Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 63 (2015) 545 – 552

Procedia Computer Science

The 2nd International Workshop on (Meta)modelling for Healthcare Systems (MMHS 2015)

Model-Driven Software Engineering in Practice: a Content Analysis Software for Health Reform Agreements

Adrian Rutle[a,*], Kent Inge Fagerland Simonsen[b], Hans Georg Schaathun[c], Ralf Kirchhoff[c]

[a]*Bergen University College, P.O. Box 7030, 5020 Bergen, Norway*
[b]*Acos AS, Trollhaugmyra 15, 5353 Straume, Norway*
[c]*Aalesund University College, P.O. Box 1517, 6025 Aalesund, Norway*

Abstract

The Coordination Reform of 2012 requires Norwegian municipalities and regional health authorities to enter into legally binding service agreements. Although several research projects have been undertaken to analyse the implications of this reform, there is no central database where researches can be given access and analyse the service agreements. In this paper we present how we use model-driven software engineering and user-centric design in an initial development of an information system designed to allow researches to access and analyse service agreements. For this project, it was crucial to discuss the requirements of the system with domain-experts at a high level of abstraction in order to elicit feedback so that the development could proceed at a fast pace and in the right direction. Furthermore, given time and resource constraints, we elected to use a model driven approach using automatic code generation coupled with high-productivity frameworks. In this way we were able to create prototypes so that the developers could get fast feedback from the domain-experts and improvements could be implemented with minimal effort.

Peer-review under responsibility of the Program Chairs
Keywords: Model-driven Software Engineering; Coordination Reforms in healthcare; Metamodelling; MVCore

1. Background

The Coordination Reform (Samhandlingsreformen)[1] was introduced into Norwegian health care services in 2012. One of its features is mandatory and legally binding service agreements between local councils, who are responsible for primary health care, and public hospitals, who are responsible for specialist health care. The reform, while comprehensive, has many implications, and many challenges occur as the interface between specialist and primary health care is reshaped. Several research projects are funded to evaluate the reform, and analyse its implications. Hence, it is important to analyse the processes and the instruments involved in the implementation of the reform.

The service agreements is a comprehensive and growing source of data. The law on health and long-term care[2] specifies eleven distinct areas to be covered by these agreements. Each of the 430 local councils make agreements with at least one hospital or health trust, covering the eleven different agreement areas, in addition to one overarching agreement. In most cases, each agreement area has a separate document, for a total of twelve agreement documents

* Corresponding author. Tel.: +47-5558-7791 ; fax: +47-5558-7790.
E-mail address: adrian.rutle@hib.no

Peer-review under responsibility of the Program Chairs
doi:10.1016/j.procs.2015.08.383

per council. Annual revision is required for area-specific agreements, whereas overarching agreements may be revised less frequently. On a national level about 4730 agreement documents are produced per year.

The agreements provide information on how collaboration between primary and specialist care is organised, how it varies across the country, and how it evolves over time. To meet this end, an information system is required, not only to store all the agreements in a standardised format, but also to automate analysis of differences and similarities between documents. Both syntactic and semantic analysis is interesting, and the latter requires semantic coding of the agreement documents in addition to verbatim storage. Differences must be analysed along two separate axes, they vary between partnerships at any given time, and they vary over time within the same partnership.

This paper discusses the initial development towards an information system to analyse service agreements. To be able to focus on system and data modelling, we have used model-driven software engineering (MDSE) and user-centric design, and we present our own MVCore framework which supports rapid prototyping and code generation from the data model. A main benefit of MDSE is that it allows the domain-experts to take part in design and development without any knowledge of programming languages. Rapid prototyping also makes our design more user-centric by allowing domain-experts to give feedback immediately and try out different solutions and ideas.

2. Model-driven software engineering

Model-driven software engineering (MDSE) is a branch of software engineering where models are regarded the first class entities in all phases of the development process. In MDSE models are used to specify the software system under development, and automatic model transformations are used for different manipulations of these models, such as code-generation, model integration and decomposition, etc. This is in contrast to traditional software engineering practices where modelling is mainly used for documentation and communication purposes and the final product often deviates significantly from the models representing them.

