

# eMoflon: Leveraging EMF and Professional CASE Tools

Anthony Anjorin\*, Marius Lauder\*, Sven Patzina<sup>†</sup> and Andy Schürr

{anthony.anjorin, marius.lauder, sven.patzina, andy.schuerr}@es.tu-darmstadt.de

**Abstract:** MOFLON supports standard compliant metamodeling, code generation and model transformations. Development started in 2002 and it has since then been used successfully in a number of case studies for various applications. In recent years, the Eclipse Modeling Framework (EMF) has become a *de facto* standard, offering stable and well-tested components. In addition, numerous professional, industrial strength CASE tools have become increasingly affordable and open to extensions. We are convinced that it is high-time to re-engineer MOFLON and leverage modern MDSD technology. In this paper, we report on a complete re-engineering of MOFLON to face future challenges and meet industrial requirements. We present the new eMoflon, listing the various factors that led to our decision to re-engineer the old system, sharing our experience in tailoring a professional UML CASE tool for our frontend, explaining how we combined EMF and Eclipse technologies with a generic model transformation engine, and discussing our support for a safe interaction between automatically generated and hand-crafted code via an explicitly modeled facade.

## 1 Introduction and Motivation

As the complexity of software systems increases in response to expanding areas of application and new requirements, model-driven techniques have established themselves as a viable means of coping with this complexity. The primary goal of Model-Driven Software Development (MDSD) is to improve productivity by providing tools that are tailored for a specific domain [AK03]. Such domain specific Computer Aided Software Engineering (CASE) tools exploit domain knowledge to further raise abstraction levels and automate complex, but routine programming tasks whenever possible. Anticipated advantages include an increase in speed of development and quality of software, improved response to change, support for interoperability, and a reduced gap between requirements and the actual specification of a solution [AK03]. In many cases, the last point allows for intense cooperation with and validation by domain experts, who might not have any professional programming experience. The central concept in MDSD is that of a *model*, an abstraction of a system that is formed with a specific goal in mind for which it is suitable by construction. A *metamodel* can be viewed as a Domain Specific Language (DSL) that captures the relevant concepts and relationships in a certain domain. Models that are valid with respect to a certain metamodel *conform to* the metamodel. Besides the

---

\*Supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School of Computational Engineering at TU Darmstadt.

<sup>†</sup>Supported by Center for Advanced Security Research Darmstadt ([www.cased.de](http://www.cased.de)).

actual process of abstraction or *metamodeling*, model *transformations* play a central role in MDS. Being able to modify models via transformations is a fundamental task for numerous MDS applications such as tool integration.

MOFLON<sup>1</sup> is a tool for building tools [A<sup>+</sup>06]: a *meta-CASE* tool that supports the MDS process by providing a metamodeling and model transformation framework. Major goals of MOFLON include conformity with standards, reuse of stable external components as much as possible, and industrial relevance implying scalability, usability and integration with Commercial Off-The-Shelf (COTS) tools and established workflows.

In recent years, there has been a lot of progress in the MDS community and especially the Eclipse Modeling Framework (EMF) has established itself as a *de facto* standard, offering stable and well-tested components. We are convinced that it is high time to re-engineer MOFLON and leverage modern MDS technology to concentrate fully on our core competencies.

In this paper, we present *eMoflon*, discussing requirements, our design decisions and giving an overview of the new architecture and our choice of standards, libraries and platform. Our contributions are:

1. Reporting on the various factors that led to our decision to re-engineer MOFLON.
2. Sharing our experience in tailoring a professional UML CASE tool for metamodeling and model transformations purposes, as opposed to handcrafting a new and dedicated editor with numerous editor frameworks such as the Graphical Modeling Framework (GMF).
3. Explaining how we combined standard EMF and Eclipse technologies with a generic model transformation engine.
4. Discussing our support for a safe interaction between automatically generated and hand-crafted code via a clean and explicitly modeled *facade*.

The paper is structured as follows: Sec. 2 gives a brief history of MOFLON 1.0 - 1.5, and highlights our historical goals and competencies. An example, used in the rest of the paper, is introduced in Sec. 3. Section 4 discusses various challenges the old system was facing and motivates a complete re-engineering. Section 5 presents our new requirement profile and primary goals, while Sec. 6 gives a system overview and justifies our design decisions and choices. In Section 7, we compare *eMoflon* with other meta-CASE tools, highlighting differences and similarities. Section 8 concludes with a brief summary and an overview of future work.

## 2 A Brief History and Description of MOFLON

When the development of MOFLON started in 2002, most existing meta-CASE tools lacked formal foundations and were not standard compliant<sup>2</sup> [A<sup>+</sup>06]. Based on the CASE

---

<sup>1</sup>[www.moflon.org](http://www.moflon.org)

<sup>2</sup>MOFLON conforms to Meta Object Facility (MOF) 2.0 of the Object Management Group (OMG).

tool FUJABA [N<sup>+</sup>00], developed by the Software Engineering department of the University of Kassel, the main goal with MOFLON was to improve tool support for MDS by providing a complete and standard compliant meta-CASE tool, with further support for formally founded model transformations. FUJABA was chosen as a platform for MOFLON as it provided an Integrated Development Environment (IDE) framework and a generic model transformation engine based on a graph grammar formalism. In 2004, a first beta version of MOFLON as a plugin for FUJABA 4 was completed. As the CASE tool FUJABA only supported UML 1.4, and MOFLON aimed to support the OMG's upcoming MOF 2.0 metamodeling standard, a complex adapter layer was implemented to allow for data interchange between the different modules in FUJABA. MOFLON supported Java Metadata Interfaces (JMI) compliant code generation for MOF 2.0 metamodels, and MOF-XMI for (meta)model serialization. After modeling the MOF 2.0 metamodel for MOFLON in Rational Rose for the first time, MOFLON was henceforth modeled, extended and maintained using MOFLON itself. Throughout the whole history of MOFLON, using our own technologies (*bootstrapping*) has always been an important proof-of-concept and major system test.

