Bridging Microsoft Oslo and Eclipse EMF

Technical Report

Stanley Hillner and Heiko Kern
Business Information Systems, University of Leipzig
Johannisgasse 26, 04103 Leipzig, Germany
{hillner, kern}@informatik.uni-leipzig.de

Abstract: In the ever-growing approach of Model-Driven Software Development, there is already a large number of modeling tools and frameworks. From this variety of tools, the result is a strong heterogeneity which makes the reuse of models and metamodels to a challenge. This paper introduces an approach for bridging modeling tools using the M3-level based bridging pattern for enabling (meta)model interchange. Therefore, two sample frameworks are presented and analyzed. These frameworks are Microsoft's latest modeling platform, which is codenamed "Oslo", and the Eclipse Modeling Framework (EMF). Finally, a possible bridge between Oslo and EMF is described, which is based on the M3-level based bridging pattern.

1 Introduction

Model-Driven Software Development (MDSD) is a software development approach which is based on the consequent usage of models during the whole life cycle of a software system. Short, models are conceptual views of a real world problem, e. g. a software system or parts of it. Now, the key goal of the MDSD paradigm is to push models into the role of enablers instead of being supporters. This means that models are intended to be the key concept during the development, maintenance and operational process. Fulfilling this role requires powerful generators, transformation tools and other model operations. This requirement often results in the usage of multiple tools and frameworks which often scope different technological spaces that typically are inoperable. Therefore, transformations take a special role during the development process. There is a variety of model transformations that scope different needs during the development. For instance, model transformations can base on one technological space or different ones. A detailed distinction is given in Mens et al. [MCVG05] as a taxonomy of model transformations.

This paper scopes the M3-level based bridging of two different frameworks for Model-Driven Software Development (MDSD), Microsoft Oslo and the Eclipse Modeling Framework (EMF). Oslo is Microsoft's latest modeling platform which is still under construction[1]. Since Oslo is developed by Microsoft, this framework is based on the .NET platform[2]. In contrast to Oslo, EMF is a mature stable modeling platform for Java developers which has a large community and performant tools for modeling and further work on models. Regarding to the previously named

[1] The project has still moved to the SQL Server project and is named SQL Server Modeling Services. Nevertheless, the project is still in CTP status (Community Technology Preview).

[2] http://www.microsoft.com/NET

taxonomy of model transformations of Mens et al. [MCVG05], this bridge would be a exogenous, horizontal bridge between different technological spaces.

The goal and outline of this paper is to give a short introduction to the M3-level based bridging pattern, including the key concept and the steps that have to be performed. After that, both frameworks will be introduced and their functionality will be investigated regarding to similarities and differences between both frameworks. Also possible benefits of bridging these frameworks will be highlighted. Furthermore, the internally used concepts of both frameworks will be examined to find a suitable approach of bridging Oslo and EMF. The last section of this paper introduces a possible bridging architecture that is based on the M3-level based bridging pattern and offers some alternative approaches for bridging both frameworks.

2 M3-Level-Based Bridges

During the last few years, the concept of M3-level-based bridges (M3B) has extensively been discussed in several contexts. Furthermore, this approach has successfully been applied for building a variety of bridges between different technological spaces, such as the ARIS[3]-EMF[4] bridge [KK07], the MetaEdit[5]-EMF bridge [Ker08], the Visio[6]-EMF bridge [KK09] or the XML-EMF bridge [SBPM09][7]. Even though it has been discussed many times, a brief summary of the intention of this pattern follows up, because of its importance for this paper.

As the name M3-level-based bridge already mentions, this approach deals with a metamodel hierarchy [Küh06], especially its third layer. This hierarchy consists of three levels, starting with the level of models (M1-level) which can contain data instances or even describe a whole software system. The M2-level contains metamodels which define instantiable concepts and structural constraints for models at M1-level. Each M1-model conforms to an associated metamodel whose structure and concepts are also described at a higher level, the metametamodel at M3-level.

Relying on the existence of this metamodel hierarchy, the M3B pattern bridges two technological spaces by defining a mapping between the metametamodels of both tools at M3-level [KK07, Ker08, KK09]. This mapping covers both, the structure of the source tool space and the structure of the target tool space, respectively theirs metamodels. Therefore, the M3-level mapping contains several functions or rules that identify semantic relations between the concepts. Regarding to the used metamodel hierarchy [Küh06], these concepts at M3-level constrain the usable relations between concepts at M2-level.

After the M3-level mapping is defined, the next step is to derive both, the M2-level transformation and the M1-level transformation from this mapping definition. Both mappings (M2-level and M1-level), which result from the transformations, own the characteristics of isomorphisms [KK09] which ensure the reversibility of each transformation. Figure 1 visualizes the idea of the M3B pattern.

[3] http://www.ids-scheer.com/en/ARIS
[4] http://www.eclipse.org/emf
[5] http://www.metacase.com/mep/
[6] http://office.microsoft.com/en-us/visio
[7] Further bridges, which are based on the M3B pattern, are named in the related work section.

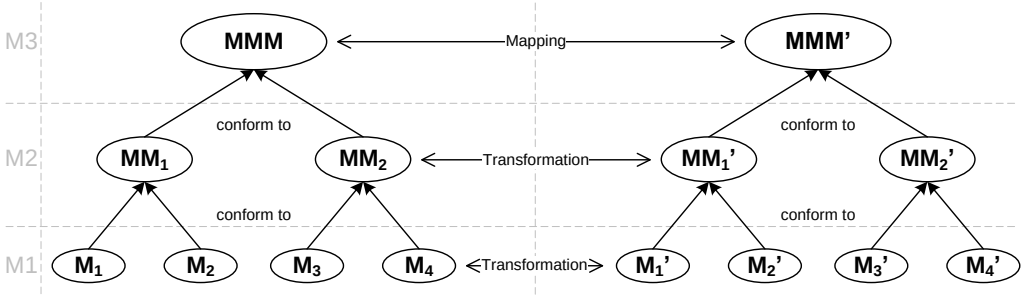

Figure 1: The M3B pattern

3 Microsoft Oslo

The goal of Microsoft's latest modeling platform is to provide a central role to the models during the whole life cycle of a software system. This means that models shall become enablers during the design, development, deployment and management of the software, instead of giving just a static overview of the system. This matches the common goals of the Model-Driven Software Development [SVB+06] approach. Therefore, Microsoft has put the ability for Model-Driven Development (MDD) directly into the .NET platform and offers both, textual and visual modeling tools to easily create and use models over the whole process of development [Mic09a]. Next, the three components of the Oslo platform are explained in brief.

The language "M" is a generic modeling language that offers the ability to capture domains by textually modeling them. "M" is not only a language but rather a family of languages including MSchema, MGrammar and MGraph. In section 3.1 modeling in "M" is explained in detail. Closely related to "M", the Oslo SDK provides an editor, named "Intellipad", a feature rich editor which is designed to model all parts of "M".