Models specify the final product at a high level of abstraction that can be far more accessible to domain experts and customers than source code written in some programming language. When models are considered primary artifacts, and kept current throughout the development process, they can be used to elicit ideas and clear up misunderstandings with stakeholders.When a system is implemented based on a model. When a system is implemented based on a model one can also correct the mistakes in the model and produce another system based on the new, corrected model. Following the MDSE methodology, the updated models can be used to regenerate the software without manual, error-prone work.

In this project we use an MDSE framework (MVCore) developed at Bergen University College. MVCore is designed to model and allow code generation and fast prototyping of web applications. MVCore is a continuation of previous work[3], and we give an up-to-date presentation in the next subsection.

2.1. Developing with users

Our methodology is iterative and prototype driven, rather than analytical. Prototype driven development is an approach well known from many areas of design and engineering, and a *prototype* is a very flexible concept, spanning from crude drawings of an envisioned product to fully functional test products. A prototype can be defined as any tangible asset created to illustrate or demonstrate an aspect of a future product to other stakeholders. In early stages of development, most prototypes illustrate visions, while later prototypes may demonstrate proposed technical solutions.

Within software engineering, many modern development methodologies are wholly or partly prototype driven (e.g. scrum). A common principle in these methodologies is that the software under design must be deliverable at the end of each deveolpment iteration. Thus the software can be seen as a prototype, and every time a new feature is designed and added to the code base, a new prototype arises, ready to be shown to other stakeholders for discussion.

We stress that the key purpose of prototypes is to share and discuss ideas and solutions using the tangible demonstration or visualisation that the prototype represents. Thereby, objectives and requirements can continuously be adjusted and concretisised, and misunderstandings can be resolved as early as possible. In the case of MDSE, models also form prototypes, which can be discussed with users. This is particularly useful for data models, where the user may have a very good understanding of how the data are logically structured in the real world. Code generation mean that the same prototype can be viewed and discussed both as a model and as running software. Thus, the manual work to implement structural, model-level changes in a software prototype can be avoided.

2.2. MVCore as an MDSE solution

The MVCore framework is based on the Eclipse modelling Framework (EMF)[4] and Graphical Modelling Framework (GMF)[5]. It allows modelling of, and code generation for, web applications based on the Model-View-Controller (MVC)[6] pattern. Currently, MVCore allows code generation for the Grails web framework[7]. We now give a brief introduction to the frameworks and technologies on which MVCore is based. EMF is a framework for creating modelling applications. EMF provides the Ecore model[4] that can be used as a meta-metamodel by other projects. Metamodels can be created using Ecore as a meta-metamodel which in turn can be used to create graphical and textual editors (see Fig. 1). The editors can then be used to create instances of the metamodels. Later in this section we show the metamodel of MVCore, which is an instance of the Ecore model, and the editor which is created from this metamodel.

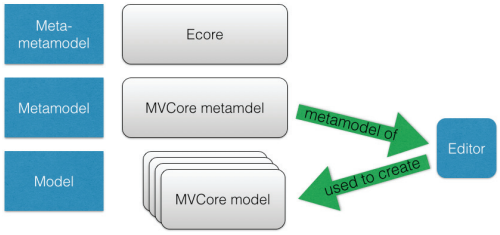

Fig. 1. EMF metamodelling hierarchy and MVCore metamodel as an example.

Grails is a web application framework based on the MVC pattern[6]. Grails uses the Groovy programming language[8]. Groovy is a dynamically typed multi-paradigm language that runs on the JVM and provides seamless integration with Java. Two of the main components of Grails applications are controllers and domain classes. These are the classes that MVCore is able to create based on MVCore models.

MVC is a popular architectural pattern for software application that have a graphical user interface. The pattern was originally developed for desktop applications, but is also popular for web applications and frameworks. MVC divides an application into three kinds of components: domain models, views and controllers. In addition, MVC defines the interactions between the components. In Grails, controllers receive commands from the user and send commands to the domain model. The model answers commands from controllers but is otherwise mainly passive.

