The Sales Scenario: A Model-Driven Software Product Line*

Henrik Lochmann and Birgit Grammel
SAP Research CEC Dresden
Chemnitzer Str. 48, 01187 Dresden, Germany
{firstname.lastname}@sap.com

Abstract: The management of variability is essential to reduce efforts and increase efficiency in development, especially for large size software companies who offer complex product portfolios. Within the research projects FeasiPLe and AMPLE, we develop concepts and tools to facilitate an end-to-end process for Software Product Line Engineering (SPLE). Both projects emphasize different viewpoints of combining Model-Driven Development (MDD) and Aspect-Oriented Software Development (AOSD) with concepts of traditional SPLE to integrate the benefits of each of these technologies. In this paper, we present the case study we are developing in both projects - the model-driven software product line *Sales Scenario*. We introduce its application domain and explain our approach in development. Although our work is to date not finished, we are able to give first insights into the challenges of product line engineering with a combination of MDD, AOSD and SPLE from scratch.

1 Introduction

In this paper we present a model-driven software product line, which is used as case study in the two research projects FeasiPLe [Con07b] and AMPLE [Con07a], SAP Research is partner of. Our goal is to document the stepwise procedure of creating a software product line with a combination of model-driven, aspect-oriented and product line technologies. With this case study, we want to demonstrate the feasibility of such a combined approach, which, to the best of our knowledge, does not yet exist within the area of business applications. We followed a bottom-up approach, starting from a reference implementation and leading to feature-based decomposed system models. We want to disclose the challenges of such a process and to rate our development approach. The here presented case study will be used to evaluate the tools and concepts we are developing within the two research projects. Our vision is to provide an end-to-end tool chain for SPLE, from feature description to implementing code, with regards to traceabilty.

The paper is structured as follows: we begin with an introduction to the application domain of our product line in Sec. 2. An overview about the solution space is given in Sec. 3. Our approach to development and the challenges herein are documented in the sections 4 and 5. We conclude and give a short outlook in Sec. 6.

*This work is partially supported by the EC-funded project AMPLE and the German BMBF-funded project feasiPLe.

2 Problem Space

This section starts with an introduction to the domain of our case study. Followed by a description of its feature model, the core features are pointed out, stating their dependencies and interactions.

2.1 Introduction to the Domain

Our case study demonstrates business application engineering in the domain of enterprise software, a rather large domain. Exemplary solutions centered around Enterprise Resource Planning (ERP) include Product Life Cycle Management (PLM), Supply Chain Management (SCM), or Supplier Relationship Management (SRM). Such solutions must be adapted and customized to a particular company (no two companies are the same), business applications often have thousands of configuration settings. To reduce the complexity for the sake of conciseness, we focus on one specific sub-domain in this paper – Customer Relationship Management (CRM) – combined with some parts of the aforementioned solutions. Furthermore, the case study is not intended to be complete, it rather concentrates on those parts of CRM that seem to be relevant and meaningful enough to be integrated into a variability-driven context. Therefore, the presented example is called *Sales Scenario* to clearly distinguish from CRM as explained by Buck-Emden and Zencke in [BEZ04].

2.2 Feature Model

Main purpose of the Sales Scenario is the holistic management of business data, including central storage and access controlled retrieval. It focuses on product sales processes, where the core features comprise: *Customer Order Management*, *Payment*, *Account Management*, *Product Management* and *Communications*. These features are depicted in the case study's feature model in Fig.1, created with an Eclipse-based feature modeling tool developed by Christian Wende et al. from Dresden University of Technology, Germany. The feature notation relates to the original FODA notation [KCH+90] but is purely cardinality-based and abstains from the original, often confusing bubble syntax. In order to clarify the interplay between certain features and to give an understanding of what is covered in the Sales Scenario, the following description of the overall sales process and its individual components is presented. To better break down certain features, this process was clearly divided into separate steps.

Step 1 To start business with a potential customer, the corresponding master data, such as customer name and address, budget estimation and a description of the sales opportunity and its time frame is saved to a customer profile (*Prospect*) by using the functionalities of *Account Management*.

