Generation in C#.
- A DSL for specifying and generating Posters.
- Definition of Language for modeling Surveys and HTML Code Generation.

The students developed their projects and make a public presentation and a demo of their work to the other students. Each group was instructed only in the tools that were necessary to develop their projects. The students who were using the Microsoft DSL Tools does not know much information about the features of Eclipse and its plug-ins and vice versa.

3.2. Research Questions

One focus group was created (composed by 2 lecturers and 5 researchers) to collect and formulate 15 research questions to the students. The questions were structured on 5 categories. Following, a brief description of each category is introduced with the main topics that were covered. In the **Appendix I**, detailed information of the formulated questions can be found.

1. **Metamodelling**: Questions in this section capture the opinion of the students about the *understandability* and *expressivity* of the metamodelling technique and the *usability* of the mechanisms provided for the metamodel specification.
2. **Graphical Editor**: Questions in this section can be divided in two groups. On the one hand, questions regarding the capabilities for defining the graphical editor were asking about *usability* and *extensibility*. On the other hand, questions regarding the generated graphical editor, were asking about *perceived quality*, *perceived completeness* and a *comparison with the graphical editors that were generated with the other tools*.
3. **Code Generator**: Questions in this section asked about the perceived *complexity* of achieving the implementation of the transformations and their *preferred technique*: a common programming language or a template language (or, in

general a specialized model-to-text technique).

4. **Satisfaction**: Questions in this section capture the feeling of the students about the *utility* of the tools. This characteristic was requested directly, but it also was asked about a future planned *industrial application* and about the possibility of changing to the alternative tools in the case of starting again the project (*fidelity*).
5. **General Questions**: Questions in this section include two issues that we thought that could be very relevant but that do not fit well in any other category. First we asked about the availability of *documentation*. Second, we requested the students to selected one of the technologies as more *mature* to the other, even when we knew that both of them were pre 1.0 releases.

For every question a closed set of options was provided (for instance "*easy*", "*medium*", "*difficult*") in order to facilitate to analysis of the data. There was not a fixed time for answering the questions.

3.3. Analysis of the Results

The raw data obtained after the questionnaires were processed is shown in the table that is included in the **Appendix II**. For each question, the table shows the percentage of students that selected each answer. The data is presented according to the technology that the students used for developing their project, but also the global percentage is included. We have analysed this data in order to extract some knowledge about the questions that were the focus of this comparison. Following we present our analysis of the results.

About Metamodelling. eCore and EMF are easier to understand than the proprietary notation provided by the DSL Tools, as it is shown in Figure 6. However, in the end, both could be understood without any problem, since success/failure ratio in each group was similar. eCore is expressive enough to build language models, although it is a subset of the MOF standard. DSL Tools notation is a little bit more difficult to understand than eCore. The EMF metamodel designer is more usable than the one provided by the DSL Tools.

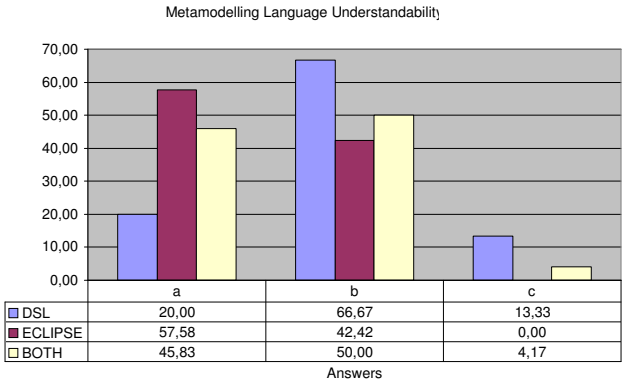

Figure 6. Results about Metamodelling Language Understandability

About the Graphical Editor. GMF and DSL Tools Graphical Designer are difficult to use, however only in the case of GMF some students say that it is easy to use (12%). The generated graphical modellers seem incomplete in both cases. GMF and DSL Tools Designer need to be improved in order to provide more mechanisms for defining and producing professional Graphical Designers. However, GMF/Eclipse reaches a high degree of acceptability (63% say that is complete enough) compared to DSL Tools. Most of the time, an extra work is needed to build professional Editors. Both DSL Tools and Eclipse Users prefer the Editors that have been generated using GMF, as it is shown in Figure 7.