The first stable release, MOFLON 1.0, was in 2006. MOFLON boasted a graphical editor for building MOF 2.0 compliant metamodels, a MOF-XMI file import and export from Rational Rose, and JMI code generation. FUJABA supports unidirectional model transformations via Story Driven Modeling (SDM) [F<sup>+</sup>00], which combines concepts from UML activity diagrams and collaboration diagrams with graph transformations. SDM code generation and editors from FUJABA were successfully integrated with MOFLON 1.0, allowing for the implementation of methods, in MOF 2.0 compliant class diagrams, via SDMs. MOFLON 1.1 was released in summer 2007 and its major new feature was a graphical editor for Triple Graph Grammars (TGGs) [K<sup>+</sup>10]. TGGs provide a declarative means of specifying bidirectional model transformations. From each production rule in a TGG, a set of unidirectional model transformations (SDMs) are derived. A control algorithm uses these derived model transformations to keep two models synchronized, producing a third traceability model that stores the correspondences between the two integrated models. The control algorithm, together with the derived SDMs, is guaranteed to have certain useful formal properties such as completeness, correctness and termination [K<sup>+</sup>10]. By combining TGGs, SDMs and MOF metamodeling, first integration scenarios could be established, integrating tools such as Telelogic Doors (requirements) with Enterprise Architect (Use Case diagrams). MOFLON 1.2 (2007), 1.3 (2008) and 1.4 (2009) provided additional functionality such as constraint checking with the Dresden OCL toolkit and modularization concepts for TGG specifications. In MOFLON 1.4, usability aspects were tackled by providing a basic static analysis and giving warnings and error messages for identified problems. The latest version of MOFLON (1.5 in December 2010) tackled, for the very first time, usability aspects for *end users*. In research cooperations with industrial partners, requirements concerning a clear and supported workflow for the end user were raised. This led to a port of our standalone Tool integration Environment (TiE) [K<sup>+</sup>09] to an Eclipse plugin, which also provided wizards and builders to support end users in setting up and maintaining a MOFLON workspace. Last but not least, some new features for SDM (e.g., reflective concepts) and a redesign of the algorithms used in TiE were implemented.

Two exemplary success stories of MOFLON are:

**MATE:** In cooperation with Daimler, enforcing complex modeling guidelines for Matlab Simulink and Stateflow models was researched. Using MOFLON, a tool adapter was designed that accessed Matlab and manipulated model data directly in the tool. Using this adapter, modeling guidelines could be successfully verified and, in many cases, even automatically repaired [S<sup>+</sup>07]. The guidelines themselves were modeled via model transformations using SDMs, ensuring a high-level specification, free of implementation and platform details.

**MDAE:** In cooperation with Siemens AG, Division Industry Automation, introducing MDSD concepts and technology to existing Automation Engineering workflows was researched. Automation Engineering requires the integration of information from other involved engineering disciplines, that each have their own established tools, workflows and “models”. As a concrete application scenario, the integration between the Location Oriented Structure of a machine plant and its Hardware Configuration was implemented using MOFLON as described in [L<sup>+</sup>10].

### 3 Running Example

In this section, we present our running example. A current project at our department aims at realizing a comprehensive, model-based security engineering process to generate security monitors in software or reconfigurable hardware. In early development phases, specifications are modeled as use and misuse cases via extended Live Sequence Charts (LSCs). Interpreting these expressive behavioral descriptions directly as LSCs is a complex task that can be made tractable via a transformation to Monitor Petri Nets (MPNs), which are easier to interpret [P<sup>+</sup>10]. This LSC to MPN transformation has been realized by metamodeling the DSLs and specifying the transformation via SDMs with the new tool.

Figure 1 depicts simplified metamodels for LSCs and MPNs in Ecore, a simple SDM rule, and models in concrete syntax. These metamodels contain only the basic elements of LSCs for existential charts and MPNs, excluding e.g. failure and end places. The SDM rule transforms all `LSCObjects` located in the LSC model into `InitialPlaces` of the corresponding MPN model.

The SDM rule starts, like UML 2 activity diagrams, with a start node at the top left of the diagram and is called with three parameters: an `LSC` (source model), an `MPN` (target model), and a `Cache` object, used for keeping track of correlations between LSC objects and MPN places. Following the control flow to the *foreach* activity `Find_all_LSC-Objects`, the contained *story pattern*, which can be viewed as a set of constraints, is fulfilled by `LSCObjects` contained in the LSC `lsc`, which is already bound to the parameter value used when invoking the rule. For our concrete models, this first story pattern would *match* all `LSCObjects`– `Car` and `Tollbridge` – in the LSC model. For each match, the activity `Create_InitialPlace` is performed: an `InitialPlace` (`place`) is created and connected to the MPN (`mpn`). Furthermore, a `CacheEntry` (`entry`), connecting the newly created `place` and the corresponding `lsc object`, is created. The new `CacheEntry` is added to the `cache` by creating a new containment link. When all *LSCObjects* have been matched, the rule ends with a stop node.

