EVALUATION ON HOW SUITABLE OPEN SOURCE TOOLS ARE FOR MODEL TO MODEL TRANSFORMATION
An industrial point of view

Bachelor Degree Project in Computer Science
Basic level 15 credits
Spring term 2013

András Márki

Supervisor: Birgitta Lindström
Examinator: Jonas Mellin
Summary

Model-Driven Development can improve the development process, but it needs tools for model transformation. For industrial companies, the most important aspect is that the transformation tools should scale well, so that they can be used with huge models. There are some open-source model transformation tools on the market, and this report aims to investigate the scalability of open source tools for model transformation. For the investigation, Eclipse Modeling Framework is used.

This report identifies four open-source model transformation tools (ATL, QVT Operational, QVT Declarative, SmartQVT) and identifies the variables needed for a tool to be evaluated within the bounds of an experiment.

The only tool which could be benchmarked was ATL, which scaled linearly in both terms of transformation time and memory consumption.

Keywords: Model-Driven Development, Scalability, ATL, QVT Operational, QVT Declarative, Benchmarking
Table of contents

1 Introduction................................................................................................................................. 1
2 Background................................................................................................................................ 2
2.1 Models and Model-Driven Development ................................................................................. 2
2.2 Model Driven Architecture ...................................................................................................... 3
2.3 MDA and software engineering ................................................................................................. 5
2.4 Model transformation ............................................................................................................... 6
2.5 Effects and difficulties with MDD ............................................................................................ 6
3 Problem ...................................................................................................................................... 8
4 Method and Approach................................................................................................................ 10
4.1 Objective 1: The OSS market situation regarding model transformation tools...10
4.2 Objective 2: Factors that impact scalability......................................................................... 11
4.3 Objective 3: Tool evaluation .................................................................................................. 11
5 Related work .............................................................................................................................. 13
6 Results....................................................................................................................................... 14
6.1 Objective 1: The OSS market situation regarding model transformation tools...14
6.1.1 The analysis on quality of open-source software ................................................................. 14
6.1.2 Evaluation of the candidates ............................................................................................... 16
6.2 Objective 2: Factors that impact scalability......................................................................... 18
6.3 Objective 3: Tool evaluation .................................................................................................. 20
6.3.1 Single class single attribute meta-model transformation using ATL and default Eclipse settings ........................................................................................................................................... 21
6.3.2 Single class single attribute meta-model transformation using ATL and modified Eclipse settings with 1024MB maximum heap size................................................................. 23
6.3.3 Multiple class single rule (MCSR) meta-model transformation using ATL and modified Eclipse settings with 1280MB maximum heap size ................................................................. 28
6.3.4 Multiple class multiple rule (MCMM) meta-model transformation using ATL and modified Eclipse settings with 1280MB maximum heap size ......................................................... 30
7 Discussion.................................................................................................................................. 33
7.1 Objective 1: The OSS market situation regarding model transformation tools...33
7.2 Objective 2: Factors that impact scalability......................................................................... 33
7.3 Objective 3: Tool evaluation .................................................................................................. 33
8 Conclusion................................................................................................................................. 38
8.1 Contribution............................................................................................................................... 38
8.2 Limitations of the method and approach ............................................................................... 38
8.3 Future work.............................................................................................................................. 39
8.4 Ethical factors.......................................................................................................................... 40
1 Introduction

Model-Driven Development (MDD) focuses on domain models rather than computing concepts. It can be used to transform models to other models over even to code. MDD has benefits as reduced time to market and better product portability. However, introducing MDD or changing the development tool means a possible change of development process for the companies willing to adopt it, which can be expensive and risky. Therefore, choosing the right tool is important.

There are proprietary tools used in the industry, but some of them are expensive, some of them are domain specific, while others lack desired features. Sometimes, the tool is produced by another company, thus adding important features can take time. Using open-source tools can solve this problem, as features can be added, and expenses can be lower, even though developing additional features can have its cost.

For industrial use, the model transformation is a crucial part of MDD. In case the transformation does not scale, the development process will fail as solutions cannot be generated. Therefore, the aim of this report is to investigate the scalability of open-source tools for model to model transformation. To achieve this aim, three objectives should be accomplished: The current open-source situation regarding model transformation tools should be investigated, the factors which affect transformation scalability should be determined and an evaluation on the tools should be made.

The transformation scalability can be shown with benchmarking, thus the chosen method for this evaluation is a controlled experiment. The tools are identified by a survey on the relevant open-source communities. A set of selection criteria is defined to choose tools for evaluation. These criteria is defined by literature analysis. The relevant factors for the benchmark are identified based on the literature. Afterwards, the scaling of the identified tool(s) fulfilling the baseline criteria are evaluated on how well they scale with regards to the selection factors.

The identified open-source tools for model transformation are ATLAS Transformation Language (ATL), Query/View/Transformations Operational (QVTo), Query/View/Transformations Declarative (QVTd) and SmartQVT. The selection criteria for evaluation are active development, the ability to install the tool as well as sufficient documentation to create single transformation with the tools. SmartQVT is not actively developed anymore, QVTd could not be downloaded and QVTo lacks sufficient documentation. ATL fulfils every criteria and can be benchmarked. ATL shows linear development for both time consumption and memory requirement.
2 Background

2.1 Models and Model-Driven Development

As defined by Mellor et al. (2003), "A model is a coherent set of formal elements describing something (for example, a system, bank, phone, or train) built for some purpose that is amenable to a particular form of analysis." Hence, a model describes something and can be analysed. The model can also be regarded as an abstraction of the described thing, as well as different models can be used to describe different aspects. As Maher (1989) states, the higher level of abstraction represent aggregation of components into systems, which "provides a basis for synthesis by hierarchical decomposition". For example, a phone can have different models focusing on the network communication or the interface. As stated by Selic (2003), "Models help us understand a complex problem and its potential solutions through abstraction." In case designing the user interface of a phone, the focus lies mainly on the human interaction while the underlying hardware is less important. Models can thus help to divide problems to more manageable levels and give possibility to focus only on what is necessary from a given perspective.

According to Selic (2003), there are five characteristics an engineering model should fulfil: Abstraction, understandability, accuracy, predictiveness and inexpensiveness. Abstraction is important, as not every detail is needed in every situation, and hiding abstraction can be necessary with complex systems. Understandability means that the model should be intuitive and relatively easy to understand. A picture can say more than a thousand words, and so can a good model too. It should be relatively easy to get an overview of the modelled part of the system by looking at its model. Accuracy means that the model represents the complex problem on the given abstraction level as it is in the real life. Accuracy means also that it should be decidable if a model is a good representation, and this attribute makes it possible to validate the system early on in the development.

Predictiveness can be used for verification. It should also be able early on to verify the system if it works correctly using the modelled systems properties. There are model verifiers as well as in case of executable models, the model can be run and the results can be analysed. A model should be inexpensive, because in case it would be more expensive to build the real system than the model.

With the use of Model-Driven Development, the five characteristics of Selic (2003) are fulfilled. The appropriate abstraction level can be defined and models can be made understandable resulting in a system easier to be analyzed. Accuracy can be validated early on by looking at the model and compare it to the modelled problem. Predictiveness can be done by model checking and executing the code. Well done and appropriate models can be re-used, with systematic reuse providing better development productivity (France, Ghosh & Turk, 2001). Source code has the lowest abstraction and generic models can result in better reuse process of artefacts (Prieto-Díaz, 1990).

Mellor et al. (2003) describes Model-Driven Development (MDD) as the following: "Model-driven development is simply the notion that we can construct a model of a system that we can then transform into the real thing." In MDD, a model of the system is constructed and then it is translated to an implementation. The benefit is that the developers can work closer to the original problem domain and the solution can be created from the representation of
that domain. Another important aspect is that the requirements are expressed similarly they are described within the domain. In case the phone, it should be possible to dial. The benefit with MDD is that verification can be easier and it is easier to detect in case the model does not fulfils every requirement.

2.2 Model Driven Architecture

Model-driven architecture (MDA) is a "set of technologies and techniques that enable model-driven development" (Mellor et al., 2004, pp. 139). With other words, "Model-driven engineering technologies offer a promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively." (Schmidt, 2006). MDA focuses on what is necessary to have so that MDD can be used for development. Model driven architecture can be divided to four steps as shown in figure 1 below. MDA standards defined by the Object Management Group (OMG) are related to the UML modelling standard. The scope of this document is thus limited to UML related solutions, as UML is the most known and used standard. UML has also extension like the Modelling and Analysis of Real-Time and Embedded Systems (MARTE) for real-time and embedded systems and the System Modelling Language (SysML) for system engineering (Gomez et al., 2012).

![Figure 1 Model driven architecture visualized (Arlow, 2009 p.8)](image)

The first step is the creation of the **Computer Independent Model** (CIM), which has the highest abstraction level. "System requirements are specified in the Computation Independent Model" (Asadi, 2012). As stated by Gasevic et al. (2006, pp. 111) "In software engineering, it is well-known as a domain model specified by the domain experts." Also, with the CIM, the focus lies on how the soon to be solved problem can be described, without any focus on how it could be solved computationally.

To give an example, an application with the purpose of drawing will be used in this chapter. The developers should know what the application should be capable of to fulfil its purpose, which means that the users should be able to draw with it. These activities could for example creating objects like lines and point, adjusting colours, and deleting objects. These activities are also the requirements for the drawing program, and these requirements should thus be part of the CIM.

CIM is used to create the Platform Independent Model (PIM). "The Platform Independent model is the model that describes the system design independent of the implementation platform" (Asadi, 2012). PIM can also described as "a computation-dependent model, but it
does not consider the characteristics of specific computer platforms" (Gasevic et al., 2006, pp. 111). Creating the PIM is the second step in MDD. In this step, the functionality is described by the model so that it is still independent from the actual platform.

The PIM can be also divided into static and dynamic structure. The static structure describes the entities and their relations to each other while the dynamic structure describes their behaviour. The static structure can be represented for example by a class diagram while state-machines can be used to describe the dynamic behaviour. Through instancing and marking, platform specific information can be added to the PIM resulting in a Platform Specific Model. In the process, the "validity of PIM is checked by "model checker" firstly, that make sure the PIM's grammatical and semantic is correct and no contradiction or ambiguity" (Yong, 2010).

To continue with the drawing example, the platform independent model is a UML diagram which only includes the parts which are the same for all the intended platforms. In case the application is developed on numerous platforms, for example Android, IOS, UNIX and Windows, the PIM does not hold any information which is not the same for all of them.

"The Platform Specific Model (PSM), on the other hand, describes the system design in the form of a platform-dependent model" (Asadi, 2012). PSM has all the information which is needed for the solution on the platform to be compiled, however it is still OS independent. It should be noted that the process can be reversed; a PSM can be mapped back to PIM by removing platform specific information thus resulting in a higher abstraction level. Both the static and dynamic structures should be translated to get a correct PSM, as both the structure and the functionality are needed for the PSM model to work.

In case of the drawing example, there will be one PSM for Android, one for IOS, one for Unix and one for Windows as the solutions need different additional information to be translated to code. The static and dynamic structure is created from the independent model by adding platform specific information for the executable code: The static structure will be the classes with their interfaces in this case, while the dynamic structure describes the behaviour, also what the classes do with their functions and variables.