MVCore consists of two main components: a domain specific modelling language (DSML) for modelling MVC web applications and a code-generation module. The metamodel of the DSML is defined using Ecore. The MVCore DSML is used to specify structural aspects of the application under consideration. Specifically, MVCore models are made up of domain classes and controllers. Furthermore, domain classes have properties and references and controllers have actions. The code generator creates the full code for domain classes and controllers while allowing developers to extend the functionality without modifying the generated code. Since MVC frameworks such as Grails often provide a scaffolding tool to automatically create views based on the model and controllers we have not reimplemented this functionality in MVCore.

The MVCore DSML allows the designer to specify the domain classes and controllers to be created. For each controller class, a list of CRUD methods can be generated automatically and other operations are supported by allowing developers to provide manual code for them. However, in many web applications, crud operations are sufficient for the majority of operations. From the domain classes and the controllers, one can generate a Grails project with all the necessary classes and controllers, then views can be generated for the controller classes simply by running the appropriate command of the Grails framework. Finally, the application can be built and run.

Updating the database, i.e. inserting and removing data, as well as querying the database will all be done seamlessly from the GORM objects, without the need of manual intervention or connections to the database socket. Grails's Object-Relational-Mapping (GORM) objects are instances of the domain classes. Fast prototyping can be achieved in this way, saving the time one would otherwise use to set up a database, write SQL (or other) code, call database related commands to update the database, set up web application server, etc. At the same time runnable prototypes can usually be generated completely from the models, without any manual work. Adapting and customising the prototype only requires changing the views and/or overriding controller actions by creating services with matching names. In the current version of the information system discussed in this paper only two out of seven controllers have any of their actions overridden by manual code although several of the views needed to be changed somewhat.

The metamodel for MVCore is shown in Fig. 2. The main elements of MVCore are instances of subclasses of the `MVCoreClass` class. The subclasses of `MVCoreClass` are `Controller` and `Domain`, representing controllers and domain classes. Controllers contain actions while domains contain attributes, references and constraints. Furthermore instances of `MVCoreClass` are contained in a `Package`.

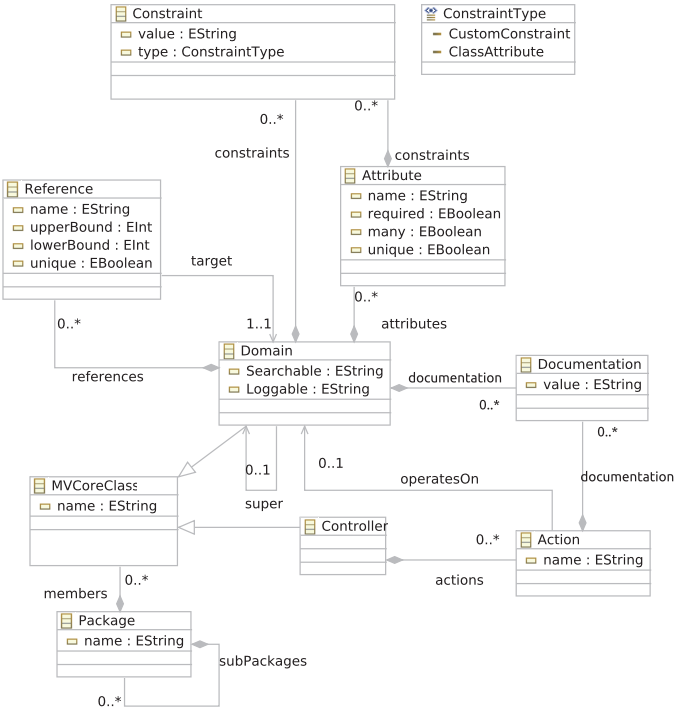

Fig. 2. The MVCore metamodel

3. The Text Analysis Database

The main purpose of the information system is to facilitate text and content analysis of the service agreements. Text and content analysis are well-known research methods in social sciences, and many tools exist for this purpose. This particular application context has a couple of features prompting a tailor-made system. The body of text is very well-defined, with well-defined, orthogonal categorisations by date, partners, and agreement areas. These categorisations form a set of meta-data which is significant for the analysis.