"Quadrant" is a customizable visual tool for viewing and editing data in the context of the domain. For that purpose Quadrant offers several possibilities for visualizing data from multiple repositories. One of these are the workpads which can give an overview over different databases or even show the same database from different points of view. Basically, Quadrant offers four main view types (list views, table views, tree views and property views), but the user is free to create any combination of them for getting the most usable interface.

The Oslo Repository serves as a central hub for managing Domain Specific Languages (DSLs) and models across the whole enterprise. The repository is built against Microsoft SQL Server 2008[8], so that it can take advantage of its features for ensuring robustness, performance and scalability. Furthermore, the repository contains a Base Domain Library (BDL) which provides an infrastructure and services which assist during the building of models and model-driven applications.

[8] http://www.microsoft.com/sqlserver

3.1 Modeling in Oslo (the "M" language)

As mentioned before, modeling in Oslo is mainly based on the "M" language family which consists of three parts, MSchema, MGrammar and MGraph. Each of the three parts of "M" offers a set of features that can be used to develop models and metamodels, respectively Domain Specific Languages. In the following, all three language components are explained by an example.

MSchema is the language for defining data structures in Oslo. Within the metamodel hierarchy [Küh06], as described in section 2, schemas created using MSchema are metamodels at M2-level. Using "M", these metamodels are created textually by the developer. Listing 1 shows a sample for a simple schema which describes a music library. In "M" each model or schema exists within modules, so the schema definition starts with a module declaration. After that, the schema defines a type, named MusicItem, which encapsulates several information about the music items of the library. Each music item has an id of the built in type `Integer64` and will be automatically created by the repository. This id is defined as the identifier for a music item at the end of its declaration by the string `where identity Id`. Furthermore, a music item consists of the album title and the artists name, both of type Text, as well as a rating for this item. This rating is of type `Integer32` and is constrained to be smaller than four. At the end of the schema definition, the `MusicLibrary` item is declared. The music library can contain an unlimited number of music items.

```
1  module Oslo.EMF.Sample {
2    type MusicItem {
3      Id : Integer64 => AutoNumber();
4      Album : Text;
5      Artist : Text;
6      Rating : Integer32 where value <= 3;
7    } where identity Id;
8
9    MusicLibrary : MusicItem*;
10 }
```

Listing 1: Schema definition for a simple music library (from [Pur08])

Populating the schema definition to the repository is a two step process. First, the schema has to be compiled using the "M" compiler (m.exe). This creates an image of the schema as an OPC[9] file which contains the semantic graph of the metamodel, described using XML[10] [Pur08]. Second, the metamodel has to be executed with the "M" execution tool (mx.exe). After executing the metamodel or schema definition, the repository contains the new schema which now can be filled with data instances using MGraph.

MGraph can be used to create data instances which can be loaded into the repository. These data can be validated against the schema before populating them to the repository. Listing 2 shows a sample MGraph data declaration which contains two music items that conform the music library schema from listing 1. This listing defines the container (`MusicLibrary`)

[9] OPC (OLE for Process Control) – http://www.opcfoundation.org/
[10] http://www.w3.org/standards/xml/

that is supposed to contain the data instances. The container is defined within the sample module again. After that, two data instances are created within curly brackets separated by a comma. In the metamodel hierarchy [Küh06], the data graph which is defined by this listing, represents a model at M1-level which conforms to the given metamodel (the music library schema) at M2-level.

```
module Oslo.EMF.Sample {
  MusicLibrary {
    {
      Album => "...Baby One More Time",
      Artist => "Britney Spears",
      Rating => 3
    },
    {
      Album => "Leave Britney Alone",
      Artist => "Chris Crocker",
      Rating => 1
    }
  }
}
```

Listing 2: Definition of Data for the music library sample [Pur08]

Populating this model to the repository leads to a two-step process again. First, the model has to be compiled using the "M" compiler. During the compilation process the model can be validated against the schema for ensuring its correctness before populating the data instances to the repository using the "M" execution tool in the second step.

MGrammar Another way of defining models is the usage of a (textual or graphical) DSL for retrieving structured data from it. MGrammar offers the possibility to define textual DSLs and parsing them into structured data, matching a specific schema. Listing 3 shows a grammar definition for a simple DSL which allows statements that match the pattern: "*AlbumName*" by "*ArtistName*" is "*Ranking*".|!. From each statement a data instance can be retrieved, to fill the model. As an example, one could express the second data instance of listing 2 with the statement «*"Leave Britney Alone" by "Chris Crocker" is terrible!*».

Each grammar definition consists of at least one syntax rule (`Main`). In this case, the main syntax rule allows 1..n statements which will be put into a music library. Each statement represents a music item, which is defined by the title of the album, the artists name and a custom rating, as enumerated in the schema definition. Both, the album title and the artists name are text literals which are separated by the fill word "by" and followed by the fill word "is". Finally, a rating follows up. The rating itself is a token whose values are defined below the statements syntax rule as a list of possible tokens. A third syntax rule assigns rating values that match the type `Integer32` to the previously defined tokens, so that the set of tokens can be used to retrieve ratings for the model. The statement rule does not only constrain allowed expressions, it also assigns values to the data instances of the model which will be created from the DSL.

```
module Oslo.EMF.Sample {
  import Language;

  language MusicLibraryLanguage {
    syntax Main = s:Statement+ => MusicLibrary{valuesof(s)};
    syntax Statement = al:Grammar.TextLiteral "by" ar:Grammar.
        TextLiteral "is" rt:Rating ("." | "!") => {Album{al},
        Artist{ar}, Rating{rt}};

    token Rating1 = "terrible" | "awful";
    token Rating2 = "so so";
    token Rating3 = "awesome";

    syntax Rating = Rating1 =>1 | Rating2 =>2 | Rating3 =>3;
    interleave skippable = Base.Whitespace;
  }
}
```

Listing 3: Grammar definition for a simple music library DSL [Pur08]

For populating the model which is written in the music library DSL. the grammar has to be compiled using the "M" compiler. The next step is to run the textual model against the compiled grammar to retrieve a model described in MGraph like the one of listing 1. The further population process equals the one, which was explained in the section of MGraph.