Another example would be to develop a model for a radar system, then use the same model as a base for a solution for a handheld radar, a vehicle mounted variant and the simulator for the vehicle. Even though the platform specific model and the actual code could be significantly different, the main functionality of the system described by the CIM (it need to identify specific objects) and the platform independent model (how the identification can be solved on various computing devices) would be the same.

From the PSM, the actual code can be generated. The used language can also decide based on the needs, for example C, C++ C#, Ada, Pascal and Java can be generated among others. The benefit is here that the most suitable programming language can be generated for solving the specific problem, or even code for more languages could be generated and compared to each other in terms of how well they perform solving the actual problem and then the most suitable could be chosen for the actual solution. This may add some overhead (as there can be differences between the models needed for the different languages), however the costs should still be smaller than implementing the same solution by hand in more than one language.
An important feature of MDD is that CIM and PIM models are portable. In case the solution should be ported to a new platform, ideally, a new mapping is needed to be created from the PIM to the new platforms PSM, but the PIM can be re-used, as the necessary functionality is described here. As the PIM stays the same, as long as the changes are on the PIM level, only the platform independent model should be changed, and all the other PSMs can be regenerated. This means good maintainability and portability, which are important factors in terms of software quality and keeping development costs as low as possible.

### 2.3 MDA and software engineering

The main difference between third-generation programming languages (like for example C, C++, C# and Java) and Model-driven Development is the fact that MDD focuses on transforming models to code or other models by tools. As described by Sendall & Kozaczynski (2003), "Model-driven approaches to systems development move the focus from third-generation programming language (3GL) code to models." The main benefit using model driven development is that the developer "describes subject matters at a higher level of abstraction than she would in a programming language" (Mellor, 2002). Actual implementation details specifically for various systems could be added later on, thus a single solution can be used for several different systems. In case of the drawing tool named before in this part of the document, if the developers would add Windows Mobile 8.0 support, it should be achievable by creating a PSM for the purpose, as the underlying PIM would be suitable for the purpose.

xtUML ensures that models will in fact be in the centre of the development. The use of components is also ensured by the use of models. The control of changes and control of various versions is ensured by the use of the same PIM for various PSMs. In case something is updated for a system which is the same for all the systems, the other PSM should be updated also, helping with portability. It can be concluded, that MDD fulfils all the best practices defined by RUP.

"The objective of model-driven development is to increase productivity and reduce time-to-market by enabling development at a higher level of abstraction and by using concepts closer to the problem domain at hand, rather than the ones offered by programming languages." (Sendall & Kozaczynski, 2003). Also, higher abstraction level means that the developers can focus on the information needed for the actual abstraction level, and additional details can be added when they are needed, which can help productivity. The developers can focus on modelling the problem instead of focusing on how they could model the problem within the limits of the programming language.

The detail on the models as well as the abstraction level can be adjusted accordingly: Details can be added or removed as well as models can also be translated to other models. Lower level solutions can also be modelled. It is even achievable to create a model of the behaviour of wires and gates connecting a logical circuit representing a special architecture running a specific machine code. The machine code is mostly created from platform specific (but still operative system independent) code, which can be created from models. To sum it up, all these steps can be modelled, by adding the necessary information to the models, solutions for each of these abstraction levels can be created.
2.4 Model transformation

As models can be translated to both other models and to code, two approaches can be distinguished when speaking about translation. Both approaches can be implemented by various methods. The first one is the model-to-model approach: "In the model-to-model category, we distinguish among direct-manipulation approaches, relational approaches, graph-transformation-based approaches, structure-driven approaches, and hybrid approaches." (Czarknecki & Helsen, 2003). The second approach is the model-to-code approach: "In the model-to-code category, we distinguish between visitor-based and template-based approaches." (Czarnecki & Helsen, 2003). The translation itself can be done manually or with a model compiler which does the translation automated (Mellor, 2004, p. 47).

In the translation process from PSM to PIM the model should be traversed. There can however be differences between how the traversing happens. The biggest difference is the approach. An iterative and a declarative translation can differ in terms of scaling, which can be interesting when dealing with bigger models and there are restrictions on how much time the translation can take.

It should also be noted that the system specific information is only added on the PSM level. From modelling point of view, the difference between a real-time system and a non real-time system are the mark-up properties present in the PSM. UML defines Time events, as well as message and change events which can be important for real-time and embedded systems. (Balandyte & Nemuraite, 2006). According to Völter & Stahl (2006), a the same model can contain the hardware and software parts of a system, as well as the relations between them.

2.5 Effects and difficulties with MDD

The effects on the usability of MDD are not well researched, as there could be only one article identified with findings on how useful actually MDD is in development: "The MDD method was regarded as the least compatible with developers' current practices, but the most useful in the long run." (Martínez, 2012). Their findings are however interesting as they show that such development requires developer adaptation to some level but can be useful on the long run, as this information could be important on how developer adaptation would be, even though other research on the subject are missing. The adaptation time for developers translates to higher total adaptation cost, as they need to learn the process as well as in the beginning, they may not be as efficient, resulting in loss of income for the company. On the other hand, a more suitable tool could help reduce the adaptation cost and on the long run, switching to MDD can be cheaper.

There is however an important factor for enabling MDD: An appropriate tool for development is needed to create and translate the models. Finding a such tool, or changing to another tool can be however difficult. Making this process easier could affect positively the market share of MDD compared to writing the code.

In case of companies already using Model-Driven Development, or even using other kind of development processes there are additional costs which should be taken into consideration when changing the process model. In case the company has legacy code too, "it is believed that that full model-driven development support may improve the efficiency of development" (MacDonald, Russel & Atchison, 2005). In case the company is not newly
founded, there should be software which is already bought by the company. Some of such software may not be used afterwards, resulting in sunk costs. Even though higher sunk cost increases the willingness to continue a project (Keil et al., 2002), changing the process model under a development project should be very well motivated as it is a risk. Changing the development process means that the developers should become familiar with the new tools. This takes time and effort, and under the learning time, productivity can decrease as the developers are less familiar with the new tools, as with the old ones. In case the productivity increases on the long run, the profit can be greater than the expenses, which can motivate the change of tools. On the other hand, the companies should be sure that changing the tool will be profitable, otherwise, they can generate loss in the process. Choosing the right tool is therefore crucial.
3 Problem

The overall aim in this report is to investigate the scalability of open source tools for model to model transformation.

Model-Driven Development requires proper tool support for model manipulation and model translation. Proprietary solutions may lack features while domain specific solutions may introduce too many restrictions. Using open source addresses these problems, since open-source tools have known and modifiable code, they can be changed to suit the needs of the organization better.

It is important to choose the right tool. The introduction of Model-Driven Development also means a change in the process and change of development software. Process changes take time and are expensive, while replacing the tools means lower developer productivity under the learning period and sunk costs. On the long run, switching to MDD can be beneficial, but it can only be beneficial in case the process works, and the process itself depends on the tools. Since the transition is expensive, it is important to invest wisely, and choose the right tool.

In case of open source solutions, thus the tools can be adapted to fit into the process or the environment. Software aspects can also be changed, for example the user interface can be changed for better usability, or the company can chose to improve the documentation of the tool providing better maintainability. Hence, there are many arguments for open source. The major question is whether there are open source tools that are good enough for adoption by the industry.

One of the main issues in the industry is if a tool scale, or not. There are other quality aspects too: ISO/IEC9126 defines multiple aspects of quality for software as functionality, reliability, usability, maintainability, portability and efficiency. Functionality can be altered or added. Usability and maintainability can be improved. Portability can be addressed with changing the environment for the tool to execute.

Scalability is on the other hand different. As stated by Selic (2003), "MDD is intended for - and most beneficial in- large-scale industrial applications". A solution can be highly functional, reliable, usable, maintainable and portable, if it doesn't scale, it won't be suitable for industrial use. In case the solution scales, the other issues can be solved one way or another. It can be concluded that for industrial needs, the most (and only) important aspect is scalability for MDD, but even generally.

The part where scalability comes into picture with Model-Driven Development is the part where the model transformation takes place. The model transformation should scale, and therefore, an open source model transformation tool should also scale to be suitable for industrial use. In case the transformation tool would not scale, the tool and therefore the process using it would fail, resulting in a loss of capital. Hence, unless there are open source alternatives that scale, proprietary solutions might be the only option.

This is the reason why the aim of this report is important. The aim can be divided into three objectives:

1. Describe the current situation regarding open source model transformation tools
2. Determine factors that impact scalability

3. Perform evaluation on the tools based on the identified factors

Giving an answer on the three objectives should identify which open source tools are on the market suitable for model transformation, how they can be compared and then show a comparison on how well they fare. This will answer the overall aim.

The results of this thesis projects should be beneficial to industrial companies, as it expected to help them choosing a model transformation tool which scales to their needs. The companies will have information about the tools and criteria to test their scaling, may resulting in lower adaptation costs for both companies interested in MDD and for companies already using proprietary tools. In addition, for companies using proprietary tools the results show what to except from open source tools in terms of scalability, which can help them to decide if it is worth to switch to an open source solution.
4 Method and Approach

The aim for this report is to investigate the scalability of open source tools for model transformation. Therefore, the method and the approach should focus on how scalability could be benchmarked, also how an experiment could be constructed so that the performance of the various tools can be evaluated. For this experiment three factors should be identified: First, which tools can be evaluated. Second, it should be known which variables are interesting to be changed so that the input transformed by the tools would give relevant results. Third, it should be determined which factors should be looked at after the transformation is done, which ones are relevant.

As the aim is to determine the scalability of the tools, other methods would be less effective. Performing a case study on an actual company and comparing the scalability of the identified tools with the tool currently used by the company would be interesting; however it would be too time consuming for a bachelor degree project. Also, it could show how well the open source tools would be suitable for the given company, but this result would be may less relevant for other companies depending on how the tool would be used.

4.1 Objective 1: The OSS market situation regarding model transformation tools

The first objective is to describe the current situation on the market as well as to identify which OSS tools for model transformation are available. There are two methods suitable for the aim: Literature analysis and survey. The literature analysis can identify some of the tools, but performing a survey on the existing open source communities could also yield results on both understanding the market situation as well as finding possible OSS tools. Afterwards, the quality of the identified tools is evaluated to ensure that they can all be benchmarked. For this, a literature analysis is done to determine the quality factors necessary to discard the unsuitable tools.

In case of the literature study, the first step is to identify the relevant databases, then performing search on them with defined search expressions and collect the results. The identified documents should be saved and references followed up. In case only previously found articles or tools can be found, the literature study can be considered finished. As the focus lies on actual tools, white papers can also be considered as relevant sources in this case.

In case of the survey, the first step is to identify the relevant open source communities, and then search information there about the OSS tools. The survey can be considered finished when no new community or tool can be found. In case there are numerous tools, the most suitable will be chosen here for further evaluation.

It is important that the controlled variables on the experiment i.e. the models remain the same during the experiment. As the aim is to evaluate scalability on the tools, it is important that the same models and meta-models are used for the model transformation for every test case on every model. Thus, a framework is needed so that the other experiment variables can be controlled better. The selected framework is the Eclipse Modeling Framework (EMF), as it is open source and can ensure that the same xmi and.ecore models, also models and meta-models can be used with the different tools. With other words, it is "possible to exchange models between the many tools supporting EMF, minimising the risk for lock-in and
enabling integration of tools to support company development practices with effective tool chains." (Gamalielsson, Lundell & Mattson, 2011) This is however a limitation for this report, as only tools compatible with the EMF are evaluated.

