Groovy and Grails Meet Eclipse Modelling Framework

Kent Inge Fagerland Simonsen[1,2], Florian Mantz[2], Alessandro Rossini[3], and Adrian Rutle[2]

[1]Machina AS, Domkirkegaten 3, 5017 Bergen, Norway
[2]Bergen University College, P.O. Box 7030, 5020 Bergen, Norway
[3]University of Bergen, P.O. Box 7803, 5020 Bergen, Norway

Abstract

This paper presents an approach to combine model-driven engineering with a popular web framework. In particular, it presents a case study of an implementation of the newest version of Noark 5, the Norwegian standard for archives in the public sector. In this project the data objects are modelled in the Eclipse Modelling Framework and the implementation is based on Groovy and the Grails web framework. This paper also describes the EMF2GORM tool, which is developed in order to automatically generate data classes for Grails. In addition to supporting simple classes and relations between them, this tool also supports setting other properties and constraints such as specifying that a data class should be available for free-text searching and specifying constraints that span over multiple model elements.

1 Introduction

Model-driven engineering (MDE) has been promoted for about a decade as a new paradigm of software development. In MDE, models are regarded as first-class entities of the development process. These models are used to automatically generate (parts of) software systems by means of model-to-model and model-to-code transformations. This is in contrast to traditional software development, in which source code is the main artefact of the development process while models are used for documentation purposes. Similar to previous shifts of paradigms, the industrial acceptance of MDE is at an early stage. Meanwhile, strict time-to-market constraints have promoted industrial adoption of high productivity techniques and tools to facilitate fast prototyping and incremental deliveries.

This paper presents an approach to combine MDE with a state of the art web framework. In particular, it presents a case study of an implementation of part of the newest version of the Norwegian standard for archives in the public sector, Noark 5 [14]. Noark 5 is a specification of requirements for electronic record-keeping which

This paper was presented at the NIK-2010 conference; see http://www.nik.no/.

is mandatory for all government agencies and administration in Norway. Noark 5 is loosely based on MoReq 2 [3], the record-keeping standard for the EU. The Noark 5 specifications are continuously revised and released.

The implementation of Noark 5 is part of the Friark project [6]. The project was initiated by Machina AS [13] in the summer of 2009 and is released under the GPLv3 license [5]. As indicated by the County Governor of Sogn og Fjordane in [4], a free software implementation of Noark 5 is expected to lead to economical and qualitative benefits.

Friark was allocated very little and sporadic resources. So Machina decided to exploit its expertise with Groovy [11] and the web framework Grails [8]. However, in order to facilitate portability, the data classes in Friark were specified using the Eclipse Modeling Framework (EMF) [2, 15]. EMF/Ecore was prefered over other formats because Machina has a long-term plan to migrate to model-centric development and in particular to the Eclipse platform. In order to bridge EMF with Grails, the tool EMF2GORM was developed to generate code from models. This tool is an Eclipse plug-in and represents the main contribution of this work.

The remainder of the paper is organised as follows. Section 2 presents the technologies and tools used in the Friark project. Section 3 introduces the EMF2GORM tool. In Section 4, the benefits and issues given by the combination of EMF with Grails are discussed. Finally, in Section 5 some concluding remarks and ideas for future work are presented.

2 Technologies and Tools

Groovy

Groovy is a dynamic programming language that runs on the Java Virtual Machine (JVM). Compared to Java, Groovy allows for compact code through features such as automatic getter and setter methods for all public fields, extended API and *closures* [10]. A closure is a concept introduced by functional programming languages: it is a first-class function with free variables that are bound in the lexical environment. In addition, Groovy enables seamless integration with Java libraries and can be easily deployed on existing Java environments. These features and the fact that Machina had existing Groovy expertise made Groovy the preferred choice for programming language on Friark.

Grails

Grails is a web application framework that is geared towards rapid development [1, 7]. It is inspired by the well-known web application framework Ruby On Rails (ROR). Like ROR, Grails is based on the *code-by-convention* paradigm which favours conventions over configurations. Grails is built as a layer on top of the popular Java frameworks Spring and Hibernate, which makes it suitable for use in various Java environments. Grails is implemented in both Java and Groovy, but most applications based on Grails are implemented using Groovy as the main language.