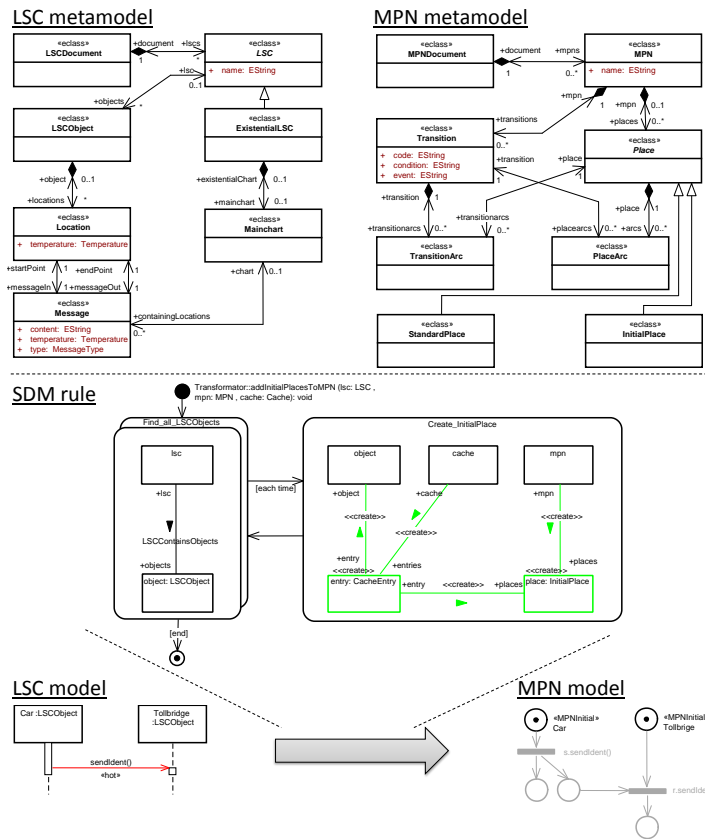


Figure 1: LSC to MPN Transformation specified with SDMs.

## 4 Motivation for Re-Engineering MOFLON

In the following paragraphs, listed in order of importance, we give the main reasons for re-engineering MOFLON and discuss various challenges the old system faced. With the new system eMoflon, we aspire to address these problems and meet a further set of current requirements presented in Section 5.

**Choice of standards:** When the development of MOFLON started in 2002, MOF and JMI appeared to be the clear choice of standards and technology for metamodeling. As there were no complete and stable implementations available, the MOFLON team implemented a MOF/JMI compliant code generator from scratch, and has been maintaining this ever since. With the rise of Eclipse and EMF in recent years, Ecore (which is isomorphic to an *essential* subset of MOF, namely EMOF) has established itself as a *de facto* standard in the metamodeling community. Moreover, EMF provides a stable and well-tested implementation and code generator for Java. In the last few years, it has become increasingly difficult for us as a research group to cooperate with other universities and share meta-models and tools. To the best of our knowledge, no other group actively uses and supports

JMI today. We are not the only group that has faced this challenge in the past, and [GR03] shows that it is indeed possible to build a *bridge* between MOF and Ecore. Such a solution is, however, costly to maintain and was actually used by the authors in practice, to switch completely to Ecore/EMF.

Nonetheless, we could still retain MOF/JMI, but this would be challenging as we would have to implement adapters for every tool to be integrated with MOFLON. This has already forced us to implement, test and maintain *all* components ourselves, including code generation, persistence, basic editors, diff/merge algorithms and our event mechanism. This goes against our philosophy of (re)using mature technology as much as possible and makes it difficult to concentrate on our core competencies and research interests.

**Mature CASE tools and MDSO technology:** The decision to realize MOFLON as a plugin in the FUJABA framework made it possible to develop a meta-CASE tool in a relatively short period of time. In addition to editors and a substantial part of the user interface, it was possible to use FUJABA's model transformation engine CodeGen2, configured with a set of JMI compliant templates. Although MOFLON would not have been possible without FUJABA, the price has been implementing and maintaining an intricate bidirectional adapter layer between MOFLON and FUJABA data repositories. This layer is costly to maintain and has always been a source of bugs [Kön09]. Furthermore, the bidirectional transformation was never re-implemented<sup>3</sup> via TGGs (no bootstrap).

In recent years, Eclipse has established itself as a solid foundation for implementing an IDE. Even in the FUJABA community, Eclipse technologies have already been leveraged successfully in the development of FUJABA4Eclipse [MW04]. Altogether, the Eclipse Plugin Development Environment (PDE), Ecore/EMF and quite a few UML CASE tools have become mature, are well tested and are used by a wide range and a substantial number of engineers. Leveraging existing and established tools simplifies the development, maintenance *and* usage of MOFLON, as stable, well-tested components for code generation, persistence and editor functionality can be reused, and well established usability concepts can be adopted<sup>4</sup>.