Both survey and literature analysis is done to fulfil objective 1. These were performed until no new tools could be identified by the search terms or by tracing references.

It is important that the tools have suitable quality for further evaluation. For this reason, a literature analysis is performed to determine which factors are important to determine the quality of open-source software. This is done by a literature analysis performed on a database of research articles with a given search pattern. The results are first identified by the name of the research articles. If their name suggest that they can provide interesting information, the abstract is evaluated. In case the abstract is relevant to the literature analysis, the article is saved and evaluated further. Its introduction and conclusion is examined, as well as other parts of the document if needed. In case they are still considered relevant, they are marked as such.

Afterwards, the articles referenced by the marked articles are checked, by name and by abstract if the name suggests interesting information. In case the abstract suggests relevant information, the article is saved and evaluated further, eventually resulting in a marked article if it has relevant content to the analysis, and has not been already marked. The same is done with the documents referenced by the marked referenced articles.

The articles citing the already marked articles can also be interesting. These are therefore evaluated in the same way as the others. The survey is considered done when no new articles can be found using this approach.

After performing this evaluation on the quality of open-source software, it is determinable which factors are necessary to ensure that the tool can be evaluated further, as well as identify other relevant factors.

4.2 Objective 2: Factors that impact scalability

The second objective is to describe which factors impact scalability. This objective is divided into three sub-objectives.

The first sub-objective is to determine which aspects are important to transformation scalability in general. The second sub-objective is to determine how they could be measured, also if there is a built in support for performance measurement in a given tool, or it is necessary to find another alternative way to benchmark it. The used method is literature analysis in this case.

4.3 Objective 3: Tool evaluation

The tools can be evaluated with experiment. The method is a systematic benchmarking on the chosen tools by the chosen criteria, to see how well they fare against each other.

The computer used for the experiment stays the same through all tests performed, so that direct comparison can be made between the results. The computer is dedicated for the tests, so that no other processes may interfere with the evaluation. Other settings are fixed under time of the experiments to ensure that only the examined variables are affecting the results.
The models used for the test will either be present as appendix, or be available from the creator of this report, in case of bigger models, as models over several thousand lines are simply too big to be added directly to the report. As the hardware, the software and the models are known, the evaluation could be replicated to ensure the reliability of the results.

Interpretation to the results is also a part of the tool evaluation and is presented as part of the chapter.
5 Related work

The Eclipse wiki has a dedicated site for M2M Benchmarks\(^1\), which has some benchmarks using ATL, QVTo and QVT Relations. However the number of benchmarks is small, results are missing as well as this page is by no means a research article, so the results should be taken with a grain of salt. This page does not focuses on the scaling with the input model size as a variable.

There is a Master Thesis on the topic by Bosems (2011), but the transformation experiment uses first the number of classes as a fixed variant (at 100) and increases the number of attributes they have, while afterwards it uses fixed number of attributes (100) with increased number of classes. In that report, ATL scales best, but it could be interesting to look at the problem with other variables. Bosems displays however no memory measurements in the report. Another Master Thesis by Nguyen (2010) also concerns model transformation, but it contains no measurements on transformation time, nor the memory required for it.

There are papers on metrics can be measured regarding model transformations. Van Amstel, van den Brand and Nguyen (2010) proposes such metrics, but they concentrate on common metrics like size, consistency and dependency. Van Amstel and van den Brand (2010) does "not consider the input and output models" either, even though their goal is to make model transformation quality measurable.

Regarding tool quality in general and not only considering model-driven development, the ISO/IEC 9126 defines an abstract model standard. Thus many researchers introduce models based on the standard, such as Correia, Kanellopoulos & Visser (2009), Correia & Visser (2008), Heitlager, Kuipers & Visser (2007), Johnson et al. (2005), Kuipers, Visser & Vries (2007), Luijten & Visser (2010) and Samoladas et al. (2008). However these address software quality in general, while this study focuses on scalability.

Regarding model-driven development and scalability, there are studies concerning large-scale systems. Foustok (2007) presents one such system, but it focuses on developing a large-scale system in general, and not on the model transformation part. Heijstek & Chaudron (2009) discusses also how a large-scale industrial MDD project look like, but it does not concerns the transformation part either.

\(^1\) http://wiki.eclipse.org/M2M/Benchmarks
6 Results

6.1 Objective 1: The OSS market situation regarding model transformation tools

In this chapter, the possible open source candidates are presented, as well as the needed criteria are defined that are necessary to evaluate them further. The criteria is based on a literature analysis on how quality can be determined on open source software. The possible tool approaches are presented first, afterwards the finding on the quality and lastly in this chapter, the candidates are compared by the quality aspects identified to be relevant to this research.

There are two major approaches within the Eclipse Modelling Framework (EMF): The Atlas Transformation Language (ATL) and Query/View/Transformations. QVT has also different implementations. These approaches are all part of the Eclipse Modelling Framework. The identified tools are the ATLAS Transformation Language, Query/View/Transformations Operational, Query/View/Transformations Declarative and SmartQVT.

6.1.1 The analysis on quality of open-source software

The reason of the literature analysis was to be able to determine if a tool is good enough for further evaluation. Also, metrics are needed to measure the attributes (Fenton, 1994), as well as an appropriate model to compare the tools against the necessary factors. Capra, Francalanci & Merlo (2008) describes that software quality is traditionally based on the quality of the design and the complexity of the software.

There is some controversy in the differences between open- and closed-source software: Perkins (1999) describes open-source low cost, high quality and the development process robust and effective. Fitzgerald (2004) mentions that open-source development addresses issues with code quality, cost and development time scale. Fugetta (2002) on the other hand points out that the difference is smaller between open and closed source: Development process claims on open-source software apply to closed source development too and open-source software is not necessarily better, more reliable or cheaper than closed-source software.

After performing the literature analysis, a couple models constructed to measure software quality were identified. The most important model is the ISO/IEC 9126 standard, with attributes as functionality, reliability, maintainability, usability, efficiency and portability and numerous articles referring to it. The ISO standard is considered too general by some (Behkamal, Kahani & Akbari, 2009), and some others criticise it for not giving definition on how quality aspects should be computed (Bakota et al., 2011; Heitlager, Kuipers & Visser, 2007; Kuipers, Visser & Vries, 2007). However there are multiple models based on the ISO/IEC9126 standard, giving solutions on how the metrics could be calculated. Such models are proposed by Johnson et al. (2005); Kuipers, Visser & Vries (2007); Samoladas et al. (2008); Heitlager, Kuipers & Visser (2007); Correia & Visser (2008); Correia, Kanellopoulos & Visser (2009); Luijten & Visser (2010).

Behkamal, Kahani & Akbari (2009) also lists other models, with Boehm, Star, BBN lacking criteria, McCall overlapping between attributes, FURPS not considering portability and Dromey which is incomprehensive. Taibi, Lavazza & Morasca (2007) defines OpenBQR, with
attributes as quality in terms of absence of defects, availability of maintenance or support, cost of modules or necessary tools and other issues like licenses and the programming language. Lavazza (2007) also describes the possibility of using balanced scorecards over OpenBQR. Capiluppi, Lago & Morisio (2003) defines the characteristics of open-source projects by age, application domain, programming language, code size, number of different type developers, modularity level, documentation, popularity, status, vitality and success of project. Capiluppi, Lago & Morisio also have an alternative variant in an article published in the same year with version number and date of version instead of success of project. Sung, Kim & Rhew (2007) proposes an own model, focusing on functionality, usability, portability and reusability. Johnson et al. (2005) classifies important software metrics as internal and external, with size, complexity and modularity as internal; and effort, productivity and reliability as external. Wong (2006) discusses different views on quality, with transcendental-based, product-based, user-based, manufacturing-based, economic-based; and possible qualities classified as technical, use, aesthetic, symbolic and organizational. The approach of Wong also demonstrates how many different point of views are existing when quality is evaluated. Medrano et al. (2010) defines an evaluation model with five simple parameters: Existence of stable version, installable, learnability, the software's ability to work with a simple example as well as future maintenance. Aberdour (2007) as well as Laplante, Gold & Costello (2007) describes three evaluation models, also Capgemini's Open Source Maturity Model, Navica’s Open Source Maturity Model and the Business Readiness Rating, however they can be classified uninteresting: The website of the first two isn't alive, while the third one contains nearly no information and was not updated under the last couple of years. Petrinja, Nambakam & Sillitti (2009) have also defined OpenSource Maturity Model, with trustworthy elements as goals.

Given the six attributes defined by the ISO/IEC 9126 standard, most discussions concern maintainability. Heitlager, Kuipers & Visser (2007) defines a Maintainability Index to measure it. Bakota et al (2005) describes the importance of maintainability and presents a probabilistic approach to compute it. Bakota et al. (2012) draws the conclusion that maintainability affects costs and describes that it can be estimated. Luijten & Visser (2010) draws a connection between ISO and SIG and concludes that ISO maintainability correlates with SIG volume, duplication and unit complexity, where unit interfacing on the other hand does not correlate.

Most metrics also concern maintainability. Alves (2010) describes that to achieve good code quality, direct measurement is needed. As code is known with open source tools, this is feasible as there are no black boxes (Kumar & Singh, 2012; Laplante, Gold & Costello, 2007; Madanmohan & Rahul, 2004). Gousios & Spinellis (2009) describes however that analysing software can be difficult and expensive. Both Buse and Zimmerman (2012) and Alves, Correia & Visser (2011) names that the interpretation of the gathered data can be difficult to be analyzed. Wong (2006) describes that maintainability is only important for developers, but not for users. Conley and Sproull (2009) describes that increasing modularity however does not reduces the number of bugs.

Measuring maintainability by different tools can be a problem. Bakar & Boughton (2012) describes that there is no standard for measure the various attributes, thus there can be differences between the metrics measured with various tools, based on information gathered with CKJM, JStyle, RSM and JHawk. Lincke, Lundberg & Löwe (2008) draws the same conclusions, with metric tools implementing and interpreting differently. Some of the tools
can also be quite expensive, for example Stamelos et al. (2002) used Logiscope to gather metrics. Because data gathered with different tools aren't directly comparable, as well as gathering the data can be resource intensive, no code metrics data is gathered in conjunction with this research. However, for companies willing to adopt them can be interesting to look at the metrics of the desired tool: In case they are using the quality assessment process and tools they are using with own developments, the code quality of the open-source tool can be directly compared to the company standards, which can be worth the effort to gather it. It can be noted that in case the there exists a plug-in for Eclipse for measuring metrics, in case the company does not have an own yet, which is called Eclipse Metrics (Ajlan, 2009).

Sim, Easterbrook & Holt (2003) describes the following properties important for benchmarking: Accessibility, affordability, clarity, relevance, solvability and portability. Also, the benchmark should be easy to use, it should be within the bounds of the given time frame, it should be clearly specified. The results should be comparable. The benchmark should be feasible with the sample and should be useable as long as the benchmarked products fulfil some basic requirements. Such basic requirements are described by Medrano et al (2010).