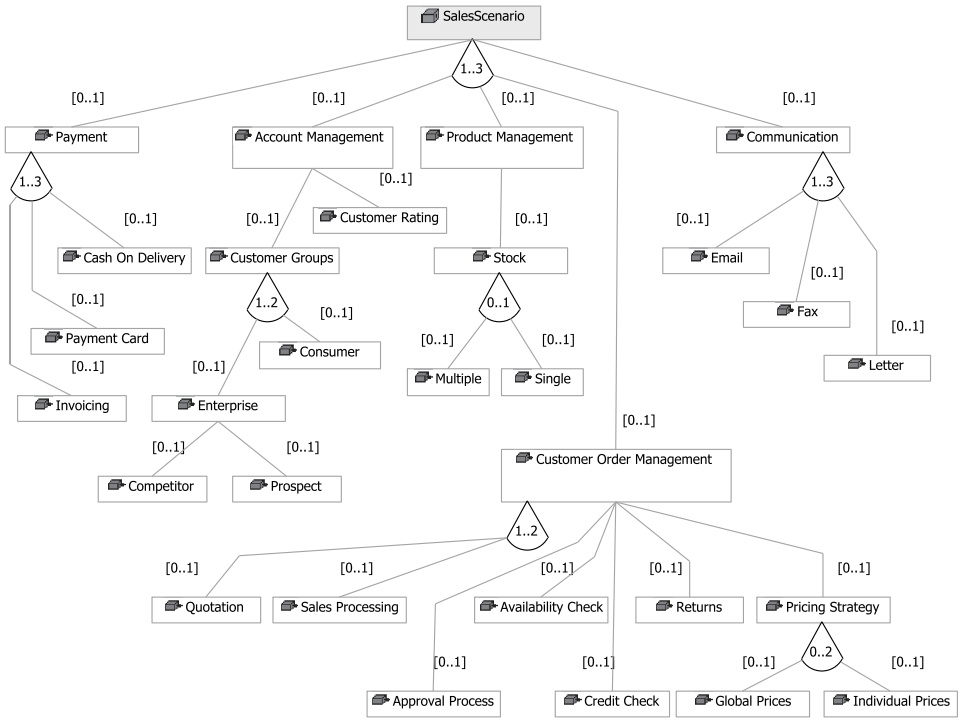

Figure 1: Feature Model of the Sales Scenario Product Line

Step 2 After subsequent evaluation and go/no go decision handling by sales management, another employee of the sales office creates a quotation (i.e., offer) using the *Quotation* functionalities and configures a quotation template based on the sales opportunity data. A potential discount is included into such a quotation by using *Pricing Strategy*. The calculation strategy is based on the categorization of *Prospects* in a *Customer Group*, estimated sales volume, and sales probability, resulting in an overall *Customer Rating*.

Step 3 After the sales office has contacted the customer and received an order, the system automatically converts the quotation into an order upon mouse click using the corresponding *Sales Processing* functionality.

Step 4 To check the creditworthiness of the customer, an (optional) *Credit Check* is performed before creating a quotation as well as before accepting a sales order, by interacting with the *Payment* module.

Step 5 An *Availability Check* is performed to check for sufficient *Stock* and necessary capacities in the warehouse. In case of *Multiple Stocks*, meaning, delivery processes and

storage of different goods depend on several warehouses, only those warehouses sufficiently close to the shipping address are included.

Step 6 If *Payment* is to be integrated into the sales process, it would be activated automatically upon creation of a binding sales order. Depending on the method of payment offered by the system and selected by the customer, at least one of the following feature options are realized: An automatic debit transfer from the customer's account can be triggered (*Payment Card*) or an invoicing document can be attached to the delivery to account for *Cash On Delivery*, or if payment is settled at a later stage, *Invoicing*.

Step 7 The order status is set to "sent" by an employee as soon as the order is delivered to the customer. Once an open invoice is paid by the customer, the order is marked as "paid" and "closed". In case an open or already "sent" sales order is returned by the customer, carried out by *Returns*, an *Approval Process* is triggered, which involves sales management processes for commitment.

The description of all *Communication* channels has been omitted, for the sake of clarity. Such instances are naturally related to mediation of quotations, invoices etc., whether sent by *e-Mail*, *Fax*, or *Letter*.