**Standardization and unification of our metamodels:** A new metamodel for SDM has been developed [D<sup>+</sup>11] in a joint cooperation with four universities and a company. As Ecore/EMF was a common denominator for all other groups, the metamodel extends Ecore models with modeled behavior for `EOperations` via SDMs. This coordinated effort and cooperation was a further reason for switching to Ecore/EMF as we would otherwise have had to maintain a separate MOF/JMI version and constantly synchronize changes and updates to the metamodel.

## 5 Requirements and Primary Goals

To discuss requirements and primary goals for the new system, we consider two different stake holders: developers and end-users. Users of MOFLON include industrial research partners [L<sup>+</sup>10, S<sup>+</sup>07], as well as academic research partners and students. With

---

<sup>3</sup>As this was part of the TGG implementation, a first version had to be hand-crafted.

<sup>4</sup>Although there exist viable alternatives, Eclipse was a common denominator to facilitate cooperation with other universities and partners.

the following requirements and goals, once more in order of importance, we plan to address the interests of our main stake holders:

**Seamless integration into established engineering workflows:** From our experience with industrial partners, a common requirement is to be able to integrate a new tool or technology in an established workflow. For MOFLON, this implies extending and reusing existing professional (UML) CASE tools already in use in the company, or at least offering a comparable professional and industrial-strength frontend for metamodeling. Important issues like stability, scalability and multi-user support are expected from the frontend and are highly non-trivial to implement from scratch.

A further requirement for integrating MOFLON into established workflows is to be able to invoke all functions via a command-line. This provides the necessary flexibility for scripting, automated usage and integration with other tools and processes.

In our experience, industrial acceptance for a generative approach appears to be greater than for interpretative solutions. An important reason is that generated code can be relatively simply integrated with hand-crafted code and libraries (c.f. Sec. 6.3). There are of course numerous advantages and disadvantages for both approaches which we cannot discuss in detail in this paper.

**Core competencies:** All core developers of MOFLON are PhD students, who each have a specific research topic and a certain field of interest and competence. For this reason, MOFLON must focus primarily on our main research interests, i.e., model transformations via SDMs, and *bidirectional* model-to-model and model-to-text transformations via TGGs. Further current fields of research include static and dynamic *testing* of model transformations and a flexible, *incremental* graph pattern matching engine.

Aspects that do not belong to our current core competencies include concrete syntax development (editors), code generation for standard metamodeling languages, and diff/merge algorithms to name a few. To be able to concentrate on our core competencies, we must constantly try to reuse established, well-tested and stable components (e.g. code generators) whenever this is feasible.

**Longevity:** MOFLON has always aspired to be more than a prototype. This is, in an academic context, quite a challenge as core developers switch every five years. Nonetheless, with an extensive test-suite and a bootstrapping philosophy (building eMoflon with eMoflon), we hope to provide a stable platform for future generations of developers to build upon.

## 6 New Architecture and Design Decisions

In this section, we present our current architecture and discuss various important design decisions we made. Figure 2 gives an overview of the most important components and their dependencies. Using basic UML 2.3 notation, each component is depicted as a rectangle with a clear interface stating (in most cases via a simple graphical symbol) what it provides and what it requires. The stereotype «external» is used and means that the corresponding component was not implemented from scratch but only slightly adapted/configured and reused from an external source.

On the highest abstraction level, eMoflon basically consists of a frontend for metamodel-

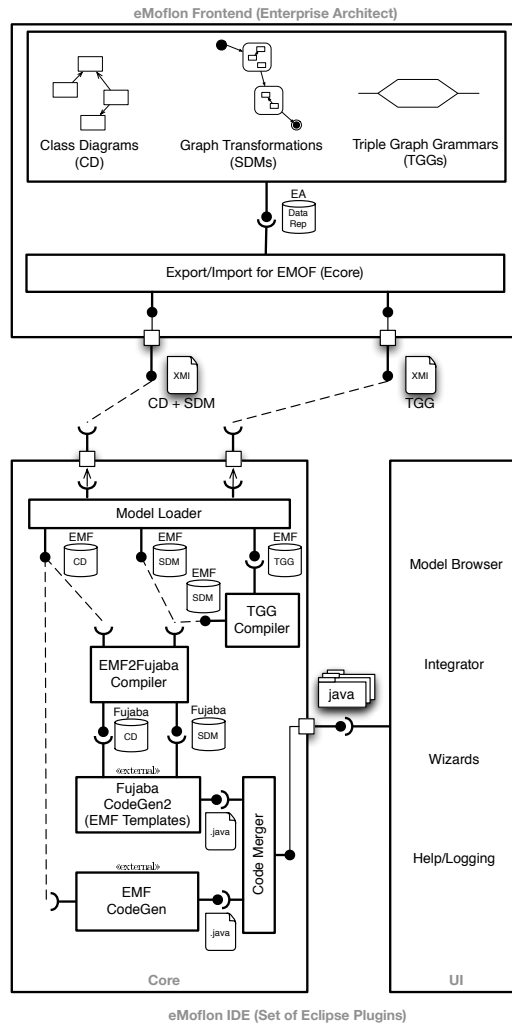


Figure 2: System Overview of eMoflon

eling with a graphical or textual editor, and an IDE for working with the generated code. Section 6.1 describes our frontend component, an *Add-In* for Sparx Systems' Enterprise Architect (EA), and its interface to our Eclipse-based IDE, which is treated in Section 6.2. In Section 6.3, we present a means of explicitly modeling the interaction between generated and hand-crafted code to enable sophisticated static analyses.