3.1. Requirements

Given the prototype-driven approach, the requirements are not fixed at this stage of the project. What we will do is to illuminate the vision by discussing technical opportunities in light of user desires. The starting points are informally phrased requirements from the user, i.e. the social science researcher, as follows: (i) find service agreements, (ii) compare different service agreements between regional/local health authorities, (iii) analyse changes in service agreements over time, and (iv) semantic comparison of documents as well as lexical comparison. Currently there is no central database of service agreements. The majority of the agreements are published on hospital trusts/Regional Health Authorities websites, but in some cases they must be requested directly from one of the partners in the agreement, and handled manually. The first requirement, mundane as it is, is merely a request for a central, public repository for existing, public data. The second and third requirements are related to plagiarism scanning, focusing on syntactic similarity between documents. In fact, the fourth author[9] has previously used PlagScan[10] for this purpose. The aim for the current project is to integrate it in a larger system to support batch scanning of the larger number of documents and tailor the presentation to the particular application problem.

The fourth requirement is related to taxonomies and semantic web as we know it from computer science, but we can also link it to methods from content and text analysis in the social sciences. In content analysis we identify a discrete set of concepts of interest and a list of words related to each concept. The text is then coded, manually or automatically, by tagging recognised words with the appropriate concept (code). This allows statistical analysis by counting frequencies both of individual concepts, and pairs of concepts occurring near each other in the text. Content analysis was applied by the fourth author[9], using the Leximancer software.

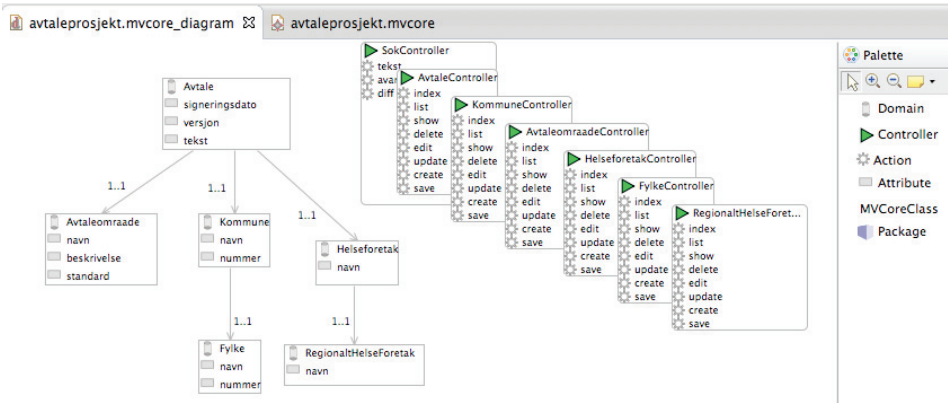

Fig. 3. The current model representing the structure of the agreement system, defined in MVCore's DSML editor

As a first prototyping step to explore the fourth requirement, it will be useful to implement relevant content analysis methods, as part of the agreement database. Where many content analysis tools already exist, reimplementation will allow us to integrate the different data models and provide more special purpose analytics.

The more ambitious vision of the fourth requirement is to go beyond the syntactic difference in requirements 3–4, and compare documents on a semantic level. Probably intractable in the general case, this may be possible in the specific domain problem, because of the structural requirements shared by the documents to be compared. A more immediate objective could be to classify and analyse the syntactic differences, building on content analysis techniques.

Based on the discussion of requirements, we can see a development plan of major prototypes for the project: (i) the document repository, tailor-made for the service agreements, (ii) syntactic difference of documents, (iii) semantic coding and support for content-analysis, and (iv) analysis of semantic difference of documents. Clearly, each of these major milestones will be developed through a sequel of incremental prototypes. At the time of writing, we have designed prototypes 1–2 above, while the last three are future work.