The evaluation models is based on the one described by Medrano et al (2010), but with some modifications. Support and an active community is important (Wong, 2006; Lavazza, 2007; Taibi, Lavazza & Morasca 2007; Fitzgerald, 2004; Sung, Kim & Rhew 2007; Meirelles et al., 2010; Capiluppi, Lago & Morisio, 2003), so the first step is to ensure if the tool is currently in development.

The second step is to validate that there is a stable version, which can be installed via the Eclipse Modelling Framework. With closed-source software, version 1.0 is considered the version which is released to the users, but it isn't the case with open-source software (Medrano et al, 2010). Therefore a successful installation is used as a criterion for this attribute.

The third step for the evaluation is that it should be feasible to construct a small, working example using the model transformation tool with given properties. This selection criterion is needed so that the same models and meta-models can be used with the benchmark, thus resulting in comparable results. Only identified model to model transformation tools passing all these three criteria are suitable to be benchmarked later on.

6.1.2 Evaluation of the candidates.
The first step is to identify the possible candidates within the Eclipse Modelling Framework, that can be benchmarked and evaluated.

ATL is a part of ATLAS Model Management Architecture and uses models as "first class entities" (Bézivin et al., 2003). Jouault et al. (2006) defines that the ATL Development Tools (ADT) has the following important features: An ATL Virtual Machine (VM), supporting editing features, building and debugging features. The ATL VM can run on top of a framework, for example the EMF. This can be an interesting feature in case a company want to add it to their own process. The editor features syntax highlighting. ATL gives error messages and ATL projects can be built directly with Eclipse run command in the EMF. Debugging makes it possible to execute the code step by step, or until a point it reaches a
breakpoint. As in May 2013, the current build is 3.3.0 and the next release is expected June 2013, as stated at the Eclipse ATL website.

QVT has more than one implementation in the EMF: QVT Operational (QVTo), QVT Declarative (QVTd) and SmartQVT. QVTo and QVTd are alive: QVT has a build number 3.2.0 as in May 2013 and an update expected within two months, while QVTd has a build number 0.9.0 and is expected to get an update within two months too. The last version from SmartQVT is 0.2.2, which was released 2008-08-07. SmartQVT is therefore discarded from the evaluation, as it is no longer in development based on the information. QVT Declarative is still in incubation phase, which is the phase to establish a functional open-source project. QVTo and ATL have higher maturity level. QVTo had 69 commits under the last three months, QVTd had 26 and ATL had 46.

The second step is to validate the existence of the stable version, as it can't be determined judging alone from the version number, whether an open-source program is stable. This is done by trying to install the plug-in into the Eclipse Modelling Framework: In case the installation is successful, the tool can be regarded stable. ATL and QVTo installs without problem. There is however a shortcoming with QVTd in this regard: This experiment was performed between January 2013 and May 2013, and under most of the time, the Eclipse download page for QVTd was down. It is back online after an update 2013/05/08, but at the time, there was no more time to evaluate the tool because of the time limitations of this project. Thus QVTd is discarded from the list of candidates, as it does not qualify for one of the criterion.

The third criterion is to verify that it is possible to create a simple transformation using the tool, so that it can be benchmarked. Thus, the documentation should be good enough so that it is possible to create a simple transformation using the tool. The most basic features when performing a model-to-model transformation are to define an input model according to the source meta-model, perform the transformation with rules and create the output model according to the target meta-model. Because of time constraints, a 40 hour (also standard Swedish job week) time period was defined as a qualifying criterion. In case the available artefacts, also documentation and tutorials are not enough to perform a working transformation within a week, the tool is dropped from the evaluation as it takes simply too much time to learn it because of the lack of supporting artefacts. This can also be motivated from an industrial point of view: Learning time is a part of the expenses and if information is hard to find about the tool, it will be less appealing.

ATL is documented well enough to qualify itself, but not QVTo. With conjunction to QVTo, there were numerous tutorials and documents identified, but they had flaws. Either it worked but contained too little information, or it did not work at all, or it did not contain all the files needed. The Eclipse page for QVTo has tutorials, but the more interesting ones are

2 http://projects.eclipse.org/projects/modeling.mmt.atl
4 http://projects.eclipse.org/projects/modeling.mmt.qvt
5 http://sourceforge.net/projects/smartqvt/files/smartqvt
6 http://www.eclipse.org/mmt/?project=qvtd
7 http://www.eclipse.org/mmt/downloads/index.php?project=qvtd&showAll=0&showMax=
8 http://blog.requirements.ws/2013/01/qvt-transformations-with-eclipse.html
11 http://www.eclipse.org/mmt/downloads/?project=qvto
not working as they are missing some files or some descriptions are simply too abstract to be executed. There is a book on the topic named "QVT - Operational Mappings"\(^\text{12}\), but it is in German and was considered therefore too difficult for this report to be used. QVTo was also discarded from further evaluation, as without a working transformation, it could not be evaluated.

Hence, only one tool is selected for evaluation in this study. All other identified open-source tools are rejected according to the selection criteria, e.g. due to poor documentation, missing download site or lack of current development.

### 6.2 Objective 2: Factors that impact scalability

As van Amstel et al. (2011) states "Metamodels, models and transformation definitions all influence the performance of model transformations." Van Amstel et al (2011) also states that the complexity and size attribute of the input models is also a factor which affect transformations. Both transformation time and memory requirement are dependent on the size of the input. The size of the input is, in this case, how big the model is, also how much instances the source model has. This is a parameter which should be focused at, at this can answer the question that how big models the tool can translate, and the difference between the various sized models can answer the scalability question. The models itself are connected to the meta-models: Meta-models can have one or more classes with respective attributes. The class structure of the meta-model impacts the model, thus this can also be a variable in the experiment, also how many classes the meta-model has. Between the source and target models there is the code for the transformation, which directs the transformation by mapping rules. Thus, the number of mapping rules is also an interesting variable to look at when researching scalability.

From execution point of view, resource requirements can be divided into two factors: (Wall clock) time requirements and memory requirements (Barney, 2013).

Time requirements describe time consumption for a given transformation. The transformation time is dependent on the transformation algorithms and the input the algorithm takes (Weiss, 2006, pp. 46). Logarithmic scaling is good, linear can be acceptable even with bigger systems, while exponential scaling is mostly useless for industrial use, as it would take simply too much time. The number of resources cannot be increased exponentially to keep the transformation time under an acceptable level with huge models.

Memory requirement is important for similar reasons: Transformation can require too much memory, which can be infeasible or at least too expensive under some conditions. As stated by Lu, Che & Wang (2009), "Memory wall is an important factor that influences program performance". The amount of memory available is limited in computers, and in case the transformation would take too much memory, it can crash before it could be finished. Memory requirement is hence a measured variable under the experiment.

Hardware related resource scaling describes how well the solution scales when it has more available resources, for example more processors for execution. Parallel execution can lower transformation time, however this is more dependent on the algorithmic complexity and the input length, rather than the number of executing units which the transformation can use.

\(^\text{12}\) http://www.ubka.uni-karlsruhe.de/hylib-bin/suche.cgi?opacdb=UBKA_OPAC&nd=318556863&fbt=4125243-30970569X
The amount of resources is arbitrary in case of a real system, also can be considered constant. Even if there is a C number of resources, in case the transformation takes O(n^x) time, the transformation will be (1/C) * O(n^x), which is still O(n^x). The same applies on memory requirement scaling. This means also that scaling with the available hardware resources can be important in some cases, but is less important than the time and memory requirement development with regards to the size of the input.

Thus, memory and time aspects should be measured to determine if a tool scales. In case there is no built in way to measure the time, an external clock can be used for this. Because the idea is to determine how well big models scale, the clock should be accurate enough to collect useful information, as transformations under one second are not relevant in this case. Alternatively, the system clock could also be used for timing the translation time. The external clock was considered however accurate enough for this purpose, as well as using the system clock may need support for command line execution, which could be infeasible in some cases. The external tool however has no such problem. The upper limit for transformation time is fixed at 15 minutes. In case the model would not translate under 15 minutes the transformation would be considered too slow for industrial use. Also, the main objective is to determine how well the tool scales: The actual values measured for a given input to be transformed can differ from environment to environment, but it is the trend of those measurements with regards to the other controlled variables which is important in this case.

The memory aspect can be a bit more problematic: Eclipse can be regarded as a whole and the built in Task Manager in Microsoft Windows 7 can be used to see how much memory (and memory for the working set) the tool uses, by controlling the memory consumption of Eclipse before and after the transformation. In Windows 7, four memory related data can be gathered relating processes: Commit, working set, private and shared data. Commit describes how much virtual memory the process touched under the session, while the working set describes the physical memory currently used. Private memory is the part of the memory the process can use but cannot share with other processes, while shared memory can be shared. In this case, only the virtual and physical memory development is important, thus only these two will be monitored. The memory requirement can be computed by noting the memory used by Eclipse before and after the transformation, and the difference can be regarded as the memory needs, in case the tool has no built in feature for memory requirement tracking. There is however a limitation: Eclipse working space and commit can vary with every time it starts, however this variation can be considered low enough to not to affect memory requirement curve development too much.

It can be noted that there are other ways to measure quality of the transformation, as described by van Amstel & van den Brand (2010) and van Amstel, van der Brand & Nguyen (2010). The later authors also define size, function complexity, modularity, inheritance, dependency, consistency and input/output metrics. Some of them are similar to the parameters considered for this study, however, they are most focusing on metrics, and not the performance measurements as in this report.

---

6.3 Objective 3: Tool evaluation

The environment is the same through the tests. The environment consists of the following relevant parts:

- Operative system: Windows 7 Ultimate SWE x64
- Eclipse: EMF Juno 1.5.0
- CPU: Intel I5-760 ("Lynnfield" Quad-core, 2.80GHZ)
- Memory: Corsair Vengeance 1333Mhz CL9
- Motherboard: Gigabyte P55A-UD3 Rev 2.0 (P55 Chipset, F11 Firmware)
- HDD: Samsung 830 128GB

ATL behaviour is evaluated in three different scenarios. The first scenario uses a single class single attribute meta-model for transformation. In this part, the impact of various EMF settings is also evaluated. The second scenario uses a more complex, but still relatively simply model to see how the addition of more used classes in the meta-model affects transformation performance. The third scenario uses the same meta-model as the second set of transformation; however an additional mapping rule is added to evaluate how this affects performance.

The only tool which is evaluated is ATL, as QVTo, QVTd and SmartQVT are not considered mature enough according to the criteria defined in Chapter 6.1.1 to be evaluated further.

The benchmarking is done on a home computer: In industrial use, the hardware can differ, but as the question is the scalability, it does not matter how much time or memory a given transformation takes exactly on the given hardware environment. The importance lays on the measured trend as the controlled variable is manipulated.

The controlled variable for the experiment is the size of the model, as well as model complexity. The size of the model is controlled by performing the transformation on models with various sizes. The complexity of the model is controlled by the use of two different meta-models, with one more complex than the other. The number of transformation rules is also controlled: For the simple model there is only one rule, but for the more complex model, there are two transformations, one with only one rule, and one with two. Thus, size of the input model, complexity of the input model and the number of transformation rule are expected to result in different transformation times, affecting measurement curve development.

Time is measured by an external clock. The benchmark focuses on model transformations which require time between one second and fifteen minutes, this method is therefore accurate enough. Each run is repeated to check for reliability. In case the difference would exceed 20%, the run is considered a fail and should be repeated.