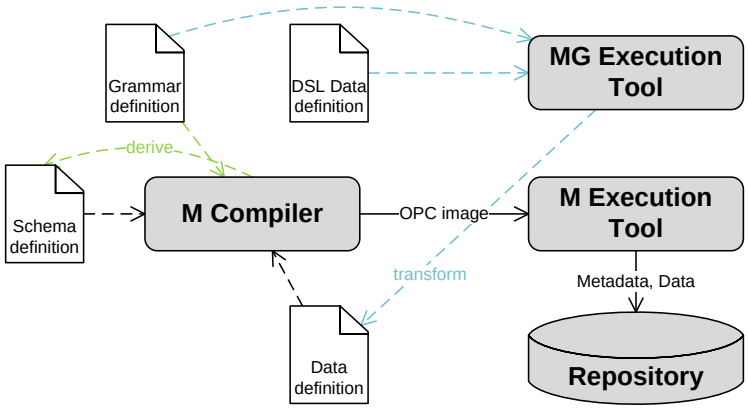

Figure 2: The modeling process in Oslo

Figure 2 gives an overview of the modeling and population process, using the "M" language. Another option of modeling in Oslo is to use the visual modeling tool Quadrant. As already mentioned Quadrant offers a variety of basic views and the possibility for defining custom views. For instance, the user could define a view where process chains can be visualized as Event-Driven Precess Chains (EPC) [SKN92] or using the Business Process Modeling Notation (BPMN) [Whi04]. For the majority of users, this would be the first idea of modeling a domain and would surely be more

interesting than defining and using a textual language, but for this paper, respectively the targeted bridge, only the abstract syntax is of interest. This syntax can be derived from the "M" language specification [Mic09b] which will be discussed later in section 5.

4 The Eclipse Modeling Framework

The key goal of EMF is to offer an infrastructure for building applications model-driven, which matches the key goal of Oslo. Also in the way of modeling, data storage and visualization, equivalent technologies can be found within EMF's technological space. For instance, the CDO Model Repository[11] can be considered as an equivalent to Oslo's Repository or graphical editors, which are based on the Graphical Modeling Framework (GMF)[12] are equivalents to custom views in "Quadrant". Therefore, a bridge between Oslo and EMF seems to be useless. But two possible benefits of bridging these two frameworks would be the following.

1. Since EMF becomes more and more a central hub for bridging modeling tools and frameworks, building such a bridge would pay off at the latest when the need for bridges to other tools arises. There are already many bridges between a variety of modeling tools and the Eclipse Modeling Framework [BBC+05, BHJ+05, KK07, Ker08, KK09] which can be used to build transitive bridges between Oslo and any other Tool or framework, using EMF as a hub. In example, building bridges between Oslo, EMF, Visio, MetaEdit+ and Aris would result in implementing ten bridges (Oslo-EMF, Oslo-Visio, ..., Visio-MetaEdit+, Visio-Aris, ...). Using EMF as a central hub for bridging would only require the building of four bridges (Oslo-EMF, Visio-EMF, MetaEdit+-EMF and ARIS-EMF), achieving the same effect.
2. EMF is a stable and powerful framework for model-driven development which arised in 2002. This Framework has a large community, which is continuously developing a large number of feature rich tools for special tasks of MDSD. For instance, there are several tools for model comparison, difference calculation and model merging[13], as well as frameworks for model transformation, validity checks or code generation[14], which use state of the art techniques and provide excellent support by their developers. Remembering the objectives of the MDSD approach, reimplementing such tools would result in a loss of performance as well as redundancy and a possible loss of quality. Compared with this, exchanging models between the Oslo tool space and EMF's tool space would grant the compliance with the goals of MDSD.

Although having highlighted Oslo and some possible benefits of bridging Oslo and EMF, we neither know anything about how to model in EMF nor anything about its components. The next few lines will introduce the three basic components of EMF [SBPM09] and section 4.1 will give an introduction to the modeling process in EMF.

[11] CDO (Connected Data Objects) – http://wiki.eclipse.org/CDO
[12] http://www.eclipse.org/modeling/gmf/
[13] e. g. http://www.eclipse.org/modeling/emft/?project=compare#compare
[14] e. g. http://www.openarchitectureware.org/

The EMF Core Framework contains EMF's metamodel, named Ecore. A simplified subset of Ecore is shown in figure 3. Since EMF implements the OMG's[15] MOF [Obj09] specification, it inherits its self-description feature. This means that constructs, which are available in Ecore, are defined in Ecore. Ecore is used to describe metamodels at M2-level which conform to the metametamodel Ecore at M3-level of the metamodel hierarchy. These metamodels typically describe the abstract syntax for a DSL or data structures of an application. The core framework also includes change notification and persistence support as well as an efficient reflection Application Programming Interface (API) for manipulating EMF objects.

EMF.Edit is EMF's framework for building model editors. Therefore, it includes generic reusable classes, such as label or content provider classes and other classes which allow EMF to display models using standard JFace[16] viewers.

EMF.Codegen is the part of the Eclipse Modeling Framework which offers possibilities for generating code or other artifacts from EMF models. This Framework includes a Graphical User Interface (GUI) for specifying generation options or invoking generators. Furthermore, EMF.Codegen uses the Java Development Tooling (JDT) component of Eclipse for generation.

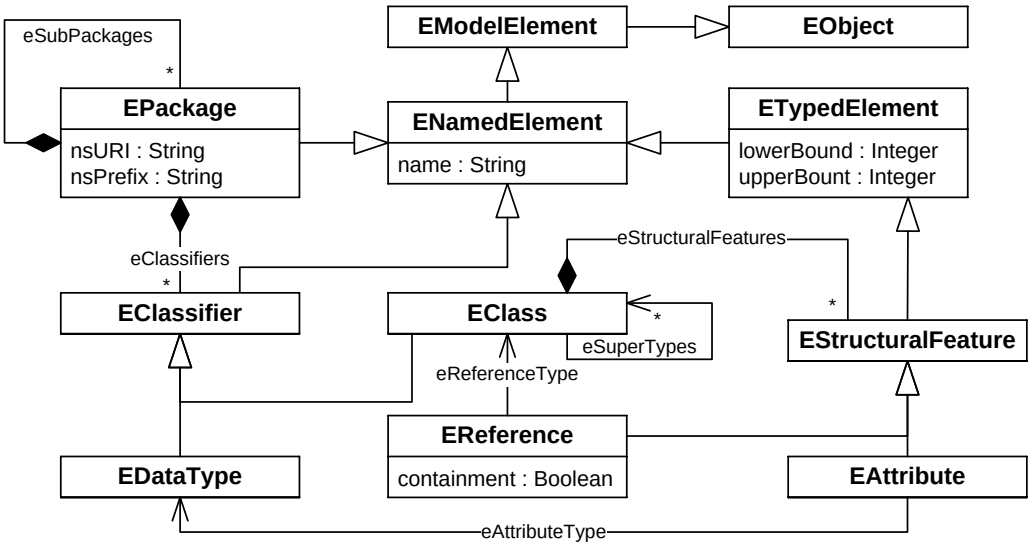

Figure 3: A simplified subset of EMF's metamodel Ecore

[15] OMG (Object Management Group) – http://www.omg.org/
[16] http://www.eclipse.org/swt/

4.1 Modeling in EMF

Equivalent to section 3.1 (Modeling in "M"), this section will outline the process of modeling with the Eclipse Modeling Framework. Therefore, this section is oriented at the modeling concepts of "M" with the intention of showing up equivalences.

Below, three main parts of modeling are described using some examples. These three steps are the definition of metamodels in EMF, the definition of data instances, respectively models and the definition of textual and graphical DSLs.