3 Solution Space

Following this paper's focus on the development process of model-driven SPLE, we decided to divide the explanation of our case study's solution space into two different parts. Thus, this section starts with an overview about the application's general architecture and applied technologies. In Sec. 4 a more detailed explanation to concrete solution artifacts is presented orthogonally to the development approach we followed to create them.

3.1 Architecture Overview

A descending node-by-node analysis of the feature model led to an initial architecture for the product line's solution space. The functional nature of the Sales Scenario features allowed the derivation of modular architecture blocks for main features. The interconnections between these blocks arose from the domain description and therein contained information about process-centric interdependencies. Additional parts, such as the user interface (UI), came from technical considerations and corresponding stakeholders.

Our product line's high-level architecture is depicted in Fig. 2 following the FMC notation [KGT06]. It outlines the key entities and their interactions according to the functionality described above. Prominent are well modularizable, coherent features on the one hand and wide spread, crosscutting features on the other. Exemplary for coherent features are *Stock Management* and *Payment Processing* components. Such components are to a certain

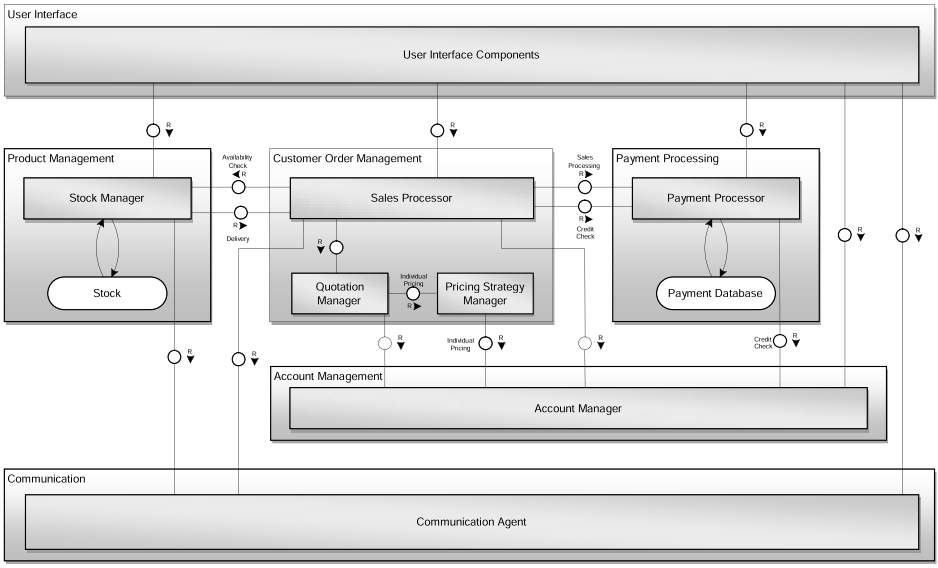

Figure 2: Sales Scenario Architecture Overview

degree self-contained and mainly expose their functionality for use by other components. The second feature type labels all those components that are scattered over a platform and interact actively with other components. Thus, such components encapsulate non-obvious cross-cutting concerns by widely distributing implementational or invokational parts of their functionality. A proper visual distinction between the two feature types and their corresponding architecture modules respectively is not given in Fig. 2. Nonetheless, one indication may be the number of communication connections of one block to others, where many connections prove cross-cutting functionality.

3.2 Platform Selection

Our software product line is a business application for product selling companies. Commonly, software applications in this domain are widely distributed, allowing multiple users and user groups to retrieve and modify different kinds of information. The distributed nature of business applications is a major technical constraint when searching for appropriate implementation means. In general, we had to make a selection between a Rich-Client Platform (RCP) with additional communication support and central knowledge database, and a web application platform. We decided to implement our product line as web application to minimize the engineering effort when following an RCP-based strategy, with regard to the various web application development frameworks, such as Spring [Spr07] or JBoss [JBo07]. A comprehensive state-of-the-art analysis revealed the JBoss Application Server

(AS) with its rich feature set and Eclipse IDE support as a promising solution. The JBoss AS provides an implementation of the J2EE component framework Enterprise JavaBeans (EJB). JBoss Seam [JBS07], the enhancing application framework for the JBoss AS, even increases the efficiency of this platform.