3.2. Structural Data Model

Based on the requirements and the development plan in the previous sections we defined a structural data model which represents the service agreement system. We used the MVCore DSML editor to define this model (see Fig. 3). Currently the domain model consists of only six classes, representing `Avtale` (Agreement), `Avtaleomraade` (Agreement field), `Kommune` (Municipality/Local authority), `Helseforetak` (Health Authority), `Regionalt helseforetak` (Regional Health Authority) and `Fylke` (County). These classes have attributes like `navn` (name), `signeringsdato` (signature date), etc. Also one controller is created for each class in the model. In these controllers we have CRUD actions such as List, Edit, Show, etc. In addition, one controller for searching across the domain classes was created.

Some screen shots of the prototype which was generated by MVCore are shown in Fig. 4. The relation between the domain classes and the generated web pages can be seen easily. For example, the web page `Avtale` (button left in the figure) is generated from the domain class `Avtale`, and it has the text fields `Avtaleomraade`, `Signeringsdato`, etc corresponding to the attributes of the domain class. In addition, some of the functionalities which were required for analysis of the agreements are implemented in the current version of the software, for instance (i) searching the agreements database based on various criteria, (ii) pairwise comparison of the agreements in order to determine similarities and differences among agreements, (iii) comparison of different versions of the same agreements to determine how much an agreements have changed over time, etc.

A screen shot showing the search and comparison functionalities is shown in Fig. 5. The comparison result follows standard Unix based `Diff`. That is, a line which begins with < is in the first file (top left in the figure) but not in the second one (top right). Furthermore, a line which begins with > is in the second file but not in the first. Finally, a line which begins with < and is followed by an almost similar line which begins with > means that the line in the second file is a changed edition of the line from the first file.

The development methodology we used in this project is a combination of MDSE and fast prototyping, something that was facilitated by the MVCore framework. We started with an initial model, and generated a web application which the domain-experts examined. The domain-experts discussed also the structure of the model, and gave us

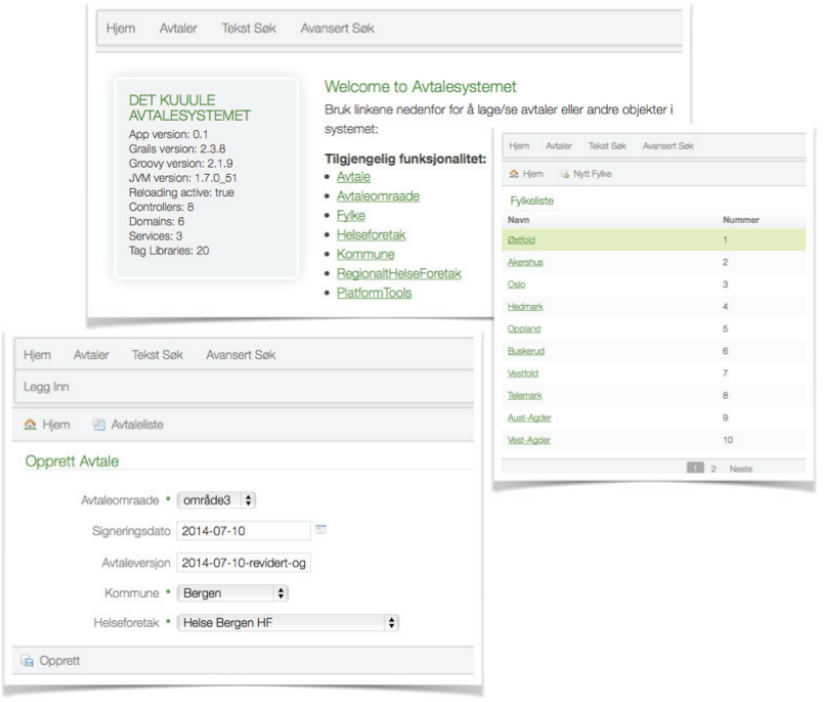

Fig. 4. The prototype generated from the model shown in Fig. 3

feedback about certain mistakes in the model. For instance, in an earlier version of the model we had modelled each agreement as a container of 11 unique agreement fields, each representing an agreement area. We changed this so that each agreement area is treated as an independent agreement in the model, regenerated the code of the application and deployed the web application during a few minutes.