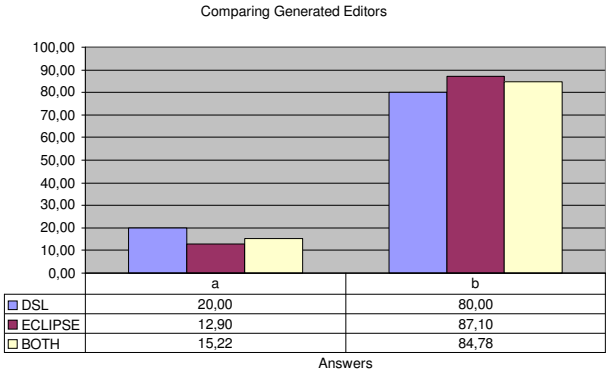

Figure 7. Results about the preferred generated graphical editors

About Code Generation: The task of defining/implementing a Code Generator has been rated as a medium degree of difficulty. It is important to note that only DSL Tools users (26%) consider this task as a difficult one. Eclipse users prefer *MOFScript* to build the code generator. However, DSL Tools users prefer a different programming language to the one (the template language) that is provided by the Tool.

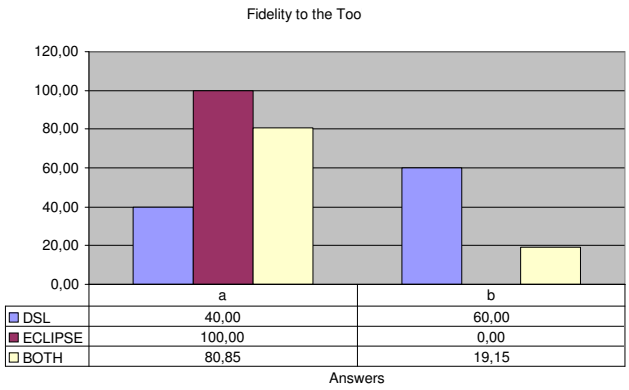

Figure 8. Results about the fidelity to the used toolset

About User Satisfaction: Eclipse and DSL Tools users think that both tools are very useful. Most students are thinking about using these tools in their professional careers. Eclipse users are 100% faithful to this environment; however 60% of the DSL Tools users would migrate to Eclipse, as it is shown in Figure 8.
About the General Issues. Both tools should publish more documentation (they are on the way). DSL Tools and Eclipse users think that Eclipse Modeling Plug-ins are more mature and robust (17% DSL Tools vs. 83% Eclipse Modeling Plug-ins).

4. CONCLUSIONS

In this paper we have presented an empirical comparison of Microsoft DSL Tools and Eclipse Modeling Plug-ins. In order to evaluate them, an experiment has been developed with 48 last year undergraduate students of computer science engineering. The students were divided into 2 groups for developing a DSL (including code generation) where each group was using a different tool. The students answered a survey about their experiences with the tools. The paper has presented how the experiment was designed, the questions that were formulated and some conclusions about the current state of the tools that are extracted by analyzing the students' answers.

In this study the Eclipse Modeling Plug-ins have been better accepted by the students than DSL Tools. We can say that in some aspects Eclipse (and its related tools) is one step forward than DSL Tools. EMF was published in 2003 and it is being used by the community during 3 years. In this way EMF is more simple, robust and stable than the Metamodel Editor provided by DSL Tools. Both tools provide enough support and expressivity to build complete MDSD tools with code generation. We think that the Eclipse Modeling Plug-ins are well conceived because Eclipse provides an open environment where many tools (external plug-ins which some of them implement OMG standards) can be easily integrated. As a result of this integration, a customized CASE tool can be easily produced with many DSDM features like graphical modelling, model validation and automatic software production. The DSL Tools are closed and vendor dependent (Microsoft) without any support to OMG standards. However, they are well integrated in Visual Studio that is one of the most used and well known development environments. This fact could give to the DSL Tools a competitive advantage for the final adoption by the software developer's community.

We are aware that current versions of the tools are not final releases and that this comparison should be repeated with commercial or stable versions. We are planning to improve and extend this study in order to develop another experiment in the next academic year using more stable versions.

REFERENCES

[1] Object Management Group. *Model Driven Architecture Guide*, 2003.
[2] J. Greenfield, K. Short, S., and S. Kent. *Software Factories*. Wiley Publising Inc., 2004.
[3] http://www.interactive-objects.com/products/arcstyler.

[4] http://www.compuware.com/products/optimalj/
[5] http://www.borland.com/us/products/together/index.html
[6] http://www.andromda.org/
[7] http://gentleware.com/index.php
[8] http://www.care-t.com/
[9] R. A Krueger. *Focus Groups : Practical Guide for Applied Research*. 1994. Sage Pubns.