JBoss Seam bridges the gap between user interface and behavior implementations, i.e., Java Server Faces (JSF) and EJB session beans. This is done through a multi-layered session context stack managed with so-called *conversations* and helping templates on both sides. Additionally, the jBPM Process Definition Language (JPDL) allows seamless integration of navigation structures over dialog pages. For detailed information consider [JBS07].

4 Development Process

In this section we elaborate the approach we followed when designing and implementing the Sales Scenario product line, which was a bottom-up process. Thus, we decided to tackle the challenge by implementing one single product variant (cf. 4.1). This way we built a reference architecture for the modeling workbench we focus in our model-driven context (cf. Sec. 4.2). The last step in development of the Sales Scenario is the feature-based decomposition of the resulting system models. The current progress of this task is described in Sec. 4.3.

One might consider our development approach as *reverse engineering* process, which actually would not be correct. A reverse engineering process derives alternative, usually more abstract respresentations from an existing software system [CI90]. Here, this is only one part of the whole process. Before modelling our system (the actual reverse engineering), we have to identify, design and elaborate the necessary Domain-Specific Languages (DSL). Additionally, the system models we have to develop cover not only the initial product variant but also all other features of our product line. This way, the DSLs, their generators and according models constitute main assets of traditional domain engineering. Through a proper binding between features and model elements (cf. Sec. 4.3), we try to reduce the application or product engineering step to the choice of an appropriate variant from our feature model and adjacent adaptation steps of our domain models.

4.1 Implementing the Reference Architecture for a Single Variant

The initial exercise in our bottom-up development approach was to select a single product variant from our product line. The variant had to have preferably significant features, from a technical point of view, to provide an as complete as possible impression of the application's architecture. The chosen variant is depicted in Fig. 3. Because it does not anymore contain variable parts, only mandatory features are illustrated.

Based on the given feature selection, the domain description and architecture considerations, a reference implementation had to deliver a first set of realizations for the major parts

Figure 3: Variant for Reference Implementation

of our JBoss Seam web application. These parts are: data structures (EJB EntityBeans), UI screens (JSF pages), behavioural parts (EJB SessionBeans) and navigation models (JPDL page flows). Common analysis, design and implementation phases led to initial structures. We followed the concept that each coherent and for a business user meaningful feature, became a management component reflected in a *management page*, from the user interface point of view. Each management page lists corresponding data entities and provides access to appropriate functions. All management pages were aggregated in the so-called *control center*, another UI screen, which is also the entry point of the application. Figure 4 illustrates this concept on the basis of the *Quotation Management* feature.

The figure's upper left shows the management page for quotations. Here, all system-wide available quotations are listed, including their current lifecycle states and interaction possibilities. The figure's bottom-right part presents a step-by-step wizard to create quotation entities.

4.2 Deriving Modeling Means

The second task in our product line development process was the derivation of the necessary means to facilitate a purely model-driven approach. As modeling workbench we made use of the Eclipse Modeling Framework (EMF) [EMF07], mainly due to its rich feature set and lively supporting community. For code generation, we applied the generator framework openArchitectureWare (oAW) [OAW07], which integrates in EMF.

The creation of modeling means took place in four phases. *First*, we had to identify which DSLs are necessary to completely cover the ingredients of our web application and to facilitate the code generation into a JBoss Seam based platform. *Second*, the metamodels for the chosen languages had to be created. *Third*, appropriate generators based on the developed metamodels had to be prepared, mapping our system models to an executable

Figure 4: Sales Scenario Quotation Management UI Excerpt

EJB web application. *Fourth* and last, within the system modeling phase, we created the actual implementation of the Sales Scenario on model level.

In general, our focus lay clearly on the creation of platform independent models (PIM) [MDA03], to not hamper the ability of porting our system models to other platforms. The languages we identified are covering: data structures, data interaction, dialog navigation and presentation, stateful data contexts and behavior. Table 1 summarizes identified languages, indicating their names, domain and architecture parts for generated code.

4.3 Feature-Based Decomposition

This last step in our development of the Sales Scenario deals with the feature-based decomposition of the developed system models. Here, a mapping between problem and solution space has to be established. To date, we are not finished with this last task. However, we are able to explain what challenges we are facing here and which tools were already developed or could at least be applied to tackle these challenges.