Most of the feedback we got from the domain-experts could be implemented during the meetings, since we only needed to change the models, and regenerate (and redeploy) the application code. For instance, in order to differentiate between standard agreement areas and other user-defined areas, we added the boolean attribute `standard` in the domain class `Avtaleomraade`. Some other feedback came in emails requesting features explained as mockups. The mockups derived the way we modified the views mostly without changing the domain classes or the controllers.

3.3. Text Analysis

Content analysis covers a range of techniques and methods to extract meaning from text. Originally the term referred mainly to quantitative methods, but also qualitative methods may be included. A more comprehensive discussion of content analysis is provided in[11].

The first step in content analysis is categorisation and coding of the text. A set of concepts of interest, called *categories*, are identified, and each category is associated with a set of words or synonyms, which may occur in the text. For instance, *obligation* and *intent* form two categories in[9]. Category *obligation* can be associated with words like *must*, *shall*, *is obliged to*, etc., while *intent* can be associated with *may*, *should*, or *the parties consider*. Categories may be hierarchically organised. The category *knowledge transfer*[9], may comprise subcategories like *internship*, *ambulant team*, *meeting arena*, etc. This system of categories is similar to a taxonomy in other contexts.

Once the categories are identified, the text is coded by tagging recognised words with the corresponding category, and based on the coded text, statistical analysis can be conducted, ranging from simple frequency counts to more complex analysis of correlation between terms, such as pairs of categories occurring near each other in the text. One obviously interesting dependency between categories is that between either obligation or intent, and another which specifies the subject of the obligation/intent. The coding can be done manually or automatically. The design of categories is largely manual, but text analysis software can often offer support by proposing candidate categories.

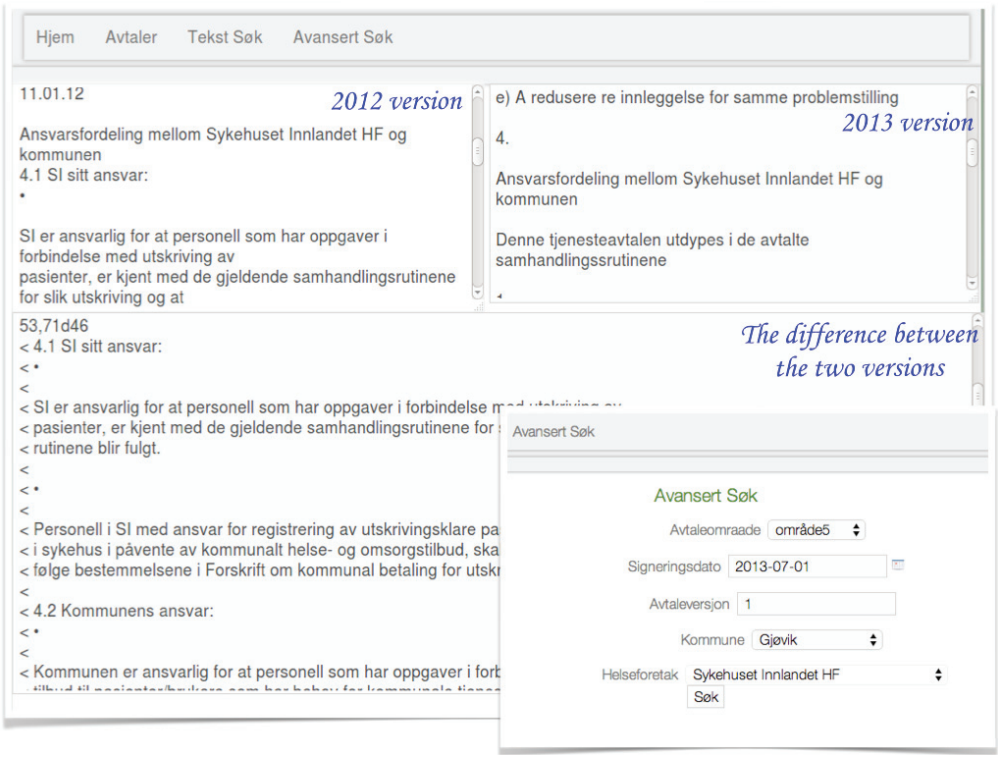

Fig. 5. Comparison of 2012- and 2013-versions of service agreement 5 between Gjoevik and Sykehuset Innlandet; also showing advanced search