Memory requirements are also measured externally. In Windows 7, four different parameters can be gathered of how a program uses the memory. Dedicated memory and the working set are measured both to determine the memory consumption by the transformation, while shared and private memory are considered less important in this case. Eclipse itself consumes some memory already when it is started. The memory consumption for the dedicated memory is measured by subtracting the value for dedicated memory before the transformation from after the transformation for the process "Eclipse.exe". The memory consumption for the working set is done similarly, only with reading the values for the
working set before and after this time. A second, control measurement is done with memory requirement too.

When performing the measurement, it is made sure that no other process is interfering with the experiment. Before starting with a new transformation, Eclipse was always restarted in case it was already running. The transformations were started by the "Run as" built-in functionality from Eclipse.

6.3.1 Single class single attribute meta-model transformation using ATL and default Eclipse settings

In the first transformation, a very simple model is transformed to another very simple model. The source meta-model has one class with one string type attribute. A class instance tends at least to have a name, so this meta-model can be considered the absolute minimum to be researched how well it scales. The target meta-model is a bit more complex, but still very basic: It has two classes with inheritance relation between them, and the super class has also a string type attribute. Figure 2 shows both meta-models. The code for the source meta-model is presented in Appendix B; the code for the target meta-model is presented in Appendix C, while the ATL transformation code is presented in Appendix D. A simple model which can be translated with help of ATL can be seen in Appendix E. It can be noted that a single class instance takes two rows.

![Figure 2 The SCSA source meta-model as well as the target meta-model](image)

With a single class-single (string) attribute (SCSA) model the x86 EMF using ATL for transformation scales as shown in Table 1 with the growth of class instances in the model to be transformed. The "change in memory requirement" is the difference between the used dedicated memory by the process "eclipse.exe" displayed by the Windows built-in Resource Monitor before and after the transformation. "Eclipse.exe" is the executable file associated with the EMF. The "change in working set" displays the difference in working set of "eclipse.exe" before and after the transformation. "Vanilla" EMF x86 means that the file "eclipse.ini" has not been modified to be able to use more dedicated memory and bigger working set, but the ATL plug-in was already installed. The original "eclipse.ini" file is presented in Appendix F.

Appendix G shows the results from the transformation with different sized input models. ATL consumes the default 512Mb dedicated working space already with 300,000 instances and later crashes with 500,000 instances. The repeated tests show two different error messages, the first one is a "java.lang.reflect.InvocationTargetException" and the other is an
"unhandled exception loop exception: java heap space". Both can imply that Eclipse hits the memory wall; also it uses as much memory as it can and cannot allocate more. A memory wall is a confounding variable in this situation, also it can affect the gathered measurements. In case the memory wall does exist and affects the results, the affected results should not be considered, as scalability only accounts for the ideal trend of the measures, but not the ones affected by another factor. To ensure that this is the case, also the memory limitation is a confounding variable, further research on the subject is performed.

![Figure 3 SCSA with ATL transformation time measurements using vanilla EMF x86](image)

Figure 3 shows the development of transformation time with the increasing numbers of instances. Figure 4 shows the development of memory used by the EMF, with respect to both dedicated memory and the working set. Note that only results with successfully transformed models are displayed, also the unsuccessful transformation is not shown on the figures. The difference between 100,000 and 200,000 instances in transition time is similar to that between 200,000 and 300,000 instances, which could imply a linear time scaling. The difference is however higher between 300,000 and 400,000 instances. It can be noted that this rapid increase takes place right after the memory requirement hits the wall. Scalability does not takes into account what happens if the memory wall is hit, as it only concerns the ideal memory allocation development. This is also the reason why this development is researched further, so that only the ideal trend of measures is taken into consideration, but not the ones affected by the memory wall. Both memory aspects are developing linear until they hit the maximum allocation, where they begin to stay near to it. This figure however could also be interpreted as a logarithmic development, so further research on the subject is needed for this reason too.
It can be noted that EMF x86 is by default single threaded, even though this attribute is less important in case of this benchmark. ATL over EMF x64 is also tested, but it only produces a java.lang.reflect.InvocationTargetException error message, which is associated with an abstract error handler. EMF x64 is multithreaded by default, but that is less important in case of this study.

### 6.3.2 Single class single attribute meta-model transformation using ATL and modified Eclipse settings with 1024MB maximum heap size

To ensure that the memory allocation development is not depending on the default maximum heap size, the settings file for EMF is modified so that it can utilize a larger heap. The only difference in these tests lays in the settings of the "eclipse.ini" file: The minimum heap size is left unchanged while the maximum heap size is adjusted to 1024MB. See Appendix H for the modified file. The meta-models (Appendix B and C) as well as the translation code (Appendix D) are the same. These tests are necessary to determine in case the memory requirement is logarithmic, or the memory development is affected by the java heap size. EMF x86 with ATL transformation was used to gather the following information.

Appendix I contains the data gathered during the transformation of the SCSA model with the modified Eclipse x86 setting using a maximum heap size of 1024 MB. The results in Appendix I are visualized in Figure 5 and Figure 6.
As it can be seen on Figure 5, increasing the heap size affects how big models can be translated before Eclipse crashes. The trend of the measures stays linear until 600,000 instances, at which point more memory cannot be allocated from Eclipse.

As it is displayed on Figure 6, memory consumption development is linear for the working set until 600,000 instances, where it comes near to the maximal allocable memory limit for it. The limit is reached for the dedicated memory at 500,000 instances. The same tendency can be noticed as before with the vanilla EMF: As the working set comes near to the maximum usable level, the transformation time begins to increase rapidly. This is not interesting when researching scalability, however, it is interesting because it shows the memory wall.

To additionally ensure that the memory requirement curve development is dependent on the working set, a third benchmark was also done on the SCSA with ATL: The only difference was...
a modified "Eclipse.ini" file, with 1280MB heap size instead of 1024, as it can be seen in Appendix J. Everything else was unchanged.

The results displayed in Appendix K are visualized in Figure 7 and 8. It can be noted that this time the model transformation was successful with 1,200,000 instances, also increasing the heap size means that ATL can translate bigger models with bigger heap size.

Figure 7 SCSA with ATL time measurements using a modified EMF x86 with a heap size of maximum 1240MB

Figure 7 shows that the trend of measures is linear until 700,000 instances, where the transformations begin to hit the memory wall. Also the linear developments lasts with larger input models.

Figure 8 SCSA with ATL memory measurements using a modified EMF x86 with a heap size maximum 1240MB
Figure 8 shows a similar development as figure 6. The difference is that ATL can translate bigger models without crashing, but eventually it uses up all the memory, begins to be significantly slower. The dedicated memory shows a higher jump here too, but the measured trend of the working set as the controlled variable is manipulated is linear until the point it hits the memory wall.

Figure 9 displays a summary of the time trend of measurements for transforming a single class single attribute meta-model using ATL over EMF x86 with different heap sizes. Figure 10 displays a summary of the workspace memory trend of measurements for transforming a single class single attribute meta-model using ATL over EMF x86 with different heap sizes.

As it can be seen on figure 9, ATL transformation time scales exponentially even with a very simple model, with one class, one attribute and one single mapping rule. The transformation time measurements show linear development until ATL hits the memory wall, but when the memory begins to be a limiting factor, the transformation time begins to increase exponentially, eventually resulting in an application crash. The recommended maximum heap size according to IBM is 1.5GB using 32-bit Java, which means that the ATL using the EMF x86 could maybe transform very simple models with 1,500,000 instances, but no more.

---

Looking on Figure 10, also the figure displaying the model scaling it can be concluded that ATL using EMF x86 uses up memory somewhat linearly until the point it cannot use more. This is a problem, as when it hits the memory wall, the translation time begins to increase as the memory wall becomes a confounding variable. Figure 11 displays the trend of measures of the difference in dedicated memory. There is a higher derivation from the ideal linear curve, most notably at 500,000 and 600,000 instances.

Memory is also a limiting factor, with EMF x86 only 1.5GB memory can be used for the working set. EMF x64 has virtually no such limitations according to IBM, but EMF x64 is not performing satisfactory: Using the custom 1280MB heap size, the EMF x64 only manages to translate 300,000 instances under 20 seconds, which is far slower than the x86 implementation. It also crashes already at 400,000 instances showing unexpected behaviour. For this reason, only EMF x86 is used for the later benchmarks.

---

6.3.3 Multiple class single rule (MCSR) meta-model transformation using ATL and modified Eclipse settings with 1280MB maximum heap size

In the previous chapters, single class transformation scaling is evaluated. However, most meta-models do not only consist of a single class. In this chapter, a more complex meta-models are used for both the source and target meta-models. The results can be then used to evaluate if the number of classes affects the resource needs of the transformation. In this case, only EMF x86 was used with a maximum heap size of 1280MB.

The class diagram for the source meta-model for multiple class single rule (MCSR) translation can be seen in Figure 12. The code can be seen in Appendix L. The source meta-model consist of two classes: One ParentClass and one ChildClass, both having a string type attribute. The relation between the parent and the child class is composition: A child class is either TypeA or TypeB, while the parent has ChildA or ChildB accordingly. The relations have 0...1 multiplicity. It can be noted that a single class is four rows long, as it can be seen on a simple source model presented in Appendix O.

Figure 12 The MCSR source meta-model

The class diagram for the target meta-model for multiple class single rule is displayed in Figure 13. The code can be seen in Appendix M. The super class named Class has two subclasses with names TypeA and TypeB, and Class also has a string type attribute DerivedAttribute.

Figure 13 The MCSR target meta-model

The transformation is included in Appendix N. It has two helpers: One for determining if the child class is TypeA, and the other is needed to gather the ParentAttribute from the
ParentClass. There is only one rule in this transformation, which only applies to TypeA ChildClass and creates TypeA Class with a string attribute in the target model.

The results for the transformation are displayed in Appendix P. It can be noticed that this time only models with 800,000 instances can be translated before ATL over the EMF x86 with 1280MB maximum working set crashes, which is lower than it was with single class models. Figure 14 and 15 are displaying the time and memory development with regards to the number of instances. Only measurements unaffected by the memory wall are displayed on the figures.

**Figure 14** MCSR with ATL time measurements using a modified EMF x86 with a heap size of 1280MB

Figure 14 shows the time aspect of the transformation. The measured trend as the controlled variable is manipulated is linear for the transformation time, and the same applies the working set. The curve for the dedicated memory has higher derivations in the size of the differences between the measuring points, but the trend is linear in that case too.

**Figure 15** MCSR with ATL memory measurements using a modified EMF x86 with a heap size of 1280MB
6.3.4 Multiple class multiple rule (MCMR) meta-model transformation using ATL and modified Eclipse settings with 1280MB maximum heap size

In this chapter, the same source and target meta-models will be used as in the previous, MCSR chapter, as well as the same models will be translated. The difference is the number of transformation rules: In the previous chapter, only one translation rule is used, in this chapter, there are two rules used for transformation, hence the name for multiple class multiple rule (MCMR) transformation.

The ATL transformation code is based on the single rule transformation rule used to translate multiple classes. Two different ATL transformations are used in this benchmark. One translates ChildB into TypeB objects as it is presented in Appendix P. The other ATL transformation translates ChildB also into TypeA objects, also both rules creates the same type of class, as it can be seen in Appendix Q.