APPENDIX I. RESEARCH QUESTIONS

- **Q1:** ***Documentation Availability***. The formulated question was: *The available documentation in Internet and other sources has been…,* and the possible answers were: *(a) Good (b) Enough (c) Poor.*
- **Q2:** ***Metamodeling Language Understandability***. The formulated question was: *The language or technique used to define the metamodel has been…*(we look for the understandability of the language), and the possible answers were: *(a) Easy (b) Acceptable (c) Difficult.*
- **Q3:** ***Metamodeling Language Expresivity***. The formulated question was: *The language or technique used to define the metamodel has been…*(we look for the expresiveness of the language), and the possible answers were *(a) Enough (b) Not Enough.*
- **Q4:** ***Language (Metamodel) Designer Usability***. The formulated question was: *The tools for defining the metamodel have been…*(we look for the usability of the tools), and the possible answers were *(a) Easy (b) Acceptable (c) Difficult.*
- **Q5:** ***Graphical Designer Usability***. The formulated question was: *The tools for defining the graphical designer have been…*(we look for the usability of the tools), and the possible answers were *(a) Easy (b) Acceptable (c) Difficult.*
- **Q6:** ***Quality of the Resulting Graphical Modeler***. The formulated question was: *The resulting graphical designer was…*, and the possible answers were *(a) Better than expected (b) As Expected (c) I miss some details (d) Poor.*
- **Q7:** ***Graphical Designer Completeness***. The formulated question was: *Were the tool and the abstractions used for specifying the graphical designer enough to completely build the visual editor?*, and the possible answers were *(a) Complete (b) Acceptable (c) Not Enough.*
- **Q8:** ***Extensibility of the Graphical Designer***. The formulated question was: *If you need to add new figures or graphical features, How can be achieved?*, and the possible answers were *(a) Easy (b) Need Some Extra Work (c) Impossible*.
- **Q9:** ***Comparing Generated Editors***. ***DSL vs. Eclipse***. The formulated question was: *Which editor do you think that it is of more quality?* , and the possible answers were *(a) DSL (b) GMF* .
- **Q10:** ***Maturity and Robustness. DSL vs. Eclipse***. The formulated question was: *Which tool do you think that it is more mature and robust?*, and the possible answers were *(a) DSL Tools (b) Eclipse.*
- **Q11:** ***Complexity in Defining the Code Generator***. The formulated question was: *The*

definition of transformations and the implementation of the code generator have been… (we look for difficulty), and the possible answers were *(a) Easy (b) Medium (c) Difficult.*

- **Q12:** ***Implementing the Code Generator. Programming Language vs. Template Engine***. The formulated question was: *Which tool do you prefer to implement the code generator?*, and the possible answers were *(a) Template Language (b) Any Other Programming Language.*
- **Q13:** ***Utility of the Employed Tools***. The formulated question was: *Do you think that the tools that have been applied to implement the DSL and the code generator are useful?* , and the possible answers were *(a) Yes (b) Not sure (c) Not.*
- **Q14:** ***Industrial Application***. The formulated question was: *Do you plan to use in the near future these kind of tools for industrial software development?,* and the possible answers were *(a) Yes (b) No.*
- **Q15:** ***Fidelity to the Tool***. The formulated question was: *If you should start a new project now, Would you use the same tools?* , and the possible answers were *(a) Same Tool (b) The Other.*

APPENDIX II. GATHERED DATA.

	Using Microsoft DSL Tools				Using Eclipse Modeling Plug-ins				Total			
	a)	b)	c)	d)	a)	b)	c)	d)	a)	b)	c)	d)
Q1	6,67	40	53,33		9,09	36,36	54,50		8,33	37,50	54,17	
Q2	20	66,67	13,33		57,58	42,42	0		45,83	50	4,17	
Q3	40	60			90,63	9,38			74,47	25,53		
Q4	0	80	20		45,45	48,48	6,06		31,25	58,23	10,42	
Q5	0	53,33	46,47		12,50	50	37,50		8,50	51,56	40,43	
Q6	0	13,33	73,33	13,33	6,25	25	62	6,25	4,26	21,28	65,96	8,61
Q7	7,14	42,86	50		21,21	42,42	30,30		16,67	41,67	35,42	
Q8	0	86,67	13,33		25	50	25		16,28	62,79	20,93	
Q9	20	80			12,90	87,10			15,22	84,78		
Q10	13,33	86,67			18,75	81,25			17,02	82,98		
Q11	13,33	60	26,67		30,30	69,70	0		25	66,70	8,33	
Q12	13,33	86,67			90,91	9,09			66,67	33,33		
Q13	80	20	0		72,73	15,15	12,12		75	16,67	8,33	
Q14	76,92	23,08			81,82	18,18			80,43	19,57		
Q15	40	60			100	0			80,85	19,15		