To achieve a feature-based model decomposition, each feature has to be bound to all the model elements it corresponds to. Different problems are to be solved here: (1) the actual mapping of one feature to its corresponding model elements, which has to be done with

Language	Domain	JBoss (Seam) Platform Part
businessojects	data structures	EJB EntityBeans
actions	data interaction	persistence manager calls in EJB SessionBeans
concreteactions	refines actions for fine grained data structure interaction	EJB SessionBean method bodies
state	state machines for entity life cycles	EJB SessionBean method bodies and EntityBean attributes
dialog	dialog navigation	JPDL page flow definitions
presentation	dialog layout and events	JSF pages
context	stateful data contexts	JBoss Seam conversation annotations and navigation rules

Table 1: Sales Scenario Modeling Languages

(2) respect to the constraints between features (e.g., if feature A supplies model elements, feature B contributes to), and (3) the dependencies between separate technical models of different architecture parts have to be resolved. As an answer to (1) and (2), Heidenreich et al. developed an Eclipse-based tool in [HW07] within the research project FeasiPLe, which allows the graphical specification of such feature mapping templates. The resulting mapping specification, will then be used as template for appropriate technologies, such as Graph Rewrite Systems (GRS) or Aspect-oriented Modeling (AOM), that are used to integrate corresponding model fragments. In [HW07], the authors consider both collaborative and cross-cutting features with the help of the graph rewrite engine Tiger EMF [Tea07]. However, for the integration of model-crossing dependencies stated with (3), alternative approaches, such as initially proposed in [Loc07] and elaborated in [BL07], might be more appropriate.

5 Challenges in Development

In this section, we explain the main challenges we experienced within the development of our model-driven software product line. As our current key problems we pointed out the balancing of stakeholder interests, the development of modeling languages and the employment of *multiple* DSLs.

5.1 Stakeholder interests

Large software projects are always influenced by many factors, such as hard product requirements, subjective interests and technical constraints. Commonly, these factors are

represented by certain persons, who are involved in the software life cycle - the *stakeholders*. The challenge is, to balance all stakeholders' interests wisely, and not to end up in a tilted project situation, where requirements cannot be satisfied by applied techniques or obligatory milestones cannot be held.

Even though the here presented case study remains a research prototype and its development is more or less easy to manage - mainly because of one local development team - above mentioned stakeholder interests had their effect as well. Mainly the tension between the functional specification in the problem space and the capabilities of the solution space's technical platform led to a trade-off in different requirements. This was especially challenging, because of sequential dependencies between feature specification and platform decision.

What we learned from this challenge is to always consider all stakeholder's demands early in development and arrange a regular, short-term information exchange, as suggested by light weight project management methods or development processes, such as Scrum [Mur04] and Extreme Programming [Bec99] respectively.

5.2 Language Development

Today, to the best of our knowledge, literature lacks of concrete design patterns in MDD, which lead from loose domain descriptions to sophisticated language constructs. This proved, the development of modeling languages to be a difficult task. Besides the challenge to express semantically correct concepts and relationships of a domain, one major problem is to ensure that the created language covers all important use cases of the corresponding domain.

In this regard, our development approach of a model-driven product line validated itself as sensible and useful. Our single variant platform implementation led to an initial understanding of the different parts involved in the Sales Scenario's architecture, and how they correspond to each other. The chosen variant provided major use cases, which could be used to derive main language concepts. Additionally, the platform's own domain specific languages, which some of our metamodels conform to large extents, helped to refine and elaborate our results. One of the main experiences from this challenge is that two important issues in development of modeling languages are *use cases* and an *iterative refinement* procedure. Besides this, common problems in MDD, e.g., round-tripping between metamodel and generator templates, hampered our development progress.

5.3 Employment of Multiple Languages and Models

In general, there were two main approaches possible when creating the modeling means for our product line. The first one was to develop one single language and model accordingly for the whole architecture. This approach could have been well done with respect to a platform centric staging. However, one single language is much more difficult to han-

dle in both development, e.g., due to complex dependencies in generator templates, and application, with regard to maintainability.