Appendix R contains the results from the ATL transformation from multiple classes targeting multiple classes. The transformation crashes already with 500,000 instances, which is quite lower, than it was with multiple classes and a single rule. The memory wall is hit at transforming a model with 400,000 instances.

![Figure 16](image)

**Figure 16** MCMR with ATL time measurements targeting multiple classes using a modified EMF x86 with a heap size of 1280MB

Figure 16 shows the trend of time measurements with MCMR targeting two different classes. The measured trend as the controlled variable is manipulated is linear.

![Figure 17](image)

**Figure 17** MCMR with ATL memory measurements targeting multiple classes using a modified EMF x86 with a heap size of 1280MB
The commit memory for the whole process hits the wall already at 300,000 instances in this case, while the working set hits it at 400,000 instances. Figure 17 shows the trend of measurement until 300,000 instances and shows a linear trend for the working set.

Appendix T contains the ATL transformation results with multiple class multiple rules targeting a single class. Appendix S contains the transformation code.

**Figure 18** MCMR with ATL time measurements targeting a single class using a modified EMF x86 with a heap size of 1280MB

Figure 18 shows the trend of time measurements with MCMR using ATL targeting a single class. The measured trend as the controlled variable is manipulated is linear.

**Figure 19** MCMR with ATL memory measurements targeting a single class using a modified EMF x86 with a heap size of 1280MB

Figure 19 shows the memory related measurements of the MCMR ATL transformation targeting a single class. The maximum commit size is reached at 300,000 instances, while the working set hits the wall at 400,000 instances. The curve development for the working set shows a linear trend.
Figure 20 Comparison of MCMR with ATL time measurements targeting single and multiple class using EMF x86 with a heap size of 1280MB

Figure 20 shows the measured trend for transformation time as the controlled variable of number of instances is manipulated for both MCMR transformations on the same figure. It can be concluded that the two curves are overlapping. This concludes that it does not make a difference in terms of time complexity if the rules are targeting the same or different models.

Figure 21 Comparison of MCMR with ATL memory measurements targeting single and multiple class using EMF x86 with a heap size of 1280MB

Figure 21 shows the measured trend for transformation difference in memory requirement as the controlled variable of number of instances is manipulated for both MCMR transformations on the same figure. It can be concluded that the two curve pairs are very similar to each other, showing small differences compared to each other. This concludes that it does not make a difference in terms of memory complexity if the rules are targeting the same or different models.
7 Discussion

7.1 Objective 1: The OSS market situation regarding model transformation tools

The first objective is to describe the current situation regarding open source model transformation tools. So that the benchmarking experiment can be done, only transformation tools working with the Eclipse Modeling Framework (EMF) are considered, because they can directly be compared using the same experimental setting. Four such tools are identified: ATL (Atlas Transformation Language), QVTo (Query/View/Transformation Operational), QVTd (Query/View/Transformation Declarative) and SmartQVT.

To ensure that the benchmarking can be performed, a literature analysis is done to determine the criteria for open-source model-to-mode transformation tools. The first criteria is that the tool is actively in development, the second is that the tool must be stable and the third one is that there should be sufficient documentation for the tool to be used.

The checks are made on the tools. SmartQVT is not developed anymore, meaning it could not pass the first criteria. Because it cannot be installed, QVTd is also discarded.

For QVTo, 40 hours is shown not to be sufficient to build the necessary transformation code as documentation is lacking. Therefore it could not be evaluated further, as the time constraints of this project could not make it possible to try further with the subject.

ATL fulfills every criteria, also it is in development, can be installed and it is possible with it to create the necessary transformation code under the appointed time, thus it is evaluated further.

7.2 Objective 2: Factors that impact scalability

The meta-models, models and transformation definitions are affecting the model transformations, so they are used as controlled variables to evaluate scalability. Two different meta-models are created as well as different transformation rules. The size of the models is also used as a controlled variable throughout the test, with model size of X * 100.000 instances, where X is a positive integer between [1...Y], where Y*100.000 is the smallest integer where the transformation fails.

The measured variables are the time requirement of the model to be transformed, as well as the memory requirement in form of difference in commit and working set size. The memory wall can be a confounding variable in this case, meaning that measurement affected by the memory wall should not taken into consideration when determining scalability.

7.3 Objective 3: Tool evaluation

The only tool qualified for the evaluation is ATL, as both SmartQVT, QVTo, QVTd has flaws which prohibit eventual benchmarking.

The smallest models included in this benchmark had at least 100.000 instances, as smaller models take mostly under one second to translate, and are thus considered too small to be taken into consideration. The tests with growing models are performed until the application
crashes. A 15 minute stop criteria is present for transformations, with everything over 15 minutes considered too high. This criteria is however always passed with successful transformations.

Two meta-model pairs are used for the benchmarking. One has only one source and target class, with one respective string attribute as so much information is needed for a class which has its own name. The other pair is slightly more complex, with composition on one side and inheritance on the other. A comparison on the two pairs shows how the number of classes present in the meta-models can affect scalability.

On the meta-model pair having multiple classes, even the effect of having a single or multiple transformation rules is evaluated, even with multiple transformation rules targeting a single class or multiple classes. The summary on the results as well as a short evaluation of them is presented below.

Figure 22 SCSA with ATL transformation time measurements using a modified EMF x86 with different heap sizes

Figure 22 shows that the even with the single class single attribute source and target meta-models, the heap size defined by the Eclipse Modelling Framework x86 affects the number of instances which can be translated before ATL over EMF crashes. Also, maximum heap size for EMF is a confounding variable. Even with the highest heap size, only 1,200,000 instances could be translated using a single class single target meta-model source-target pair. EMF x64 was also tested, but it crashed with lower models than the x86 variant, thus EMF x86 was used later on in the benchmarking, as it was more stable.
Figure 23 SCSA with ATL working set memory measurements using a modified EMF x86 with different heap sizes

Figure 23 shows the working set memory requirement scaling with respect to the number of instances in the models as well as the heap size defined in EMF x86. Figure 24 shows the trend of measurements regarding the dedicated memory. It can be concluded that the EMF x86 heap size affects the measurements, as hitting the memory wall has an effect with every three measured trends. When only looking at the measurements unaffected by the memory wall, all of the three parameters have a linear development, with the dedicated memory having prudentially higher difference derivation between measurement points as the two others.

Figure 24 SCSA with ATL dedicated memory measurements using a modified EMF x86 with different heap sizes
Figure 25 ATL transformation time measurements using a modified EMF x86 with a heap size of 1280MB

Figure 25 sums up the time scaling behaviour of ATL now with a fixed heap size. The trend of measures is linear in both three displayed cases. The transformation curve is steeper with multiple classes and even steeper with multiple rules. However, it does not make significant difference if there multiple rules are targeting same or different classes. The single class single rule transformation manages to translate 1,200,000 instances, the multiple class single rule can transform models with up to 800,000 instances, while the multiple class multiple rule transformation manages only 400,000 instances. As there is a confounding variable, namely the maximum heap size, only those measurements are taken into consideration to determine scalability which are unaffected by the memory wall.

Figure 26 ATL transformation dedicated memory measurements using a modified EMF x86 with a heap size of 1280MB

Figure 26 and 27 sums up the memory requirement scaling using ATL for transformation with regards to number of class instances and different meta-models.
Figure 27 ATL transformation dedicated memory measurements using a modified EMF x86 with a heap size of 1280MB

The memory requirement is controlled by measuring both the difference of dedicated memory and the working set before and after the transformation. The measured trend for the working set as the controlled variable number of instances is manipulated is linear for all of the different model complexities. The differences between measures can differ, but this does not affect the overall trend. The derivation between measurements is bigger for the dedicated memory, but the overall trend is linear in this case too. The complexity of the classes and the number of rules affects however how steep the curves are and how big models with number of instances can be translated, but the measurements show a linear trend in every case. It can be also noted that the difference between having multiple rules targeting the same or different classes is within error margins.

Thus, it can be concluded that ATL scales linearly with regards to the input size with both transformation time and memory consumption. More complex models have also linear scalability, however the difference between measurements is bigger, thus the measurement development curve is steeper.
8 Conclusion

8.1 Contribution

The aim for this report is to investigate the scalability of open source tools for model to model transformation. This aim is divided into three objectives, which objectives are fulfilled to achieve the aim.

The objective about current situation regarding open source model transformation tools is described by presenting the four OSS tools suitable for model transformation under the study over the Eclipse Modeling Framework. Criterion are determined by a literature analysis to ensure that transformation tools fulfilling them can be benchmarked afterwards: These criterion are active development, ability to install the tool and sufficient documentation to create the necessary transformation code for the benchmark. From the four tools only three is actively in development, meaning that SmartQVT is discarded as it does not developed anymore. Query/View/Transformations Declarative (QVTd) could not be downloaded, thus it does not live up to the criteria that the tool should be able to be installed. Query/View/Transformations Operative can be installed, but the documentation for the tool is lacking meaning which means that the transformation code could not be created within the given time frame. ATL fulfils the criterion and can be benchmarked. This concludes that the first objective is achieved.

The factors needed for the observation during the benchmark are determined, also the second objective is also fulfilled.

The only open-source tool for model to model transformation identified and fulfilling every criteria is benchmarked, and its scalability is determined. ATL has a linear measured trend as the controlled variable is manipulated for both the time and memory requirement aspect, which are needed to evaluate scalability.

This paper can also be a help to industrial companies seeking an open-source tool for model-to-model transformation to choose right: This document identifies a model transformation tool which has a linear time and memory requirement dependency on the input model. Companies can decide, if this scalability is suitable for their needs. As the document identifies criteria for determining if a given tool could be benchmarked as well as the how a benchmark can be performed to determine scalability, industrial companies can use these information to create their own benchmarks with tools not discussed in this report. This benchmark could also be used to compare tools in case they can identify more than one. These information can help industrial companies choose right, resulting in a lower adaptation cost as they can use information gathered in this thesis project. This is true for both companies already using proprietary solutions and companies interested in switch to model-driven development.

8.2 Limitations of the method and approach

It is important that the gathered results are valid and reliable, also that it is measured what should be measured, as well as the measurements are consistent. To determine scalability, the measured trend as the controlled variable is manipulated should be not affected by confounding variables. Therefore, it is researched in the report how the memory wall affects the measurements, as well as the results. Measurements affected by the maximum heap size
in EMF are not taken into consideration when determining tool scalability. This is important to ensure that the measurements are right.

The environment is controlled: The same computer is used throughout the tests, with other unnecessary programs for the benchmarks disabled and the network connection physically removed, so that the benchmarks are not affected by other processes.

Scalability is how the ideal trend of measurements develop. In industrial use, chances are low that model transformation would happen using the same environment as is used under the benchmarks. However, even though the actual hardware and software environment can differ, scalability should be the same. Also, the relation between the measured points should be the same, also a linear trend will be linear even on a completely different environment used in the industry. This is why the measurement development trends are relevant and usable even for industrial use, as scalability does not depend on the environment.

Control measurements are also performed to ensure that the measured values are right. Every measurement was followed up with a control measurement and was ensured that the measurements are only considered reliable in case both of them stayed within a given error marginal compared to each other.