The third prototype supporting standard content analysis techniques is expected to be straight forward, and it form the basis for the fourth prototype where semantic differences are to be analysed. It is not immediately obvious what kind of analysis will be undertaken by the fourth prototype, but there are several ideas to explore. One could perform content analysis to a syntactic difference, something which would give information about the nature of the changes. At the opposite extreme, one could compare the descriptive statistics produced by content analysis on different documents. To explore these and other alternatives, we will continue on our model-driven approach and develop a data model to capture the categorisation of content analysis and link it to the existing data model.

4. Related Work

We will summarise a few projects and experiences which are related to our development methodology based on MDSE. Model-driven techniques aim at improving productivity and platform independence. However, certain problems have been reported such as lack of mature tools, skilled model designers and domain-experts with knowledge in modelling. Despite this, several case-studies have shown the benefits of MDSE in industry; some examples will follow. Conventional software development and Eclipse-based model-driven development are compared in [12], concluding that the latter could be carried out in 11% of the time of the former, while simultaneously improving code quality. The benefits of domain-specific modelling gives at least 750% improvement in developer productivity and greatly improve quality of code and development process [13]. The experience of 15 years of MDSE at Motorola is described in [14], demonstrating significant benefits in quality and productivity. The application of MDSE is shown to improve performance (productivity) 2.6 times compared with a legacy project [15]. An empirical assessment of MDSE in industry (which includes the analysis of several case studies) concludes that many different MDSE approaches are being actively used in industry and delivering significant benefits [16].

In [17] the benefits of MDSE based on three industrial cases (car industry, SAP and embedded systems) are summarised as (i) abstraction and hiding details of complexity (ii) communication with non-technical staff (iii) simulation and model-based execution (iv) model-based testing. Despite these benefits, there is little empirical evidence of the acceptance of MDSE in industry [18,19]. In general, the case studies suggest that model-driven approaches are useful for

investigating some problems of complex systems despite the immaturity of tools. Examples of other domain specific modelling frameworks are MetaEdit+[20] and DPF Workbench[21,22]. These tools allow the users to create graphical DSMLs such as MVCore and also supports code generation based on DSML instances.

5. Conclusion and Future Work

We have discussed a software system to facilitate research on the service agreements which are required by the recent Coordination reform in Norwegian health care. We have presented an MDSE framework, called MVCore, for database driven web systems, and discussed how it benefits the development, and how it in particular allows rapid prototypes generated from high-level models. The software as presented is the first few prototypes in an on-going project. Other prototypes are in progress to meet the planned requirements. In addition, we will extend MVCore to generate code for other web application frameworks. We also intend to add support for modelling and automatic code generation of other actions than the simple CRUD actions. One possible way to do this is to combine MVCore with a tool for behavioural code generation such as PetriCode[23].