One of Grails' strengths is in its implementation of the Model-View-Controller (MVC) pattern. The controller in Grails consists of Plain Old Groovy Objects (POGOs), which are analogous to Plain Old Java Objects (POJOs). POGOs in the controller have a closure for each action. The view is handled by Groovy Server Pages (GSP) which are analogous to Java Server Pages (JSP). The main difference

between JSP and GSP is that GSP pages use Groovy code in scriplet mode and have its own set of default taglibs. Grails also enables addition of custom taglibs, which has been exploited in Friark. Furthermore, the model uses Grails' Object Relational Mapping (GORM), which is the persistence layer in Grails. This layer builds on top of the persistence framework Hibernate [12]. The data classes are called domain classes and are injected automatically with convenient methods such as `save()` and `list()`. The domain classes themselves are implemented as POGOs. This reduces the length of the code since the compiler adds getters and setters automatically. In addition, there are several conventions such as a closure called `constraints` which defines constraints on the objects of a domain class. The Friark project uses mainly the model part of Grails' MVC implementation since only the data classes are modelled in the first stage. Grails was particularly suited to Machina's need in the Friark project because it already comes with tools to build controllers and services.

Eclipse Modeling Framework

Eclipse is a popular integrated development environment (IDE). It is implemented primarily in Java and can be used to develop applications in e.g. Java and C++. EMF is a modelling framework built on Eclipse. EMF is considered a low cost entry to MDE and is the *de facto* standard for structural modelling in the Eclipse community. EMF is also highly extendible by means of plug-ins. This made it a good choice for Friark since EMF could be extended to transform a EMF model to GORM classes.

As a part of the code generation process, EMF generates a special version of the model called a GenModel. The GenModel may contain additional implementation specific information. In Friark and EMF2GORM any additional information in the GenModel is ignored. Hence, this step could be skipped and the Gorm classes could be generated directly from the EMF model.

The code generation process in EMF usually follows three steps:

1. Creation of an EMF model using the tree-based or graph-based editors provided by EMF/GMF. Alternatively an EMF model can be imported from other formats.

2. Generation of a GenModel from the EMF model.

3. Code generation from the GenModel.

Another reason to use Eclipse in Friark is its metamodelling capabilities. This may be used in future versions of EMF2GORM to formalise additional constraints on the data model and to model and generate the code for other classes such as controller classes.

3 EMF2GORM

At the beginning of the development of Friark, Machina was not able to find tools to generate GORM classes from the EMF model. Therefore, Machina decided to develop a new tool, EMF2GORM. One reason to create EMF2GORM was to easily control the generation process and add new features when the need was discovered. The tool needed to be general enough that it could be used for other projects

that would rely on EMF and Grails when Friark was completed. Java Emitter Templates (JET) [15] were considered for the code generation, but since the authors had considerably more experience using Groovy and *builders* [9] than template based approaches, EMF2GORM was developed using familiar tools.

EMF2GORM Flow

EMF2GORM is called from the view of a GenModel in the EMF editor. It expects the GenModel to hold one `EPackage` with several `EClasses`. Each `EClass` may hold an arbitrary number of `EAttributes`, `EReferences` and `EAnnotations`. How `EAnnotations` are handled by EMF2GORM are described in the next section.

For each `EClass` in the first `EPackage` in the GenModel, EMF2GORM creates a builder that is capable of building GORM classes. Then, the content of `EClass` is fed to the builder. When all the content has been fed to the builder, the class is converted into a textual representation of the GORM class and written to the appropriate file.

Additional Constraints and Attributes

In order to provide the required functionality in the data layer of Friark, the functionality of regular EMF models had to be extended. This is because some constraints used in Friark were not readily available in EMF. This was done by adding special `EAnnotations` and handling them while generating the GORM classes in EMF2GORM. An approach using Object Constraint Language (OCL) for constraint specification was considered but not realised because this would have required another transformation from OCL expressions to GORM constraints. This was not possible because Machina had very limited time and resources for the project.

The additional constraints were specified in `EAnnotations`. Some of them are handled by EMF2GORM in the following ways:

- Code is embedded directly into the GORM classes' `constraints` closure, e.g. `inList` and `oneOf` constraints. The former makes sure that an element is equal to one of the elements in a list, while the latter instructs that at most one of a set of `EReferences` is set in any instance of the constrained class.