Metamodel definition in EMF is the equivalent to schema definition with MSchema. In EMF, every metamodel is an instance of the metametamodel Ecore, which defines the usable concepts for metamodeling. In comparison to schema definition in "M", the metamodeling process in EMF is simplified by the built in Ecore model editor. Furthermore, it is possible to derive EMF metamodels from XML Schema Definitions (XSD)[17] or annotated Java. Figure 4 shows the MusicLibrary example from listing 1 as an Ecore metamodel which was created with the Ecore model editor. This tree-based editor shows the standard representation of models and metamodels in EMF which can be generated by the EMF.Edit framework for each custom metamodel. Listing 4 contains the underlying XMI[18] data format which consists of the concepts that are visualized by figure 3. The EMF equivalent to Oslo's `modules` are `EPackages` which define a space of existence for other concepts. Concepts, which can be children of `EPackages`, are `EClassifiers`, such as `EPackages` or `EClasses`. An `EClass` can be seen as an equivalent to Oslo's `type` definition and can contain `EAttributes` as well as `EReferences`, which define references to other concepts. For `EAttributes`, a number of data types exist within the Ecore metametamodel, which can be mapped onto Oslo's built in data types. For instance, `EString` could be mapped to `Text` and `EInt` to `Integer8`, `Integer16`, `Integer32` or `Integer64` in Oslo. A first mapping approach between Oslo and EMF is described in section 5.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"...>
  <eClassifiers xsi:type="ecore:EClass" name="MusicLibrary">
    <eStructuralFeatures xsi:type="ecore:EReference" name="
        MusicItems" upperBound="-1"
        eType="#//MusicItem" containment="true" eKeys="#//MusicItem
            /Id"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="MusicItem">
    ...
  <eClassifiers xsi:type="ecore:EEnum" name="Rating">
    <eLiterals name="terrible" value="1"/>
    ...
</ecore:EPackage>
```

Listing 4: MusicLibrary.ecore (XMI representation)

[17] http://www.w3.org/XML/Schema

[18] XMI (XML Metadata Interchange) – http://www.omg.org/technology/documents/formal/xmi.htm

Before creating model instances of the MusicLibrary metamodel, the next step is to generate an EMF generator model (Genmodel) from the Ecore metamodel. This generator model uses the EMF.Codegen and EMF.Edit framework for generating model code, test code and a standard tree-based editor from the metamodel.

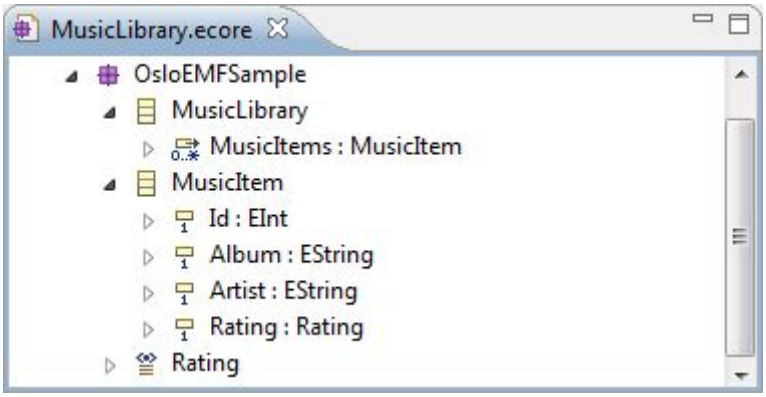

Figure 4: MusicLibrary.ecore (Ecore model editor)

Model creation can be done in different ways. First, EMF offers the possibility to create dynamic model instances of the metamodel without generating code or an editor. This feature uses Dynamic EMF [SBPM09] to interpret the metamodel[19].

Second, deriving the Genmodel from the metamodel offers the possibility to generate Java classes which represent the data structure that is determined by the metamodel. For instance, these classes can be used as the model for an application.

Third, depending on the generated Java classes, one can generate a standard editor[20] for the DSL, which is defined by the metamodel. This can also be done by the Genmodel. Modeling the instance data using this editor equals the modeling of the metamodel using the Ecore model editor. The resulting model can then be accessed during runtime under utilization of the EMF Reflection API, so that the underlying XMI serialization only serves as an exchange format for models and metamodels. Figure 5 shows the generated editor for the MusicLibrary example.

DSLs There are many different possibilities to define concrete syntaxes for EMF metamodels. For instance, these syntaxes may be graphical ones which can be generated with the Graphical Modeling Framework or the Graphical Editing Framework (GEF)[21]. Another type of concrete syntaxes for EMF metamodels are standard tree- or table-based editors which can be generated by the EMF.Edit framework. An example for such an editor is the Ecore model editor, which was shown in figure 4. The last type of concrete syntaxes for EMF metamodels which is mentioned here are textual DSLs. There are several frameworks for

[19] Using Dynamic EMF one can also build EMF metamodels during runtime.

[20] It is also possible to generate custom editors from the metamodel. For instance, you can use GMF to build a graphical editor for the DSL that is defined by the metamodel.

[21] http://www.eclipse.org/gef/

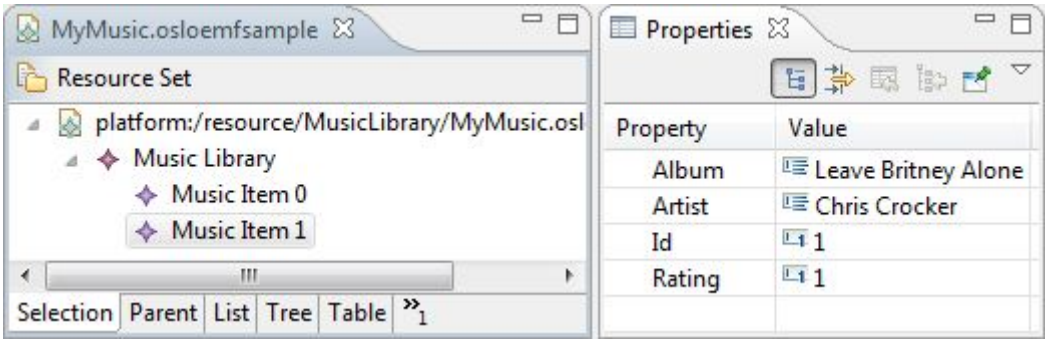

Figure 5: MyMusic (generated tree-based editor and properties view)

creating textual DSLs for EMF metamodels. One of these is the Textual Modeling Framework (TMF)[22]. Similar to the syntax definition in Oslo, one have to define a grammar for the target DSL in EMF. For instance, this can be done with XText[23] which is part of TMF. A grammar definition, which equals the MusicLibrary grammar created with MGrammar (listing 3), is shown in listing 5. This grammar describes the MusicLibrary DSL, which accepts statements that match the pattern: "*AlbumName*" by "*ArtistName*" is "*Ranking*".