References

1. Helse- og omsorgsdepartementet, . The coordination reform - proper treatment - at the right place and right time. 2009. URL: `https://www.regjeringen.no/en/dokumenter/report.no.-47-to-the-storting-2008-2009/id567201/?q=coordination\&ch=1`; report No. 47 to the Storting (2008-2009). Submitted to the Storting on 19 June, accessed 2015-07-26.
2. Helse- og omsorgsdepartementet, . Lov om kommunale helse- og omsorgstjenester m.m. (helse- og omsorgstjenesteloven). 2011. URL: `http://lovdata.no/dokument/NL/lov/2011-06-24-30`; proposisjon til Stortinget (forslag til lovvedtak), accessed 2015-07-26.
3. Simonsen, K.I.F., Mantz, F., Rossini, A., Rutle, A.. Groovy and Grails meet Eclipse Modeling Framework. In: *NIK 2010*. ISBN 978-82-519-2702-4; 2010, p. 34–43.
4. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.. *EMF: Eclipse Modeling Framework 2.0 (2nd Edition)*. 2008. ISBN 978-0-321-33188-5.
5. Graphical Modeling Framework, . Project Web Site. 2015. URL: `http://www.eclipse.org/modeling/gmp/?project=gmf-tooling`; accessed 2015-07-26.
6. Reenskaug, T.. The model-view-controller (mvc) its past and present. 2003. URL: `http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/HM1A93.html`; accessed 2015-07-26.
7. Grails, . Project Web Site. 2015. URL: `http://grails.org`; accessed 2015-07-26.
8. Groovy, . Project Web Site. 2015. URL: `http://groovy.codehaus.org`; accessed 2015-07-26.
9. Kirchhoff, R., Grimsmo, A.. Samhandling og pasientforloep i stoepeskjeen - forloepige resultat fra arbeidspakke 1. In: *Forskerseminar for deltakere i prosjektene i evalueringen av Samhandlingsreformen*. 2013, .
10. PlagScan, . Product web page. 2014. URL: `http://www.plagscan.com`; accessed 2015-07-26.
11. Neuendorf, K.A.. *The Content Analysis Guidebook*. Thousand Oaks: Sage Publ.; 2002.
12. Krogmann, K., Becker, S.. A case study on model-driven and conventional software development: The palladio editor. In: Bleek, W.G., Schwentner, H., Züllighoven, H., editors. *Software Engineering (Workshops)*; vol. 106 of *LNI*. GI. ISBN 978-3-88579-200-0; 2007, p. 169–176.
13. Karna, J., Tolvanen, J.P., Kelly, S.. Evaluating the use of domain-specific modeling in practice. In: *9th OOPSLA Workshop on Domain-Specific Modeling (DSM)*. 2009, .
14. Baker, P., Loh, S., Weil, F.. Model-driven engineering in a large industrial context - motorola case study. In: Briand, L.C., Williams, C., editors. *MoDELS*; vol. 3713 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-29010-9; 2005, p. 476–491.
15. Kapteijns, T., Jansen, S., Brinkkemper, S., Houet, H., Barendse, R.. A comparative case study of model driven development vs traditional development: The tortoise or the hare. In: *4th European Workshop on from Code Centric to Model Centric Software Engineering: Practices, Implications and ROI*. 2009, .
16. Hutchinsen, J.E.. *An Empirical Assessment of Model Driven Development in Industry*. Ph.D. thesis; School of Computing and Communications, Lancaster University; UK; 2011.
17. Mohagheghi, P., Gilani, W., Stefanescu, A., Fernández, M.A., Nordmoen, B., Fritzsche, M.. Where does model-driven engineering help? experiences from three industrial cases. *Software and System Modeling* 2013;**12**(3):619–639.
18. Mohagheghi, P., Gilani, W., Stefanescu, A., Fernández, M.A.. An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empirical Software Engineering* 2013;**18**(1):89–116.
19. Mohagheghi, P., Dehlen, V.. Where is the proof? - a review of experiences from applying mde in industry. In: Schieferdecker, I., Hartman, A., editors. *ECMDA-FA*; vol. 5095 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-540-69095-5; 2008, p. 432–443.
20. The MetaEdit+ Workbench, . Project Web Site. 2015. URL: `http://www.metacase.com/mwb/`; accessed 2015-07-26.
21. Rutle, A.. *Diagram Predicate Framework: A Formal Approach to MDE*. Ph.D. thesis; Department of Informatics, University of Bergen; Norway; 2010.
22. Lamo, Y., Wang, X., Mantz, F., MacCaull, W., Rutle, A.. DPF Workbench: A Diagrammatic Multi-Layer Domain Specific (Meta-)Modelling Environment. In: Lee, R., editor. *Computer and Information Science*; vol. 429 of *Studies in Computer Intelligence*. Springer; 2012, p. 37–52. doi:`10.1007/978-3-642-30454-5\_3`.
23. Simonsen, K.I.F.. PetriCode: a tool for template-based code generation from CPN models. In: *Software Engineering and Formal Methods*. Springer; 2014, p. 151–163.