### 6.1 Enterprise Architect as Frontend

The frontend component is responsible for providing the user with a means of specifying metamodels via class diagrams, unidirectional model transformations via SDMs,

and bidirectional model transformations via TGGs. Independent of how these diagrams or transformations are displayed and manipulated in concrete syntax, the frontend component must provide a clear interface to the IDE component: Class diagrams, SDMs and TGGs are to be exported as models that conform to Ecore/EMF, our SDM and TGG metamodels, respectively. In each case, all models are persisted as Ecore/EMF conform XML. SDMs are embedded as annotations to methods of classes and are thus exported in the same XMI file as the corresponding class diagram (Fig. 2). A crucial point is that eMoflon has no dependencies on any concrete editors in contrast to MOFLON, which has only an internal editor for SDMs, and that eMoflon can be used, for example, as a command-line operated code generator/compiler.

For the current version of eMoflon, we decided to realize our frontend component as an EA Add-In consisting of two sub-components: an extension of the UML editor and an export/import module. Alternative frontend platforms we considered included GMF, textual Eclipse editors, and other professional UML CASE tools. We decided on EA for the following reasons (in order of importance):

**A professional, affordable industrial-strength UML CASE tool:** In our experience with our industrial partners, a professional frontend is crucial for acceptance. The chances of getting engineers to metamodel using an industrial-strength UML CASE tool such as EA – used by our industrial partners – are much higher than with a newly developed GMF-based editor. EA is furthermore relatively affordable<sup>5</sup> and is thus not that much of a hurdle for students or non-professional users. Implementing editors, although very important, is not one of our core competencies and we chose a UML CASE tool to be able to get as much basic editor functionality as possible with minimal effort.

**Our concrete syntax is traditionally visual:** MOFLON has always supported visual modeling and our established concrete syntax is at the moment almost completely visual, with OCL as the only exception. Although we have plans to provide a textual frontend in the near future, we decided to retain a visual modeling language for the new re-engineered eMoflon.

**A clean separation between diagrams and the data repository:** EA provides a data repository that has basic concepts like elements, connectors and tagged values. EA makes a clear distinction between the visual model (data repository) and the diagrams that a user can specify. Each diagram is simply a view or excerpt of the complete data repository and can contain an arbitrary subset of the elements and connectors (Fig. 3). This separation avoids the typical case of having huge complex diagrams with numerous connections, and can be compared to splitting a textual specification in files and folders. We have already used the possibilities of having multiple diagrams extensively, for example, to separate the activity diagram part of SDMs from individual story patterns (graph transformations) as depicted in Fig. 4. As the decomposition in arbitrary diagrams has absolutely no effect on the data repository, the export/import module remains unaffected.

**Declarative editor extension via UML profiles:** EA can be configured via UML profiles that define a new set of stereotypes, and tool boxes (Fig. 3). We have implemented an Ecore/EMF UML profile that provides stereotypes such as «class» and «reference», to configure the UML class diagrams in EA for Ecore/EMF modeling (Fig. 3). Similarly, SDMs are modeled via UML activity diagrams for control flow, and object diagrams for

---

<sup>5</sup>There is also the possibility of a 30 day trial license.

graph transformations (Fig. 4). TGGs are modeled via object diagrams and stereotyped class diagrams with a hexagon for depicting correspondence types <sup>6</sup>.

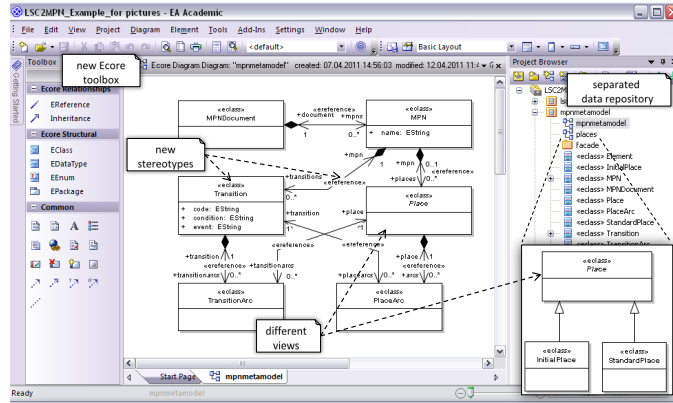


Figure 3: EA eMoflon frontend showing the MPN metamodel

**A scalable database and efficient queries via SQL:** Scalability and robustness are important goals of eMoflon, and EA provides support by basing its data repository on a database that can be queried using SQL. This provides the potential of optimizing the export/import process by determining bottlenecks and gradually switching to SQL, reducing readability when compared to direct API calls, but increasing efficiency substantially. Based on this database, EA provides further functionality such as multi-user support, which would be highly non-trivial to implement for a proprietary editor.