```
1  grammar oslo.emf.musiclibrary.Dsl with org.eclipse.xtext.common.
       Terminals
2  import "platform:/resource/Oslo-EMF/model/MusicLibrary.ecore"
3
4  MusicLibrary :
5          (MusicItems+=MusicItem)*;
6
7  MusicItem :
8          Album=STRING "by" Artist=STRING "is" Rating=Rating
               ("!"|".");
9
10 enum Rating :
11         terrible|awful="1" |
12         so_so="2" |
13         awesome="3";
```

Listing 5: XText grammar for MusicLibrary DSL

Once the grammar for the DSL is defined, one has to start an openArchitectureWare (OAW) workflow which generates a text editor for modeling in the defined DSL. In contradiction to Oslo, this editor natively supports syntax highlighting and code completion for modeled concepts. A similarity to the DSLs in Oslo is the parsing of DSL statements into the underlying structured format (EMF models) and features like error highlighting. For this

[22] http://www.eclipse.org/modeling/tmf/
[23] http://www.eclipse.org/Xtext/

purpose an XText grammar is connected to an existing metamodel[24], so that instances of this metamodel can be created by the editor.

5 Bridging Oslo and EMF

This section introduces a possible approach for bridging Microsoft Oslo and the Eclipse Modeling Framework. This approach uses the M3-level based bridging pattern whose idea has been explained in section 2. An overview of a possible architecture for a bridge between Oslo and EMF is given by figure 6. This bridge scopes the parts MSchema and MGraph of the "M" modeling language and leaves out the MGrammar specification. Below, a short introduction to the bridge is given.

Applying the M3-level based bridging pattern requires the existence of a metamodel hierarchy conform structure at each side of the bridge. EMF explicitly offers this structure with Ecore being the metametamodel. Furthermore, Ecore conform metamodels are located at M2-level and the M1-level is represented by models that conform to metamodels at M2-level. At the side of the Oslo modeling framework, this structure is not so obvious. Oslo offers a metametamodel that determines all three parts of the "M" modeling language, MSchema, MGraph and MGrammar. This metametamodel is given implicitly in the framework and is described by the Oslo modeling language specification [Mic09b]. Within this paper, the Oslo metametamodel is named "O3". Since this bridge leaves out the MGrammar part of the "M" modeling language, this part is not visualized in figure 6. Metamodels in Oslo are described under utilization of MSchema. For this reason, schemas which are instances of MSchema and conform to O3 are located at M2-level of the metamodel hierarchy. Finally, models at M1-level are instances of MGraph and must conform to a given schema.

Since the visualized bridging approach uses the M3B pattern, it requires a mapping between both metametamodels at M3-level. It would be sufficient to realize an unidirectional mapping between both metametamodels, for instance on the side of EMF. This would save much conceptional work but would restrict the user to work at the technological space of EMF. Depending on this mapping, a transformation of schemas of the Oslo modeling framework can be done generically. Therefore, a metamodel in EMF must contain a generic structure for schemas created with MSchema. Figure 6 visualizes this by the composed EMF metamodel which consists of an MSchema part and the original schema definition connected via a generalization relationship. The last part of the bridge is the transformation of models which are instances of MGraph. These models will be transformed based on the M3-level mapping and the M2-level transformation. These dependencies to both higher levels of the bridge ensure the conformance of the transformed models to the transformed metamodels. Both transformations, at M2-level and M1-level, are bidirectional so that an interchange of models and metamodels between Oslo and EMF can be done in any direction.

The next sections explain the bridge more in detail, especially the transformations of metamodels and models. An implementation of the bridge is not part of this paper, so that a use case can only be handled theoretically.

[24] There is also the possibility to generate a metamodel from the grammar definition during the editor creation process.

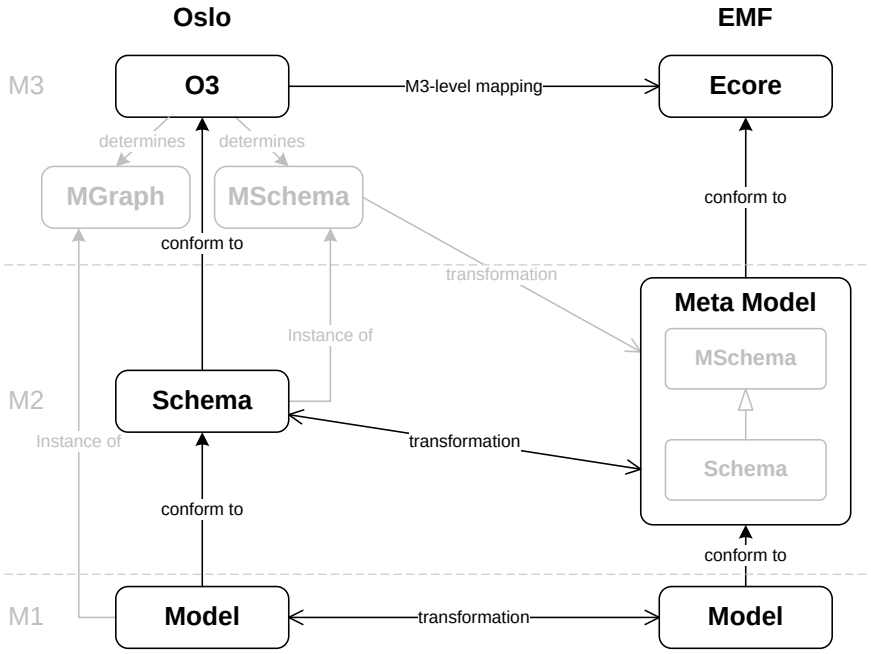

Figure 6: M3-level based bridging of Oslo and EMF

5.1 M3-Level Mapping

This section describes the M3-level mapping of both modeling frameworks. The M3-level based bridging pattern requires this mapping for offering the possibility to build bridges generically. This means that once, the M3-level mapping is done, any metamodel at M2-level can be transformed between both frameworks without reimplementing or configuring the bridge.

Figure 7 shows a sample mapping between the MSchema part of the "M" language and the EMF metametamodel Ecore. This mapping consists of an EMF metamodel for the MSchema definition, which later can be used as a part of the M2-level transformation. In the following the metamodel is explained more in detail.

The metamodel consists of a package with name `M`. This package can contain all concepts of the "M" language, but is restricted to the MSchema part in this paper. As stated in section 3, each schema definition in Oslo exists within a module. So the metamodel contains an EClass named `Module`. This EClass inherits from `IdentifiedElement`, which offers qualifiers and an identifier which conforms to the "M" language specification. Furthermore, a module can contain any number of module members, which can be exported to use within other modules, and imports of members of other modules. Each module exists within one `CompilationUnit` and each `CompilationUnit` must contain at least one module. An `ImportDirective` can import any number of modules or members of modules. These are represented by The EClasses named `ImportModule` and `ImportMember` which inherit from `AliasElement`. This is necessary becau-

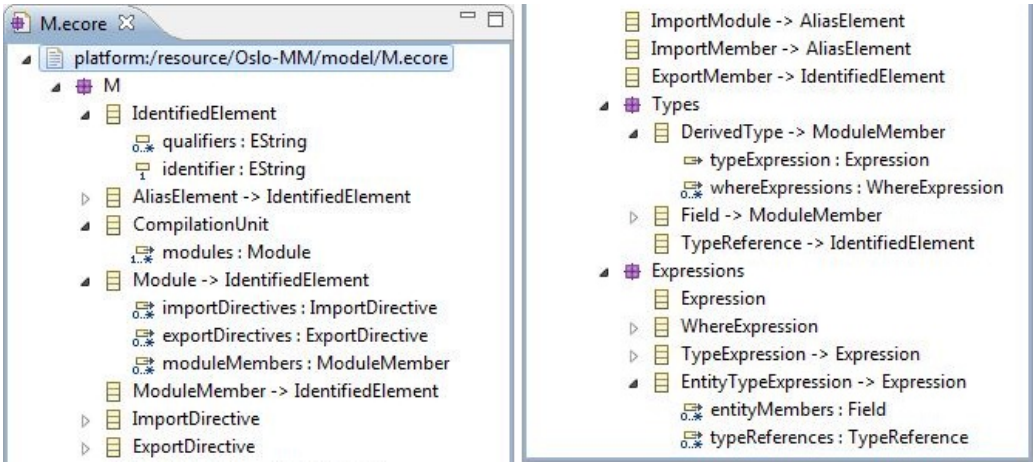

Figure 7: M3-Level mapping of MSchema (Ecore metamodel)