- Attributes are generated in the GORM classes, e.g. the `searchable` and `loggable` constraints. The former instructs some plug-ins of Grails that the constrained classes should be indexed for full-text searching, while the latter instructs that operations on the classes should be logged.

It is the authors experience that greater care should have been taken when defining the `EAnnotations`. Currently, how the `EAnnotations` are interpreted by EMF2GORM is decided by the key-value pairs it contains. In addition, there is typically only one such pair per `EAnnotation`. We have started migrating to a more elegant way of describing the `EAnnotations`, by using their name to decide their behaviour. We expect this to make both the usage and the code of EMF2GORM easier to understand.

The Output

An example of a GORM class generated by EMF2GORM is shown in Listing 1. This example is the same `FondsCreator` class as shown in Fig. 2.

Listing 1: Sample output

```
package org.friark.ds
class FondsCreator {
  String fondsCreatorID
  String fondsCreatorName
  String description
  static constraints = {
    fondsCreatorID(nullable: false)
    fondsCreatorID(unique: true)
    fondsCreatorName(nullable: false)
    fondsCreatorName(unique: true)
    description(nullable: true)
    description(unique: true)
    fonds(nullable: true)
    fonds(unique: false)
  }
  static hasMany = [fonds:Fonds]
  static mapping = {
  }
  static searchable = [except: ['fonds']]
}
```

4 Experience from Friark and EMF2GORM Development

At the time of this writing, Friark has been in active development using MDE techniques for nearly a year. EMF2GORM has been developed in parallel with Friark and features have been added as the need has arisen in Friark. One of the main obstacles in using EMF has been poor availability of documentation on extending Eclipse and particularly EMF. For instance, it was difficult to find out how to extend the GenModel editor with a new command, and to make that command behave as intended. This lead to some man-hours spent in reading source-code and testing different extension points. Except for this, there were no big obstacles to getting the development environment to cooperate properly, besides the general context switching involved in using different environments for different tasks.

Table 1 shows the evaluation of using MDE with Grails and EMF2GORM compared to plain Grails development based on our experience. "+" denotes that the combination was a better approach for the given criteria and "-" denotes that it was worse.

Since Machina was able to generate a preliminary version of the EMF model from the Noark 5 specification with most of the classes and attributes already in place, it is the authors' opinion that MDE made development time slightly faster in this instance. This was the case although some time was spent on developing the EMF2GORM tool for generating the model from the specification. As an added bonus we were able, as EMF2GORM matured, to reduce bugs in the data layer of

Criteria	Evaluation
Development speed	+
Bug frequency	+
Code consistency	+
Reusability/portability	+
Self documentation	+
Readability of code	-
Communication	+

Table 1: Evaluation of combining Grails with MDE

Friark. At the time of writing no bugs have been reported. This was thanks to the forced model validation of EMF before generating the GenModel.

Since the code of the application is generated, it is consistent through the entire data model tier. The model can be reused by implementing transformations into other languages, a feature that may be required since Machina searches for new customers for Friark. Moreover, the model can be used for documentation and customer communication which were also benefits of this approach. A drawback we had with the generated code was the reduced code readability. This is because more code than required was generated such as constraints specifying default behaviour of Grails. The readability of the code could be improved by improving the generation of GORM classes.

Approach in Friark

In Friark, an EMF model is used to model all the data classes. This model itself was partially generated automatically using the Noark 5 specification by using scripts and tools (see Fig. 1). The fact that we were able to automatically generate the model gave us a head start. This helped in representing almost all data structures that we needed later in the development of Noark 5 in an EMF model even though at that point some classes had flaws. Some of these flaws were actually flaws in the tables from the Noark 5 specification. The most noticeable of these were missing references. These are still being added as the need arises.