The study has also own limitations: This given project corresponds to 15 credits, which is equivalent to 400 hours. This is a limitation of the study and also a reason why QVTo is discarded after 40 hours showed to be insufficient to build the necessary transformation with it. This factor affected the results to that only one tool could be benchmarked.

Another limitation is the memory resources available: Even though the environment has 16 GB of memory, only EMF x86 could be used, meaning a lower memory wall. Lower memory wall means that the number of measurements taken before hitting the memory wall and getting unusable measurements is also lower, leaving lower number of measurements to base the trends. Few measurement points mean that the trend is more difficult to interpret with confidence. On the other hand, as long as there are at least measurements, the trend can be determined, and in most of the cases, more than three measurement points were unaffected.

### 8.3 Future work

Originally, it was planned to use bigger meta-models based on the Action Language for Foundational UML (ALF). As it was seen that even models with relatively small meta-models can crash rather early with under 1.500.000 instances and transformation complexity makes the situation with ATL even worse, these tests were discarded as the gathered data showed already evidence considered to be enough to determine scalability. On the other hand, it could be interesting to try even bigger models and see how they fare.

QVTo was dropped because the documentation was considered lacking and it took too much time to learn how to use it. The limitation on how much time could be spent on a single tool was a necessary stopping criterion for this work to be completed on time, but it could be interesting to see how it scales. Evidence could be found that it scales better than ATL. QVTo could also be interesting now that its download page is alive again, however it has even less documentation than QVTo. Additionally, it could be interesting to look at other model transformation tools even outside EMF.
Regarding the meta-models, the number of classes was evaluated as an input variable for the experiment, but not the number of attributes. The impact of higher number of attributes used in the models could be interesting.

Alternatively, a case study could also be used as a method. Also, the chosen tools could be taken to a company in the industry and the development process could be changed to review, how a given tool or tools would fare in a real-life scenario, as well as in case they does not scale well, how they could be changed so that they could fit in the process of an industrial company. In case the company already uses MDD, it could be interesting to compare the open-source tools to the proprietary tools. Some of the companies have long record with the use of MDD as stated in Appendix A by "Person A" (2013), so this comparison could be interesting how well the open-source transformation tools should scale before they could compete directly with currently used industrial transformation tools. Völter & Stahl (2006) also states that the for embedded use, MDD adaptation grows also very quickly, so evaluating an embedded scenario with a case study could also be interesting. Such an evaluation was however far beyond the bounds of this evaluation, even though it could yield very interesting results.

8.4 Ethical factors

The aim of the report is motivated detailed to reinforce that this thesis project is beneficial for the society, as the use of MDD can be beneficial for the industrial companies, but even for people using their products.

The project was performed with respect to research methodical factors. Validity and reliability was taken into consideration as described in chapter 8.2. The experiment was performed so that it could be repeated, thus the findings can be validated by others.

Ethical factors were considered when writing the report. The name of the interviewed person in Appendix A is made anonym, and so are the companies named in the interview. This way, the interview and the work is anonym to secure personal integrity of the interviewed person.

There are some important factors which were important when performing the interview. Having the right questions is important: The questions should not be leading. Most of the questions are open ended, which is important to gather a resource rich interview. The language used is Swedish, which felt more suitable for this reason than English used in the rest of the document.

The answers given by the interview was also followed up and controlled that they have been understood right, however it was a bit difficult with feedback. The reason why the interview can be considered compact is that the interviewed person can be regarded busy, therefore it was important the answering it should not take more time that necessary.
References


Bakar, N. S. A. A. & Boughton, C. V. (2012) Validation of measurement tool to extract metrics from open source projects. IEEE Conference on Open Systems


Appendix A - Interview with "Person A"

[Email interview 2013-02-06/07.]

- Använder ni Model Driven Development i företaget du jobbar för?
  o Ja

- Ifall ni använder det, när har ni börjat med att använda det?
  o Jag började på [Company A] 1997 och det var första gången jag kom i kontakt med MDD. Jag vet inte exakt när företaget började med MDD innan min tid, omkring 1990 kanske. I dag jobbar jag på [Company B] och jag har ingen information om vilket årtal man började med MDD. Men jag kan tänka mig att det var väldigt tidigt p.g.a. att regleralgoritmer oftast modelleras i en eller annan form.

- Finns det speciella områden där du anser att MDD är mer lämplig att använda?
  o Det passar så gott som inom alla områden där information behöver beskrivas och tydliggöras mellan intressenter. MDD begreppet infattar så mycket, en modell kan vara en UML modell det kan också vara en regleralgoritm uttryckt med symboler istället för de matematiska uttrycken.

- Vilka fördelar finns det med att använda MDD?
  o Modellering handlar om att formalisera struktur eller beteende så att går att kommunicera vidare i utvecklingsprocessen inom det specifika området eller mellan olika områden.

- Vad var anledningen till att företaget började använda MDD?
  o Tydliggöra karv och design

- Kan du nämna några nackdelar som du upplevde i samband med att använda MDD?
  o Det är svårt att förändra en organisations arbetsätt till MDD, från att vara dokument och kod intensivet till att kunna uttrycka sig mer "formellt" i modeller.

- Kan du nämna några fördelar med de verktyg ni använder?
  o Vissa av verktygen har en öppen arkitektur, vilket gör det möjligt för oss att anpassa verktyget speciellt bara för vår verksamhet. Med de andra måste en dialog föras med verktygsleverantören för att få igenom bugrättningar och funktionstillväxt i verktyget.

- Finns det några nackdelar med dem som du anser att man kunde förbättra?
  o Ja

- Använder ni domänspecifika eller proprietära verktyg?
  o Båda

- Som utvecklare, vad anser du, vilka fördelar och nackdelar finns mellan att utveckla med MDD och tredje generationens programspråk?
  o Fördelar: Jag kan uttrycka min design på olika nivåer och samtidigt analysera designen utifrån olika systemaspekter.
  o Nackdelar: Om jag inte lyckas att abstrahera och formalisera mina krav i en design, kan det ibland "känns" lättare att bara börja koda något och iterera tills en lösning kanske finns. Men denna nackdel beror oftast på att den kravbild jag har är ofullständig och då "känns" det lättare att start direkt i kod.

- Vad tror du, använder andra företag också MDD för utveckling av mjukvara?
  o Ja

- Kan du motivera ditt svar lite, varför anser du det?
Därför att MDD begreppet är så stort. Många använder olika typer av modeller och all dessa olika typer av modeller fyller ett specifikt syfte inom begreppet MDD.
Appendix B - Single class single attribute source meta-model

```xml
<?xml version="1.0" encoding="UTF-8"?>
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    name="SourceClasses">
    <eClassifiers xsi:type="ecore:EClass" name="ClassA">
        <eStructuralFeatures xsi:type="ecore:EAttribute" name="attributeA"
            eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#EString"/>
    </eClassifiers>
</ecore:EPackage>
```
Appendix C - Target meta-model for the single class single attribute transformation

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <eClassifiers xsi:type="ecore:EClass" name="Class">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="derivedAttribute" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore://EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="TypeA" eSuperTypes="#//Class"/>
</ecore:EPackage>
```
Appendix D - ATL transformation code for single class single attribute transformation

```
module SourceToTarget;
create OUT : TargetClasses from IN : SourceClasses;

rule SourceClassA2TargetClass{
  from
    s: SourceClasses!ClassA
  to
    t: TargetClasses!TypeA (;
        derivedAttribute <- 'ClassA' + s.attributeA
    )
}
```

Appendix E - Source model sample for the single class single attribute transformation

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns="SourceClasses">
  <ClassA attributeA="memberA">
  </ClassA>
</xmi:XMI>
```
Appendix F - The vanilla "Eclipse.ini" file

-startup
plugins/org.eclipse.equinox.launcher_1.3.0.v20120522-1813.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.1.200.v20120913-144807
--product
org.eclipse.epp.package.modeling.product
--launcher.defaultAction
openFile
--launcher.XXMaxPermSize
256M
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Dhelp.lucene.tokenizer=standard
-Xms40m
-Xmx512m
Appendix G - SCSA with ATL scaling with respect to number of instances using vanilla EMF x86

<table>
<thead>
<tr>
<th>Number of instances</th>
<th>Transformation Time</th>
<th>Difference in dedicated memory (commit)</th>
<th>Difference in working set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.000</td>
<td>3 sec</td>
<td>172.600KB</td>
<td>146.900KB</td>
</tr>
<tr>
<td>200.000</td>
<td>6 sec</td>
<td>331.200KB</td>
<td>313.100KB</td>
</tr>
<tr>
<td>300.000</td>
<td>9.5 sec</td>
<td>459.900KB</td>
<td>457.680KB</td>
</tr>
<tr>
<td>400.000</td>
<td>20 sec</td>
<td>461.200KB</td>
<td>469.500KB</td>
</tr>
<tr>
<td>500.000</td>
<td>385 sec (crashed)</td>
<td>458.900KB (crashed)</td>
<td>470.300KB (crashed)</td>
</tr>
</tbody>
</table>
Appendix H - The first modification on "Eclipse.ini" with 1024MB working space

-startup
plugins/org.eclipse.equinox.launcher_1.3.0.v20120522-1813.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.1.200.v20120913-144807
--product
org.eclipse.epp.package.modeling.product
--launcher.defaultAction
openFile
--launcher.XXMaxPermSize
256M
--showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256M
--launcher.defaultAction
openFile
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Dhelp.lucene.tokenizer=standard
-Xms48m
-Xmx1024m
## Appendix I - SCSA with ATL scaling with respect to number of instances using a modified EMF x86 with a heap size of maximum 1024MB

<table>
<thead>
<tr>
<th>Number of instances</th>
<th>Transformation Time</th>
<th>Change in memory requirement (commit)</th>
<th>Change in working set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0000</td>
<td>3 sec</td>
<td>139.800KB</td>
<td>142.600KB</td>
</tr>
<tr>
<td>200.000</td>
<td>6 sec</td>
<td>285.500KB</td>
<td>306.400KB</td>
</tr>
<tr>
<td>300.000</td>
<td>9 sec</td>
<td>517.500KB</td>
<td>475.400KB</td>
</tr>
<tr>
<td>400.000</td>
<td>12 sec</td>
<td>586.300KB</td>
<td>588.100KB</td>
</tr>
<tr>
<td>500.000</td>
<td>15.5 sec</td>
<td>983.400KB</td>
<td>775.600KB</td>
</tr>
<tr>
<td>600.000</td>
<td>19 sec</td>
<td>984.200KB</td>
<td>897.100KB</td>
</tr>
<tr>
<td>700.000</td>
<td>28 sec</td>
<td>986.300KB</td>
<td>982.900KB</td>
</tr>
<tr>
<td>800.000</td>
<td>36 sec</td>
<td>985.000KB</td>
<td>986.100KB</td>
</tr>
<tr>
<td>900.000</td>
<td>55 sec</td>
<td>983.800KB</td>
<td>986.400KB</td>
</tr>
<tr>
<td>1.000.000</td>
<td>132 sec</td>
<td>982.900KB</td>
<td>987.500KB</td>
</tr>
<tr>
<td>1.100.000</td>
<td>Crashed after 319 sec</td>
<td>986.100KB (Crashed)</td>
<td>991.800KB (Crashed)</td>
</tr>
</tbody>
</table>
Appendix J - The second modification on "Eclipse.ini" with 1280MB working space