se of the alias feature of "M" which offers the possibility to assign an alternative name to an imported module or module member for the usage within the module. This feature works similar to the namespace aliases in C# or C++. `ExportDirectives` only contain `ExportMembers`, because only module members are able to be exported not the whole module. In contrast to `ImportMembers`, an `ExportMember` only inherits from `IdentifiedElement` to give an export an id but no alias. `ModuleMembers` are the main part of a module. These members can be intrinsic types, which are natively built into the "M" language, `Fields` for data field definition or `DerivedTypes` which can extend an intrinsic type or define a whole new type. A `DerivedType` can contain several expressions, for instance an initialization expression or an `EntityTypeExpression`, to define the data type as a set of elements. For instance, the initialization expression can be an enumeration of elements that are allowed for this type or a restriction of another data type using constraints. Furthermore, a `DerivedType` can contain `WhereExpressions` which are used as constraints on the data type. An example for such a constraint is the identity expression which is used in the schema definition in section 3. Another where expression could restrict an integer value within a type to be less than ten. The last mapped concept, that is explained here, is the `EntityTypeExpression`. Such an expression is responsible for defining entities like the `MusicItems` of the schema definition in section 3. An `EntityTypeExpression` can contain `Fields` which define variables of the entity and `TypeReferences` which are responsible for associating the type with another data type.

This mapping is based on the "M" language specification [Mic09b] and shall be understood as a sample mapping between both M3-levels. Because of the complexity of both metametamodels, some concepts are simplified. For instance, the `WhereExpression`-EClass in this sample only contains a list of queries on the type as strings, where the "M" language specification uses `QueryExpressions` and several other concepts which would raise the complexity of the metamodel.

5.2 M2-Level Transformation

As stated in the introduction of the bridging approach, the M2-level transformation is based on the mapping of both metametamodels at M3-level. The M2-level transformation is not only a horizontal transformation, but also a vertical transformation between M3-level and M2-level. This is the result of the composed EMF metamodel which contains both, the schema definition which is described in MSchema and the MSchema metametamodel from the "M" language specification. This metamodel will contain an abstract structure for the MSchema metametamodel and a concrete class structure for the schema modeled in Oslo. These concrete classes inherit from the abstract MSchema definition. Since this metamodel contains the abstract MSchema definition, it can be used for any schema which can be modeled in Oslo, without changing the bridge. This avoids the need for bridging the gap between both technological spaces again, when having to transform other metamodels. But there is also a disadvantage. The metamodel, which results from the transformation, is only a generic one which contains the whole MSchema definition (or a part of it). Generating code from this metamodel will result in inefficient code structures which are miles avay from the original metamodel of the opposite framework. For instance, at the MusicLibrary example the user would expect a metamodel that looks like the one which was shown in figure 4. But applying this bridging approach to the schema (see listing 1) would result in a metamodel which contains the MSchema definition as abstract EClasses (similar to the mapping from figure 7) and the schema part consisting of concrete EClasses which inherit from the abstract MSchema structure. For instance, the MusicLibrary and the MusicItem EClass would inherit from the DerivedType EClass and there would be an EClass for each Attribute of MusicItem which inherits from Field. This metamodel would differ from the one from figure 4 in any case.

To get the target metamodel, which can be used in a custom editor or with other tools, another transformation is needed. This transformation bridges the gap between the generic metamodel and the tool-specific metamodel. Although one has to implement or customize this transformation for each new tool, this task is much simpler than transforming the original metamodels again for every new metamodel and tool. One reason for this is, that one has to overbear the technological gap between both frameworks only once. All other transformations, which bring the metamodel into the custom format, can be done within the same framework. For such tasks, those frameworks often offer powerful transformation frameworks, like OAW or Atlas in EMFs technological space.

5.3 M1-Level Transformation

With respect to the taxonomy of model transformations of Mens et al. [MCVG05], the transformation of models at M1-level is only a horizontal one. The level of abstraction stays the same, both models (source and target) reside at M1-level. This makes the transformation a bit easier, because the target model only must conform to the metamodel without raising or lowering the level of abstraction, what was done during the metamodel transformation.

Similar to the transformation of the metamodels, the model transformation at M1-level would be a generic one. The result of this transformation would also be a generic model which conforms to the generic metamodel. Although such a model would include all information of the original model, but would rather be useless for a specific modeling tool, just like the generic metamodel.