**A generic and flexible data repository:** EA's data repository is so generic and flexible that it can be used to represent models and diagrams that are clearly not UML conform. This is of utmost importance as we must use and combine various UML diagrams to represent TGGs and SDMs and reuse the editor functionality.

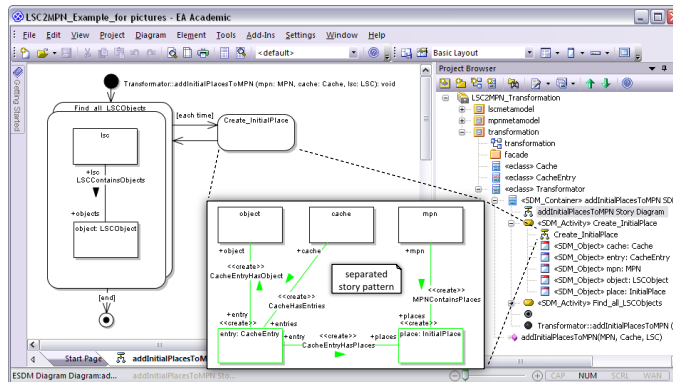


Figure 4: EA eMoflon frontend with SDM rule

<sup>6</sup>Support for TGGs is currently in a beta version and is not discussed in detail in this paper.

Our experience and results with tailoring EA show that it is indeed a viable alternative to GMF/Xtext and that it is clearly possible to satisfactorily adapt a professional UML CASE tool for modeling different kinds of graph transformations. Although this requires in-depth knowledge of the tool and its API, our opinion is that the advantages outweigh any disadvantages.

## 6.2 eMoflon IDE

Our eMoflon IDE consists of a set of Eclipse plugins, which provide an integrated environment for working with generated code from (meta)models specified using the frontend. We support Java code generation via a `Core` set of plugins, and a series of other tasks via a `UI` set of plugins (Fig. 2). The interface between `Core` and `UI` can be viewed as Java projects with EMF conform Java code, which adhere to a certain project structure depending on the type of the project. Currently, `UI` consists of a model browser for model navigation and manipulation, an integrator for visualizing and executing bidirectional transformations generated from TGGs, and a set of Wizards and other subcomponents (Help/Logging). Please note that we refer to Eclipse as a framework for building an IDE (with Builders, Wizards, etc.) and not as the whole Eclipse ecosystem which includes various editor frameworks such as GMF and Xtext.

The heart of eMoflon is implemented in `Core` and provides: EMF conform code generation for (EMOF) class diagrams, code generation for SDMs (implementation of methods in classes), and code generation for TGGs (set of Java methods performing various tasks including bidirectional model transformations and consistency checking). The first step in the code generation process is performed by a `Model Loader`, whose task is to load all models and build up corresponding EMF repositories.

The `TGGCompiler` converts TGG models to a set of SDM models, which are treated in the ensuing process just like any other SDM models. For SDM code generation, we (re)use `CodeGen2` from FUJABA configured with EMF templates [G<sup>+</sup>07]. `CodeGen2` is a model transformation engine and performs, in addition to the actual code generation, various complex tasks including pattern matching. As `CodeGen2` requires input models in a different repository (conform to a FUJABA UML-like metamodel), we convert our EMF repositories as required via a simple batch process that we plan to replace later by SDMs (as part of a bootstrapping process). Our `EMF2FujabaCompiler` that implements this transformation is based on previous work and code from [B<sup>+</sup>08]. For (EMOF) class diagrams, we leverage the standard EMF code generator directly, and complete the process via a `Code Merger` module. This module is responsible for integrating the method implementations generated by `CodeGen2` correctly in the corresponding static EMF code from the standard EMF code generator.

## 6.3 Explicit Interaction between Generated and Hand-Crafted Code

As a preparatory step for an in-depth static analysis of instances of our metamodels (EMOF, SDM, TGG), we have carefully added support for explicitly modeling the layer

between fully generated code and hand-crafted code and libraries. Our approach is inspired by experience from PROGRES [Sch97], a predecessor of FUJABA, that also enforced a clean separation, and could thus offer sophisticated static analyses and fixes. In MOFLON 1.5, the philosophy was to have completely generated code without allowing any manual changes. This is theoretically possible, using SDMs for implementing methods and allowing only a one way communication between a model and, for example, a graphical user interface (GUI) layer. In practice, however, one must provide means for the model layer to communicate with hand-crafted code and libraries. The rather unsatisfactory solution was to introduce “statement nodes” in SDMs, with which one could introduce Java code snippets directly in the model. Introducing such “black boxes” defeats any attempts at providing static analyses for SDMs and, for example, offering fixes to keep all SDMs consistent with the corresponding class diagrams, even after complex changes.

Our solution for eMoflon, is to forbid statement nodes, but allow “collaboration statements”, with which explicitly modeled methods of classes in class diagrams can be invoked. The implementation of these methods need not be modeled via SDMs, but can be hand-written and thus serve as a *facade* for the rest of the hand-crafted code and libraries, which is not present in the model but is part of the complete project. Modeling a facade explicitly, guarantees that methods are invoked correctly and that changes in the class diagrams can be propagated appropriately to all SDMs, even for collaboration statements. Last but not least, the standard EMF code generator already supports merging hand-crafted code and generated interfaces, allowing the generated method signatures for the facade to be merged with the manual implementation and additional methods that are not part of the model. In our opinion, this strategy works quite well, and even has additional advantages including simplified integration with source code management systems (e.g., SVN).