-startup
plugins/org.eclipse.equinox.launcher_1.3.0.v20120522-1813.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.1.200.v20120913-144807
--product
org.eclipse.epp.package.modeling.product
--launcher.defaultAction
openFile
--launcher.XXMaxPermSize
256M
--showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Dhelp.lucene.tokenizer=standard
-Xms512m
-Xmx1280m
-XX:PermSize=256M
-XX:MaxPermSize=512M
Appendix K - SCSA with ATL scaling with respect to number of instances using a modified EMF x86 with a maximum heap size of 1280MB

<table>
<thead>
<tr>
<th>Number of instances</th>
<th>Transformation Time</th>
<th>Change in memory requirement (commit)</th>
<th>Change in working set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0000</td>
<td>3 sec</td>
<td>144.400KB</td>
<td>149.900KB</td>
</tr>
<tr>
<td>200.000</td>
<td>6 sec</td>
<td>269.800KB</td>
<td>281.300KB</td>
</tr>
<tr>
<td>300.000</td>
<td>9 sec</td>
<td>520.400KB</td>
<td>471.200KB</td>
</tr>
<tr>
<td>400.000</td>
<td>11.5 sec</td>
<td>652.000KB</td>
<td>589.000KB</td>
</tr>
<tr>
<td>500.000</td>
<td>14.5 sec</td>
<td>1.013.500KB</td>
<td>793.600KB</td>
</tr>
<tr>
<td>600.000</td>
<td>17 sec</td>
<td>1.029.400KB</td>
<td>915.400KB</td>
</tr>
<tr>
<td>700.000</td>
<td>20 sec</td>
<td>1.082.700KB</td>
<td>1.063.300KB</td>
</tr>
<tr>
<td>800.000</td>
<td>27 sec</td>
<td>1.257.300KB</td>
<td>1.179.000KB</td>
</tr>
<tr>
<td>900.000</td>
<td>35 sec</td>
<td>1.259.400KB</td>
<td>1.263.600KB</td>
</tr>
<tr>
<td>1,000.000</td>
<td>46 sec</td>
<td>1.257.800KB</td>
<td>1.249.400KB</td>
</tr>
<tr>
<td>1.100.000</td>
<td>60 sec</td>
<td>1.247.200KB</td>
<td>1.270.200KB</td>
</tr>
<tr>
<td>1.200.000</td>
<td>94 sec</td>
<td>1.246.500KB</td>
<td>1.277.200KB</td>
</tr>
<tr>
<td>1.300.000</td>
<td>1275 sec (Crashed)</td>
<td>1.246.000KB (Crashed) (Crashed)</td>
<td>1.271.100KB (Crashed)</td>
</tr>
</tbody>
</table>
Appendix L - Multiple class source meta-model

<?xml version="1.0" encoding="UTF-8"?>
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    name="SourceClasses">
  <eClassifiers xsi:type="ecore:EClass" name="ParentClass">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ParentAttribute"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ChildA"
        eType="#//ChildClass"
        containment="true" eOpposite="#//ChildClass/TypeA"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ChildB"
        eType="#//ChildClass"
        containment="true" eOpposite="#//ChildClass/TypeB"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="ChildClass">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ChildAttribute"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="TypeA"
        eType="#//ParentClass"
        eOpposite="#//ParentClass/ChildA"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="TypeB"
        eType="#//ParentClass"
        eOpposite="#//ParentClass/ChildB"/>
  </eClassifiers>
</ecore:EPackage>
Appendix M - Multiple class target meta-model

<?xml version="1.0" encoding="UTF-8"?>
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="TargetClasses">
  <eClassifiers xsi:type="ecore:EClass" name="Class">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="derivedAttribute"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#/EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="TypeA" eSuperTypes="#//Class"/>
  <eClassifiers xsi:type="ecore:EClass" name="TypeB" eSuperTypes="#//Class"/>
</ecore:EPackage>
Appendix N - ATL transformation code for multiple class single rule (MCSR) transformation

```plaintext
module SourceToTarget;
create OUT : TargetClasses from IN : SourceClasses;

helper context SourceClasses!ChildClass def: isTypeA(): Boolean =
    if not self.TypeA.oclIsUndefined() then
        true
    else
        false
    endif;

helper context SourceClasses!ChildClass def: parentAttribute: String =
    if not self.TypeA.oclIsUndefined() then
        self.TypeA.ParentAttribute
    else
        self.TypeB.ParentAttribute
    endif;

rule SourceClassA2TargetClass{
    from
        s: SourceClasses!ChildClass (s.isTypeA())
    to
        t: TargetClasses!TypeA (derivedAttribute <- 'ClassA' + ' ' + s.parentAttribute + ' ' + s.ChildAttribute)
}
```
Appendix O - Multiple class source model sample

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns="SourceClasses">
  <ParentClass ParentAttribute="Parent0">
    <ChildA ChildAttribute="ChildA"/>
    <ChildB ChildAttribute="ChildB"/>
  </ParentClass>
</xmi:XMI>
```
### Appendix P - MCSR with ATL scaling with respect to number of instances using EMF x86 with a heap size of 1280MB

<table>
<thead>
<tr>
<th>Number of instances</th>
<th>Transformation Time</th>
<th>Change in memory requirement (commit)</th>
<th>Change in working set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.000</td>
<td>7 sec</td>
<td>2.000KB</td>
<td>177.000KB</td>
</tr>
<tr>
<td>200.000</td>
<td>12 sec</td>
<td>3.400KB</td>
<td>327.000KB</td>
</tr>
<tr>
<td>300.000</td>
<td>18 sec</td>
<td>419.000KB</td>
<td>610.400KB</td>
</tr>
<tr>
<td>400.000</td>
<td>26 sec</td>
<td>790.000KB</td>
<td>756.700KB</td>
</tr>
<tr>
<td>500.000</td>
<td>32 sec</td>
<td>794.000KB</td>
<td>1.024.000KB</td>
</tr>
<tr>
<td>600.000</td>
<td>46 sec</td>
<td>792.000KB</td>
<td>1.149.500KB</td>
</tr>
<tr>
<td>700.000</td>
<td>69 sec</td>
<td>792.000KB</td>
<td>1.150.800KB</td>
</tr>
<tr>
<td>800.000</td>
<td>146 sec</td>
<td>791.000KB</td>
<td>1.153.000KB</td>
</tr>
<tr>
<td>900.000</td>
<td>930 sec (Crashed)</td>
<td>1.095.000KB (Crashed)</td>
<td>1.151.000KB (Crashed)</td>
</tr>
</tbody>
</table>
Appendix Q - ATL transformation code for multiple class multiple rule (MCMR) transformation with different class target

```plaintext
-- @path SourceClasses=/BENCHMARK%ATL%MULTIPLE%CLASS/SourceMetaModel.ecore
-- @path TargetClasses=/BENCHMARK%ATL%MULTIPLE%CLASS/TargetMetaModel.ecore

module SourceToTarget;
create OUT : TargetClasses from IN : SourceClasses;

helper context SourceClasses!ChildClass def: isTypeA(): Boolean =
    if not self.TypeA.oclIsUndefined() then
        true
    else
        false
    endif;

helper context SourceClasses!ChildClass def: parentAttribute: String =
    if not self.TypeA.oclIsUndefined() then
        self.TypeA.ParentAttribute
    else
        self.TypeB.ParentAttribute
    endif;

rule SourceClassA2TargetClass{
    from
        s: SourceClasses!ChildClass (s.isTypeA())
    to
        t: TargetClasses!TypeA (derivedAttribute <- 'ClassA' + ' ' + s.parentAttribute + ' ' + s.ChildAttribute)
}

rule SourceClassB2TargetClass{
    from
        s: SourceClasses!ChildClass (not s.isTypeA())
    to
        t: TargetClasses!TypeB (derivedAttribute <- 'ClassB' + ' ' + s.parentAttribute + ' ' + s.ChildAttribute)
}
```
## Appendix R - MCMR targeting multiple classes with ATL scaling with respect to number of instances using EMF x86 with a heap size of 1280MB

<table>
<thead>
<tr>
<th>Number of instances</th>
<th>Transformation Time</th>
<th>Change in memory requirement (commit)</th>
<th>Change in working set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.000</td>
<td>10.5 sec</td>
<td>460.400KB</td>
<td>392.700KB</td>
</tr>
<tr>
<td>200.000</td>
<td>20 sec</td>
<td>955.400KB</td>
<td>791.500KB</td>
</tr>
<tr>
<td>300.000</td>
<td>29 sec</td>
<td>1.214.900KB</td>
<td>1.126.600KB</td>
</tr>
<tr>
<td>400.000</td>
<td>58.5 sec</td>
<td>1.248.400KB</td>
<td>1.269.500KB</td>
</tr>
<tr>
<td>500.000</td>
<td>565 sec (Crashed)</td>
<td>1.247.200KB (Crashed)</td>
<td>1.268.700KB (Crashed)</td>
</tr>
</tbody>
</table>
Appendix S - ATL transformation code for multiple class multiple rule (MCMR) transformation with same class target

```plaintext
-- @path SourceClasses=/BENCHMARK%ATL%MULTIPLE%CLASS/SourceMetaModel.ecore
-- @path TargetClasses=/BENCHMARK%ATL%MULTIPLE%CLASS/TargetMetaModel.ecore

module SourceToTarget;
create OUT : TargetClasses from IN : SourceClasses;

helper context SourceClasses!ChildClass def: isTypeA(): Boolean =
  if not self.TypeA.oclIsUndefined() then
    true
  else
    false
  endif;

helper context SourceClasses!ChildClass def: parentAttribute: String =
  if not self.TypeA.oclIsUndefined() then
    self.TypeA.ParentAttribute
  else
    self.TypeB.ParentAttribute
  endif;

rule SourceClassA2TargetClass{
  from
    s: SourceClasses!ChildClass (s.isTypeA())
  to
    t: TargetClasses!TypeA (derivedAttribute <- 'ClassA' + ' ' + s.parentAttribute + ' ' + s.ChildAttribute)
}

rule SourceClassB2TargetClass{
  from
    s: SourceClasses!ChildClass (not s.isTypeA())
  to
    t: TargetClasses!TypeA (derivedAttribute <- 'ClassB' + ' ' + s.parentAttribute + ' ' + s.ChildAttribute)
}
```

65
Appendix T - MCMR targeting a single class with ATL scaling with respect to number of instances using EMF x86 with a heap size of 1280MB

<table>
<thead>
<tr>
<th>Number of instances</th>
<th>Transformation Time</th>
<th>Change in memory requirement</th>
<th>Change in working set</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.000</td>
<td>10.5 sec</td>
<td>508.800KB</td>
<td>408.400KB</td>
</tr>
<tr>
<td>200.000</td>
<td>20 sec</td>
<td>954.100KB</td>
<td>794.900KB</td>
</tr>
<tr>
<td>300.000</td>
<td>29 sec</td>
<td>1.257.200KB</td>
<td>1.178.400KB</td>
</tr>
<tr>
<td>400.000</td>
<td>59 sec</td>
<td>1.249.100KB</td>
<td>1.271.900KB</td>
</tr>
<tr>
<td>500.000</td>
<td>552* sec (Crashed)</td>
<td>1.245.300KB (Crashed)</td>
<td>1.267.500KB (Crashed)</td>
</tr>
</tbody>
</table>