For this reason, a framework-intern transformation of the generic models into models, which conform to specific metamodels, like the MusicLibrary metamodel, is needed again. Such a model could look like the one of figure 5, which conforms to the MusicLibrary metamodel. Again, it would be much easier to transform generic models into models that are conform to a specific metamodel, than reimplementing or customizing the transformation between both frameworks. There are also many other advantages, like the assurance of quality for any bridge or the lower number of transformations, when using the M3-level based bridging pattern which would suggest to apply this bridging approach.

5.4 Alternative approaches

This section introduces alternative approaches of bridging both frameworks. These approaches differ in the way of implementation (or access of the different models) as well as the applied bridging pattern.

Oslo Repository-EMF connection Rather, this approach offers an alternative in the implementation of the bridge, than in the applied bridging pattern. Since Oslo uses the SQL Server-based repository, a transformation could access this repository directly. The M3-level based bridging approach, which was introduced in section 5 uses Oslo's schemas and models in their textual representation. For instance, an Oslo-Schema could be generated from the EMF metamodel for the MusicLibrary example, by using a custom generator, a custom DSL or by customizing the serialization function of EMF. An Alternative for this would be the direct access of the Oslo Repository. This could be realized by SQL statements or already existing technologies, like Teneo[25] which already offers database serialization for EMF models. Using such technologies would connect EMF directly to the repository without the need of running the whole process of publication for each (meta)model.

M2-level based bridging pattern This approach introduces a real alternative to the M3B pattern, which implicitly was named in the previous sections. The idea of this bridging approach is to transform metamodels and models between two frameworks or tools without an M3-level mapping. Figure 8 shows the idea of this approach for the Oslo-EMF bridge. The M2B approach uses custom transformations for each metamodel or schema and its model instances. Applying this bridging approach requires that the user implements or customizes a bridge for each metamodel and it's model instances. The advantage of this approach is, that models and metamodels match a custom format after the transformation, instead of being represented by a generic (meta)model. This makes framework-intern transformations unnecessary but requires much more transformations between different technological spaces which raises the redundancy and may lower the quality of code or even the bridge itself. Nevertheless, the M2B pattern may be useful for seldom-used bridges with a small number of involved metamodels and tools. In such cases this approach can save much conceptional work and can speed up the development process.

[25] http://www.eclipse.org/modeling/emft/?project=teneo

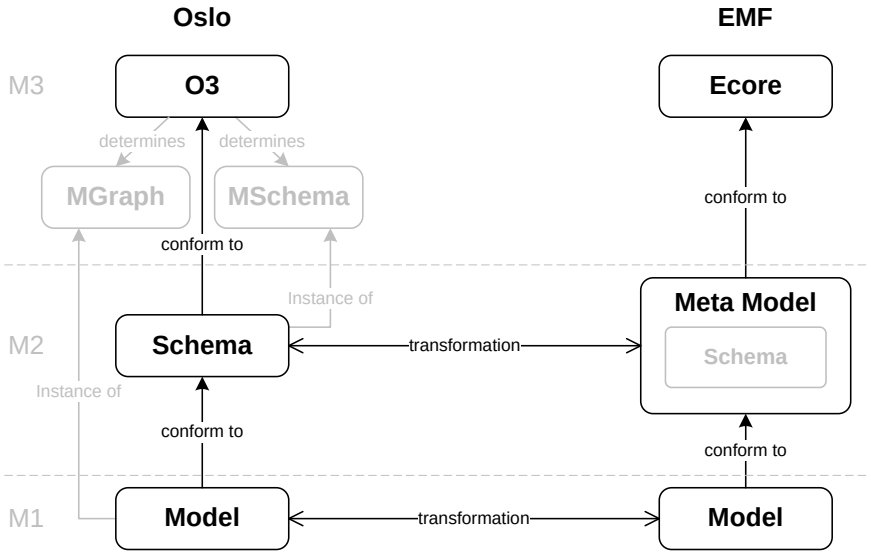

Figure 8: M2-Level based bridging of Oslo and EMF

Vertical transformations The last bridging approach, which will be introduced as an alternative for the M3B pattern, is also a kind of an M3-level based bridge. In contrast to the bridge of section 5, which was shown in figure 6, this pattern uses vertical bridging to transform the MSchema and MGraph parts of the O3 metametamodel into separate metamodels for MGraph and MSchema at M2-level. Furthermore, the schema, which originally is located at M2-level, is transformed into a model at M1-level, which describes the schema and is conform to the MSchema metamodel. The model-transformation remains horizontal at M1-level, but in contrast to the other bridging approach, the target model is conform to the MGraph metamodel, not the schema. This conformance relationship will emerge at the second step of the bridging. In this step, a vertical and framework-intern transformation is used to create a metamodel (such like in figure 4) from the schema-describing model at M1-level. The last transformation affects the model. At this step, an in-place transformation will be done on the model to build up the conformance relationship between the model and the metamodel. Figure 9 shows the first step of this bridge which is used to overbear the gap between the technological spaces. This pattern shall be understood as a suggestion for an alternative way of bridging. It neither has been tested for a particular bridge, nor has it been investigated with regard to functionality, advantages or disadvantages.

6 Related Work

There are many tools and papers which deal with the bridging of several modeling tools and frameworks. A large number of them are applying the M3B approach, which is based on the mapping of the metametamodels of both tools. Examples for such papers and tools are the ARIS-to-

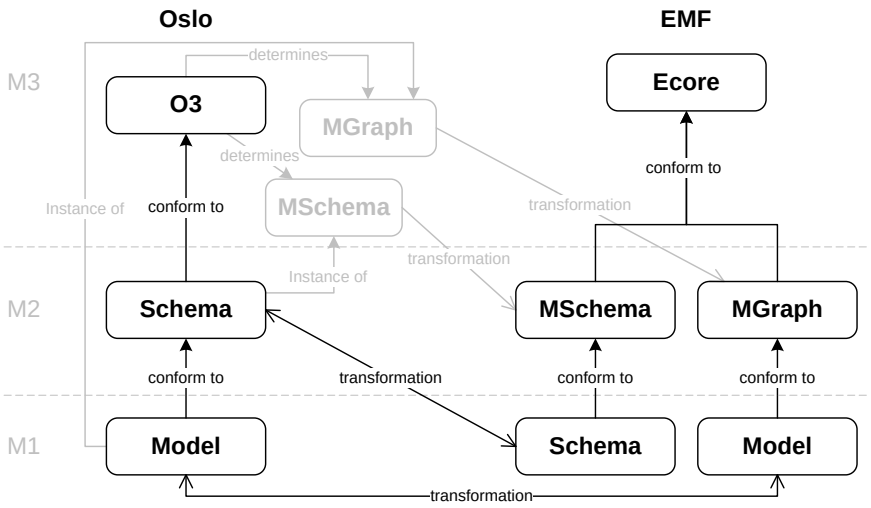

Figure 9: Vertical bridging of Oslo and EMF

EMF bridge [KK07], the MetaEdit-to-EMF bridge [Ker08] and the Visio-to-EMF bridge [KK09] from the University of Leipzig, the GME-to-EMF bridge [BBC+05] and the MS/DSL Tools-EMF bridge [BHJ+05] from the AtlanMod team, as well as the EMF-to-XML bridge [SBPM09] or the Java Architecture for XML Binding (JAXB) [Sun09] by Sun Microsystems[26]. Besides the M3B pattern for bridging modeling tools and frameworks, there are also approaches which use the M2-level bridging pattern or other approaches to bridge (modeling) tools [BBJK05, KLN03].

There is also an already existing bridge between Oslo and EMF [JB09], which was introduced at the eclipse summit, Ludwigsburg, Germany in October 2009. This bridge also scopes the MSchema part and uses a separate transformation model which is also used in [BBC+05] and [BHJ+05] for bridging both frameworks. Actually, the bridge is still in development, even though a first working implementation exists. Using this implementation, they already transformated a whole zoo of metamodels (schemas) for “M” from EMF.

[26] http://www.sun.com/

7 Summary and Conclusion