The Noark 5 specification includes a document that holds tables describing most of the attributes and references between the data classes in Noark 5. This specification for the data classes is available as an HTML document. Using the spreadsheet tool in OpenOffice.org and some light formatting, this HTML document

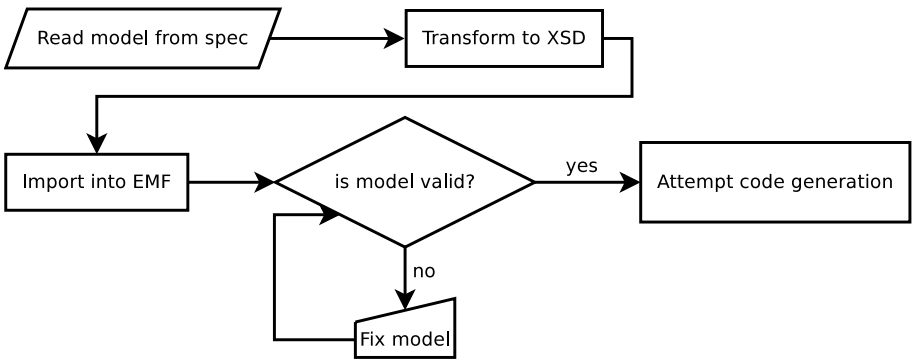

Figure 1: Generation of an EMF model from the Noark 5 specification

was transformed into a comma-separated values (CSV) file. Then, a Groovy script was developed to transform the CSV file into an XSD file. The XSD file was chosen as an intermediate step simply because it seemed to be the easiest format that EMF could import. Afterwards, EMF was used to import the XSD file and generate an EMF model out of it. Fig. 1 provides a overview of this process. With the EMF model in place, Machina started working on transforming that model into GORM classes.

Friark's EMF Model

Friark's EMF model contains 39 `EClasses`. Although this is not a very big number, we did experience some performance problems using the diagrammatic view of the model. However, there were no such problems using EMF's tree-based Ecore Editor. Hence, it became the main way to view and modify the model in Friark. Diagrammatic views are still used, but mainly for documentation and marketing purposes. In Fig. 2 an excerpt of the model is shown diagrammatically. This figure shows the `Base` class on the top which provides the classes extending it with a universally unique `systemID`. `Fonds`, to the left, is the class that represents an entire archive. Each archive is divided into archive-parts which are represented in the model by the `Series` class in the middle. Archive parts can be divided further into folders. `BasicFile`, on the right, is the class that represents the simplest folder. All other folders extends this class. At the bottom of the figure `FondsCreator` is shown. This class is generated into the class listed in Section 3.