## 7 Related Work

In this section, we give an overview of related tools and approaches, highlighting interesting parallels, differences and similarities. Our selection is by no means complete, but should, nonetheless, put our design decisions and chosen emphasis in perspective, and show that eMoflon offers an interesting and relevant set of features, which is not yet supported by any single tool. For our discussion, we consider three groups of meta-CASE tools: tools that we are conceptually or technologically based on, tools with which we share our current metamodels, and a further group of important metamodeling and model transformation tools, which are, however, not directly related to eMoflon. For each tool, we point out similarities and differences to eMoflon, by considering the following points:

**Does the tool take a holistic approach?** In this context, a holistic approach provides model transformations as an integrated part of metamodels, for example, by embedding the transformations as methods of classes in the metamodel. In a holistic approach, the complete workflow of metamodeling and specifying uni- and bidirectional transformations is supported by the tool and environment. Such holistic tools including FUJABA, UML Lab and MetaEdit+. Tools which provide model transformations that are completely separated from metamodels, according to this definition, do not take a holistic approach and aim at integrating with other existing metamodeling tools.

**Is bidirectionality supported?** An open challenge in the context of model transformations is catering for bidirectionality. Bidirectional transformations are indeed relevant for a multitude of applications that cut across various technologies and communities [C<sup>+</sup>09].

Some tools provide added support for specifying bidirectional transformations by automatically deriving forward and backward transformations from a single specification, automatically deriving a backward / forward transformation from a given forward / backward transformation, or guaranteeing a certain well-behavedness for a given pair of unidirectional transformations. All tools that transform models via TGGs support bidirectionality. Furthermore, UML Lab supports bidirectionality via a template-based approach for parsing and emitting Java source code from and to UML class diagrams.

**Declarative, rule based style:** A declarative specification states *what* is expected as a result and not *how* this should be derived or achieved. To support declarativity, underlying algorithms derive the necessary operational steps to achieve the specified goal. ATL, ETL, Henshin and TGG tools support a declarative means of specifying model transformations.

**Is underlying formalism based on graph grammars?** Some tools like PROGRES, FUJABA, Henshin and Viatra are built upon formal foundations. An advantage of a formal foundation is being able to guarantee certain formal properties. For instance, it might be possible to prove that transformations terminate for all possible input models, or that transformations never produce invalid models. In combination with bidirectionality, one could, for instance, show that derived “forward” transformation rules are indeed inverse to “backward” transformation rules [K<sup>+</sup>10, E<sup>+</sup>07]. eMoflon shares a common, or at least similar formal, foundation with Henshin and TGG tools.

**Core competency of the tool:** Typically, each tool focuses on certain aspects. For example, MetaEdit+ excels in DSL and graphical concrete syntax development. Other tools focus primarily on model transformations and model integration (ATL, Epsilon, TGG tools). Finally, UML Lab and FUJABA concentrate on round trip engineering.

Tool	holistic	bidirectional	declarative	graph grammar formalism	Core Competency
eMoflon	✓	✓	✓	✓	model transformation and integration
PROGRES [Sch97]	✓	×	✓	✓	graph transformation
FUJABA [N <sup>+</sup> 00]	✓	×	✓	✓	UML/Java round-tripping plus SDM
MOFLON 1.5	✓	✓	✓	✓	MOF 2.0 compliant code generation and TGGs
UML Lab [Yat11]	✓	✓	✓ <sup>2</sup>	×	UML/Java round-tripping
TGG Interpreter [K <sup>+</sup> 04]	×	✓	✓	✓	model integration
MDE Lab [GW09]	×	✓	✓	✓	model integration plus SDM
ATL [J <sup>+</sup> 08]	×	×	✓	×	model transformation
Viatra2 [VB07]	×	×	✓	✓	model transformation
Epsilon [Kol08]	×	✓ <sup>1</sup>	✓	×	high-level model manipulation
Henshin [A <sup>+</sup> 10]	×	×	✓	✓	model transformation
MetaEdit+ [TR03]	✓	×	×	×	DSL and graphical concrete syntax development

Table 1: Summary of related approaches and CASE tools

<sup>1</sup>: together with Epsilon Verification Language (EVL), <sup>2</sup>: template-based

Table 1 summarizes all mentioned aspects for eMoflon and a set of related tools. Please note that the chosen points are by no means mandatory for any tool but rather indicate

core competencies, tendencies and philosophy. For example, choosing a holistic approach instead of a clear separation between model transformations and the relevant metamodels is a matter of taste and both approaches have advantages and disadvantages depending on the relevant context and application.

## 8 Summary and Future Work

In this paper we presented eMoflon, a completely re-engineered version of the meta-CASE tool MOFLON. We shared our experience with tailoring a professional UML CASE tool Enterprise Architect (EA) as our frontend, and discussed how we leveraged EMF and Eclipse technologies in our new architecture. Although we made major changes, we retained many design decisions and principles that MOFLON has always been known for including: preferring a generative over an interpretative approach, using CodeGen2 as a stable, well-tested component, and concentrating on *bidirectional* model transformations with TGGs and their mapping to unidirectional transformations.