We decided to follow the second approach, to split system modeling into multiple languages. This was also motivated (1) through the applied platform and (2) from a separation of concerns point of view. However, the architectural split into multiple modeling languages produces new challenges. When decoupling one solid architecture into different parts, dependencies arise between these parts. The challenge is then to resolve, describe and respect these dependencies during system modeling and generator specification. The literature mainly proposes so called weaving models as solution to such dependency problems. An advanced approach is proposed by Hessellund et al. [HCW07], who applies Prolog to check dependencies between models. An even more advanced approach was proposed in [Loc07] and [BL07]. Here, the use of a system modeling ontology in combination with description logic axioms was proposed to resolve dependencies and provide active guidance when modeling systems with multiple modeling languages.

Fortunately, the employed JBoss Seam platform alleviates this challenge to a certain degree. Here, most interconnections between architecture parts are managed on with String-based references. In JPDL page flows, for instance, concrete identifiers of available JSF view pages are applied for specification.

6 Conclusion and Future Work

Within this paper we introduced a software product line in the area of business applications. For the implementation, we apply different technologies, namely Model-Driven Development with multiple DSLs, Aspect-Oriented Software Development and Software Product Line Engineering. We explained our bottom-up approach to development of this SPL and highlighted the challenges we are facing in development. As stated above, our work is to date not finished. Accordingly, in future work we will report on the feature-based decomposition of the Sales Scenario system models. Additionally, an evaluation of the developed system models concerning their actual platform independence would be interesting. Therefore, the implementation of alternative platform generators by using the same system models had to be performed.

7 Acknowledgment

We would like to thank Patrick Wustmann, Christoph Pohl, Steffen Goebel and other reviewers for their valuable comments on previous versions of this paper.

References

[Bec99] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, October 1999.

[BEZ04] Rüdiger Buck-Emden and Peter Zencke. *mySAP CRM: The Official Guidebook to SAP CRM 4.0*. Galileo Press, 2004.

[BL07] Matthias Bräuer and Henrik Lochmann. Towards Semantic Integration of Multiple Domain-Specific Languages Using Ontological Foundations. In *Proceedings of ATEM'07 co-located with MoDELS 2007, Nashville, Tennessee*, October 2007.

[CI90] E.J. Chikofsky and J.H. Cross II. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software Engineering Journal*, pages 13–17, January 1990.

[Con07a] AMPLE Consortium. AMPLE Research Project. `www.ample-project.net`, Dec 2007.

[Con07b] FeasiPLe Consortium. FeasiPLe Research Project. `http://feasiple.de`, Dec 2007.

[EMF07] Eclipse Modeling Framework (EMF) Project. `http://www.eclipse.org/modeling/emf`, 2007.

[HCW07] Anders Hessellund, Krzysztof Czarnecki, and Andrzej Wasowski. Guided Development with Multiple Domain-Specific Languages. In *Proceedings of MoDELS 2007, Nashville, TN, USA*, volume 4735 of *LNCS*. Springer, September/October 2007.

[HW07] Florian Heidenreich and Christian Wende. Bridging the Gap Between Features and Models. In *Proceedings of AOPLE workshop co-located to GPCE*, 2007.

[JBo07] JBoss Enterprise Application Platform. `http://www.jboss.com/products/platforms/application`, 2007.

[JBS07] JBoss Seam. `http://www.jboss.com/products/seam`, 2007.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute, Carnegie Mellon University Pittsburgh, Pennsylvania 15213, 1990.

[KGT06] Andreas Knopfel, Bernhard Grone, and Peter Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. John Wiley & Sons, May 2006.

[Loc07] Henrik Lochmann. Towards Connecting Application Parts for Reduced Effort in Feature Implementations. In *Proceedings of 2nd IFIP CEE-SET 2007 (WIP Track), Posen, Poland*, October 2007.

[MDA03] Object Management Group (OMG). *MDA Guide Version 1.0.1*, June 2003. OMG document omg/2003-06-01, `http://www.omg.org/docs/omg/03-06-01.pdf`.

[Mur04] Craig Murphy. Adaptive Project Management Using Scrum. *Methods & Tools*, 2004.

[OAW07] openArchitectureWare. `http://www.eclipse.org/gmt/oaw/`, 2007.

[Spr07] Spring Framework. `http://www.springframework.org/`, 2007.

[Tea07] Tiger EMF Transformation Project Team. Tiger EMF Transformation. http://tfs.cs.tu-berlin.de/emftrans/, Dec 2007.