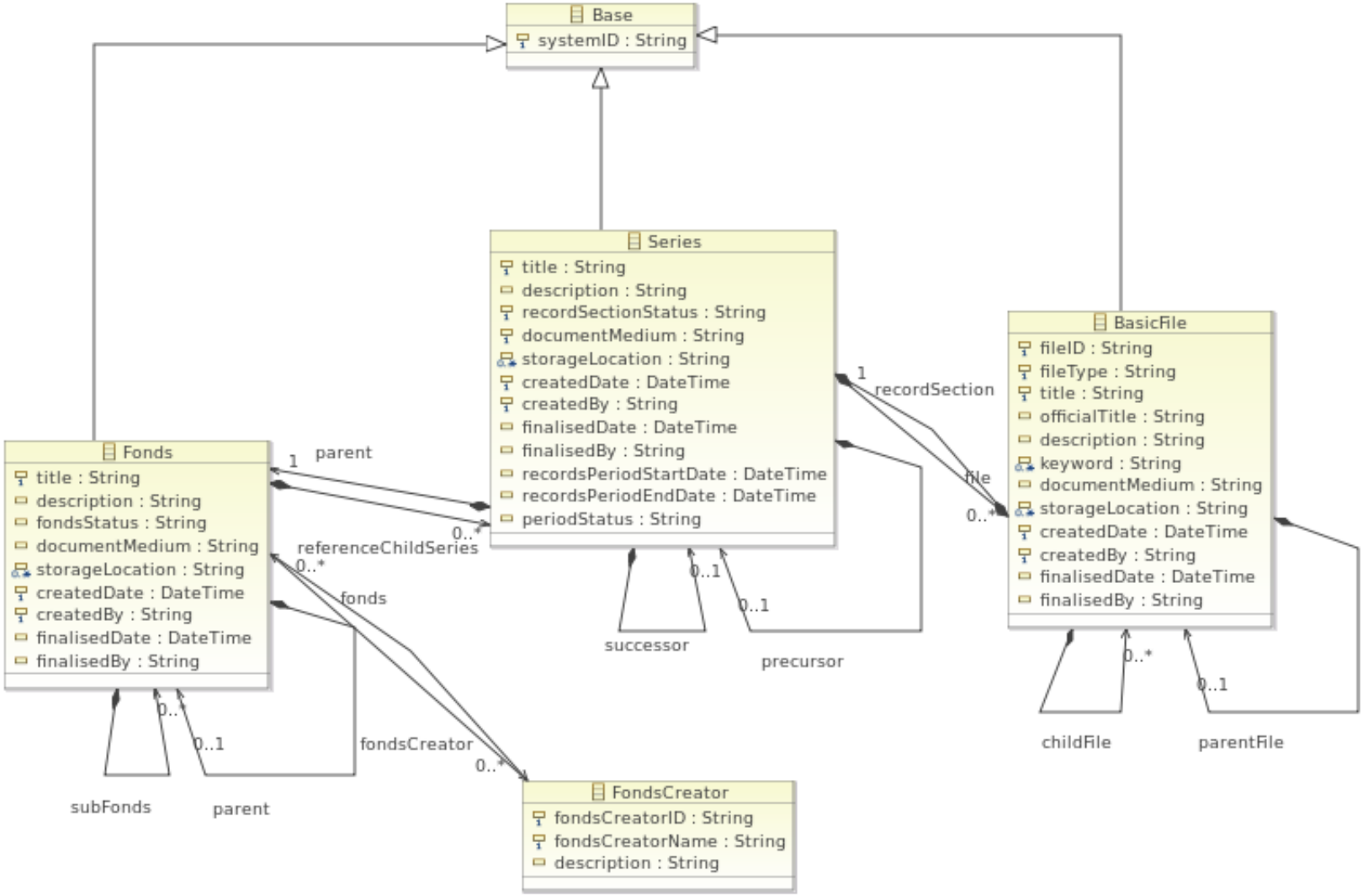

Figure 2: Diagrammatic view of a part of Friark's model

Another problem with using the diagrammatic view of Friark's EMF models is that many constraints are not visible. Since some of the new constrains include programming code (see Fig. 3) much information about the model is missing in the diagram. Moreover, since the logic of these constraints is specified as source code in `EAnnotation` values, the model validation is not able to validate these constrains.