In the near future, we aim to support substituting the underlying model transformation engine (currently CodeGen2) with alternative engines. We plan to implement various extensions to TGGs and to provide a textual concrete syntax and textual editors as a free alternative to our EA frontend.

## References

- [A<sup>+</sup>06] C. Amelunxen et al. MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. In *Proc. of the 2nd ECMFA*, volume 4066 of LNCS, pages 361–375. Springer, 2006.
- [A<sup>+</sup>10] T. Arendt et al. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In *Proc. of the 13th MODELS*, volume 6394 of LNCS, pages 121–135. Springer, 2010.
- [AK03] C. Atkinson and T. Kuhne. Model-driven development: a metamodeling foundation. *IEEE Software*, 20(5):36–41, 2003.
- [B<sup>+</sup>08] B. Becker et al. Fujaba’s Future in the MDA Jungle - Fully Integrating Fujaba and the Eclipse Modeling Framework? In *Proc. of the 6th Int. Fujaba Days*, 2008.
- [C<sup>+</sup>09] Krzysztof Czarnecki et al. *Bidirectional Transformations: A Cross-Discipline Perspective, GRACE Meeting Notes, State of the Art, and Outlook*, volume 5563 of LNCS, pages 260 – 283. Springer, 2009.
- [D<sup>+</sup>11] M. Detten et al. A new Meta-Model for Story Diagrams. In *Proc. of the 8th Int. Fujaba Days*, 2011. accepted for publication.
- [E<sup>+</sup>07] H. Ehrig et al. Information Preserving Bidirectional Model Transformations. In *Proc. of the 10th FASE*, pages 72–86, 2007.
- [F<sup>+</sup>00] T. Fischer et al. Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In *TAGT '98 Selected Papers*, volume 1764 of LNCS, pages 296–309. Springer, 2000.

- [G<sup>+</sup>07] L. Geiger et al. EMF Code Generation with Fujaba. In *Proc. of the 5th Int. Fujaba Days*, 2007.
- [GR03] A. Gerber and K. Raymond. MOF to EMF: there and back again. In *Proc. of the 2003 OOP-SLA workshop on Eclipse technology eXchange*, Eclipse '03, pages 60–64. ACM, 2003.
- [GW09] H. Giese and R. Wagner. From model transformation to incremental bidirectional model synchronization. *SoSyM*, 8(1), 3 2009.
- [J<sup>+</sup>08] F. Jouault et al. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31 – 39, 2008. Special Issue on 2nd Issue of Experimental Software and Toolkits.
- [K<sup>+</sup>04] E. Kindler et al. An Adaptable TGG Interpreter for In-Memory Model Transformation. In *Proc. of the 2nd Int. Fujaba Days 2004*, 2004.
- [K<sup>+</sup>09] F. Klar et al. TiE - A Tool Integration Environment. In *Proc. of the 5th ECMDA-TW*, volume WP09-09 of *CTIT Workshop Proc.*, pages 39–48, 2009.
- [K<sup>+</sup>10] F. Klar et al. Extended Triple Graph Grammars with Efficient and Compatible Graph Translators. In *Graph Transformations and Model Driven Engineering - Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*, volume 5765 of *LNC3*, pages 141–174. Springer, 2010.
- [Kol08] D. Kolovos. *An Extensible Platform for Specification of Integrated Languages for Model Management*. PhD thesis, Department of Computer Science, University of York, 2008.
- [Kön09] A. Königs. *Model Integration and Transformation - A Triple Graph Grammar-based QVT Implementation*. PhD thesis, TU Darmstadt, 2009. Dissertation.
- [L<sup>+</sup>10] M. Lauder et al. Model-Driven Systems Engineering: State-of-the-Art and Research Challenges. *Bulletin of the Polish Academy of Sciences, Technical Sciences*, 58(3):409–422, 2010.
- [MW04] M. Meyer and L. Wendehals. Teaching Object-Oriented Concepts with Eclipse. In *Proc. of the Eclipse Technology eXchange Workshop (ETX)*, 2004.
- [N<sup>+</sup>00] U. A. Nickel et al. Tool demonstration: The FUJABA environment. In *Proc. of the 22nd ICSE*, 2000.
- [P<sup>+</sup>10] L. Patzina et al. Monitor Petri Nets for Security Monitoring. In *Proc. of the Int. S&D4RCES workshop*, 2010.
- [S<sup>+</sup>07] I. Stürmer et al. Enhanced Simulink/Stateflow Model Transformation: The MATE Approach. *Proc. of the MAC 2007*, 2007.
- [Sch97] A. Schürr. Programmed Graph Replacement Systems. In G. Rozenberg, editor, *Handbook of graph grammars and computing by graph transformation: vol. 1: foundations*, volume 1, pages 479–546. World Scientific, 1997.
- [TR03] J.-P. Tolvanen and M. Rossi. MetaEdit+: defining and using domain-specific modeling languages and code generators. In *Proc. of the 18th ACM SIGPLAN OOPSLA*, pages 92–93. ACM, 2003.
- [VB07] D. Varró and A. Balogh. The model transformation language of the VIATRA2 framework. *Science of Computer Programming*, 68(3):214 – 234, 2007. Special Issue on Model Transformation.
- [Yat11] Yatta Solutions GmbH. UML Lab. <http://www.uml-lab.com/>, April 2011.