This paper introduced both, the Microsoft codename Oslo modeling framework and the Eclipse Modeling Framework and an approach for bridging them was developed. Therefore, the key concepts of both frameworks were highlighted and the process of modeling was explained for each framework by the MusicLibrary sample. This gave an introduction to the relevant components and the way of working with them. From this outline, relevant components and concepts for bridging could be derived. Based on these information, an approach for the bridging of both frameworks using the M3B pattern could be developed. Every step of this bridge was explained by the MusicLibrary sample which was introduced in the Oslo section. The theoretical background of the M3-level based bridging pattern is also explained in section 2. Finally, alternatives for bridging Oslo and EMF were named and explained. These alternatives scoped the architecture of the bridge as well as the possible implementation.

A next step would be a concrete implementation of the bridge, using the M3B pattern. As already mentioned in section 6, the AtlanMod team is developing an Oslo-to-EMF bridge yet. This bridge already delivers correct, MSchema conform schemas. From this point of view, it would be unnecessary to implement a further bridge. Another topic for future investigation could be the usage of EMF as a central hub for bridging between a large number of modeling tools. This would enable the user to build transitive bridges, what would reduce the effort for the building of bridges. Furthermore, this could enable the possibility for synchronization of modeling tools (for example at modeling-time).

Literatur

[BBC+05] Bézivin, Jean ; Brunette, Christian ; Chevrel, Régis ; Jouault, Frédéric ; Kurtev, Ivan: Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF). In: *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA'05*. San Diego, California, USA, 2005

[BBJK05] Bézivin, Jean ; Bruneliére, Hugo ; Jouault, Frédéric ; Kurtev, Ivan: Model Engineering Support for Tool Interoperability. In: *Workshop Model Transformations in Practice, collocated with MoDELS 2005*, 2005

[BHJ+05] Bézivin, Jean ; Hillairet, Guillaume ; Jouault, Frédéric ; Kurtev, Ivan ; Piers, William: Bridging the MS/DSL Tools and the Eclipse Modeling Framework. In: *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*. San Diego, California, USA, 2005

[JB09] Jouault, Frédéric ; Bruneliére, Hugo: *Possible Benefits of Bridging Eclipse-EMF and Microsoft „Oslo"*. online. `http://www.eclipsecon.org/summiteurope2009/sessions?id=885`. Version: October 2009

[Ker08] Kern, Heiko: The Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF Using M3-Level-Based Bridges. In: Gray, Jeff (Hrsg.) ; Sprinkle, Jonathan (Hrsg.) ; Tolvanen, Juha-Pekka (Hrsg.) ; Rossi, Matti (Hrsg.) ; University of Leipzig (Veranst.): *8th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2008* University of Leipzig, 2008, 14–19

[KK07] Kern, Heiko ; Kühne, Stefan: Model Interchange between ARIS and Eclipse EMF. In: Tolvanen, Juha-Pekka (Hrsg.) ; Gray, Jeff (Hrsg.) ; Rossi, Matti (Hrsg.) ; Sprinkle, Jonathan (Hrsg.) ; University of Leipzig (Veranst.): *7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2007* University of Leipzig, 2007, 105–114

[KK09] Kern, Heiko ; Kühne, Stefan: Integration of Microsoft Visio and Eclipse Modeling Framework Using M3-Level-Based Bridges. In: Hein, Christian (Hrsg.) ; Ritter, Tom (Hrsg.) ; Wagner, Michael (Hrsg.) ; University of Leipzig (Veranst.): *2nd ECMDA Workshop on Model-Driven Tool & Process Integration, 24 June 2009, at Fifth European Conference on Model-Driven Architecture Foundations and Applications 2009* University of Leipzig, 2009

[KLN03] Karsai, Gabor ; Lang, Andras ; Neema, Sandeep: Tool Integration Patterns. In: *Workshop on Tool Integration in System Developement, ESEC/FSE*. Helsinki, Finland, September 2003, S. 33–38

[Küh06] Kühne, Thomas: Matters of (Meta-) Modeling. In: *Software & Systems Modeling* 5 (2006), December, Nr. 4, 369–385. http://dx.doi.org/http://www.springerlink.com/content/f56076p682811625/. – DOI http://www.springerlink.com/content/f56076p682811625/

[MCVG05] Mens, Tom ; Czarnecki, Krzysztof ; Van Gorp, Pieter: A Taxonomy of Model Transformations. In: Bezivin, Jean (Hrsg.) ; Heckel, Reiko (Hrsg.): *Language Engineering for Model-Driven Software Development*. Dagstuhl, Germany : Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005 (Dagstuhl Seminar Proceedings 04101). – ISSN 1862–4405

[Mic09a] Microsoft Corporation: *Microsoft Codename Oslo product site*. http://www.microsoft.com/soa/products/oslo.aspx. http://www.microsoft.com/soa/products/oslo.aspx. Version: October 2009

[Mic09b] Microsoft Corporation: The „Oslo“ Modeling Language Specification / Microsoft Corporation. 2009. – Forschungsbericht

[Obj09] Object Management Group: *OMG's MetaObject Facility*. http://www.omg.org/mof/. http://www.omg.org/mof/. Version: April 2009

[Pur08] Purdy, Douglas: *A lap around Oslo – at the Professional Developers Conference (PDC), Los Angeles Convention Center*. http://channel9.msdn.com/pdc2008/TL23/. http://channel9.msdn.com/pdc2008/TL23/. Version: October 2008

[SBPM09] Steinberg, Dave ; Budinsky, Frank ; Paternostro, Marcelo ; Merks, Ed ; Gamma, Erich (Hrsg.) ; Nackman, Lee (Hrsg.) ; Wiegand, John (Hrsg.): *EMF: Eclipse Modeling Framework*. 2nd. Addison-Wesley, 2009. – 704 S. – ISBN 0321331885

[SKN92] Scheer, A.-W. ; Keller, G. ; Nüttgens, M.: Semantische Prozessmodellierung auf Grundlage Ereignisgesteuerter Prozessketten (EPK) / Institut für Wirtschaftsinformatik–Universität des Saarlandes, Saarbrücken. Version: 1992. http://www.iwi.uni-sb.de/Download/iwihefte/heft89.pdf. 1992. – Forschungsbericht

[Sun09] Sun Microsystems: *Java Architecture for XML Binding (JAXB)*. online. http://java.sun.com/developer/technicalArticles/WebServices/jaxb/. Version: December 2009

[SVB+06] Stahl, Thomas ; Völter, Markus ; Beltin, Jorn ; Haase, Arno ; Helsen, Simon: *Model Driven Software Development: Technology, Engineering, Management*. John Wiley, 2006. – 428 S. – ISBN 978–0–470–02570–3

[Whi04] White, Stephen A.: *Business Process Modeling Notation (BPMN)*. Business Process Management Initiative (BPMI), 2004. – 296 S. http://www.workflow-research.de/Downloads/BPMN/