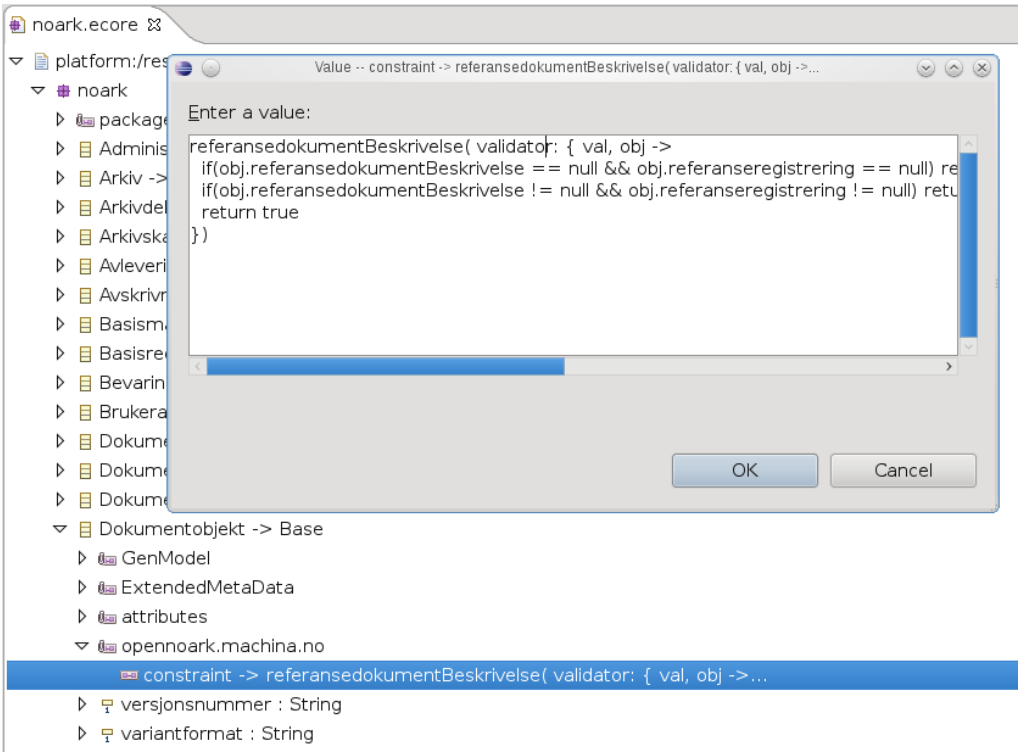

Figure 3: Example of a constraint written in code

5 Conclusions and Future Work

In the project described in this case study, some techniques and tools for high productivity web development are combined successfully with MDE. Even though the platform of choice for modelling was found to be slightly lacking on documentation, it proved to be relatively easy to extend to generate code for Grails. In addition, using MDE together with Grails did not seem to slow down development pace significantly, even though tools had to be developed from scratch.

The tool that has been developed for Friark, EMF2GORM, has made it possible to generate the data model with growing confidence. Despite its relative immaturity it is the authors' and Machina's belief that EMF2GORM, with some additional polishing, will be useful in other projects where data classes are modelled with EMF and translated to Grails code.

The data model of Friark project was to a large extent predefined by the Noark 5 specification. This project is not entirely representative of typical web application development. Further studies might be needed to determine if the methods and tools described here are useful in a more general sense.

EMF models are often the first choice to build structural models due to the rich tool set which is provided by EMF. In Friark, we first created XSD files from the requirement specification, then we used EMF's possibility to import XSD files and generate EMF models. There were several constraints in the domain model which

could not be expressed by EMF. For this reason we used `EAnnotations` to extend the expressive power of EMF. These constraints were manually added to the EMF model by annotating the model.

Usually `EAnnotations` are used to express features which are not supported directly by EMF. The code-generation process had to be adapted to handle these `EAnnotations`. Based on our experience in Friark, these annotations are not readable in the graphical editor and not easily validated automatically. It might be a better approach to extend Ecore to support these constraints. It could also be beneficial to model more of the application such as controllers in EMF. This would make the model more comprehensive and also make refactoring easier because more of it could be done on the model level. Doing these improvements will require to extend the standard EMF model and editors which requires expertise in Eclipse plug-in development and EMF's extension points.

References

[1] ALTERthought: Grails Vs. Rails – the Thrilla in Manila: A Study on Platform Productivity, http://alterlabs.com/technologies/java/grails-vs-rails-the-thrilla-in-manilla-a-study-on-grails-productivity/

[2] Eclipse Modeling Framework: Project Web Site, http://www.eclipse.org/emf/

[3] European Union: MoReq 2 Standard, http://www.moreq2.eu/

[4] Fondenes, J., Skarsbø, O., Øksenvåg, A., Guttormsen, K.T., Bennæs, S.O.: Noark 5 kjerne som fri programvare (in Norwegian). Technical report, County Governor of Sogn og Fjordane (August 2008)

[5] Free Software Foundation: GNU General Public License version 3 (June 2007), http://www.gnu.org/licenses/gpl.html

[6] Friark: Project Web Site, http://www.friark.org

[7] Geertjan Wielenga: Significant Productivity Gains with Grails?, http://java.dzone.com/news/significant-productivity-gains

[8] Grails: Project Web Site, http://grails.org

[9] Groovy: Builders Guide, http://groovy.codehaus.org/Builders

[10] Groovy: Closures Guide, http://groovy.codehaus.org/Closures

[11] Groovy: Project Web Site, http://groovy.codehaus.org

[12] Hibernate: Project Web Site, http://www.hibernate.org

[13] Machina AS: Company Web Site, http://www.machina.no

[14] Norwegian Archive: Noark 5 standard, http://www.arkivverket.no/arkivverket/Offentlig-forvaltning/Noark/Noark-5/English-version

[15] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0 (2^{nd} Edition). Addison-Wesley Professional (2008)