APRiL: A DSL for Payroll Reporting

Xiaorui Zhang[1], Yun Lin[2] and Øystein Haugen[1,3]

[1]SINTEF, Pb. 124 Blindern, NO-0314, Oslo, Norway
[2]Agresso R&D, Pb.4244 Nydalen, NO-0401, Oslo, Norway
[3]University of Oslo, Oslo, Norway

Abstract. The highly diverse payroll reporting structures within and between organizations pose challenges to enterprise information system vendors. Producing the database scripts for customized configuration of payroll reporting has been traditionally a costly manual process. We show how this process can be automated and made less error-prone and more user-friendly by introducing a combination of Model-Driven Development (MDD) and a Domain Specific Language (DSL). This paper addresses the development of Agresso Payroll Reporting Language (APRiL), a DSL to describe payroll structures and hierarchies. The language is supported by tailored tools created with open source technologies on Eclipse. We look at the potential implications of our approach on the development of payroll reporting system, along with its advantages and challenges. We also explore possible improvements and application of our approach in other areas of enterprise information systems.

1 Introduction

As the payroll reporting structures vary considerably within and between organizations, time-consuming customization has been a key challenge to Enterprise Resource Planning (ERP) software vendors. Although Agresso Business World © (ABW, an ERP product from Agresso[1]) provides powerful reporting tools – reporting browser and templates, customization of payroll reporting is still a costly process because the payroll reporting components and structures are difficult to re-use and change from one organization to another. Customized configuration of reporting is sometimes performed by experienced customer consultants producing helping tables with SQL scripts. Besides business domain knowledge, profound SQL knowledge is also required from customers consultants. Agresso is seeking a model-driven approach to deal with the variability of customized payroll reportings.

Agresso Payroll Reporting Language (APRiL), a domain specific language (DSL) to specify payroll reporting structures and to allow automatic generation of the corresponding SQL scripts, is developed by SINTEF and Agresso R&D, Norway. APRiL is designed to provide conceptual view of the payroll reporting structure as well as to reduce human effort to customize the configurations. At first this approach

[1] www.agresso.no

will help the consultants and speed up their customization. Furthermore, it is also feasible that advanced customers could perform configuring themselves. It is one of our goals that the APRiL-based configurations can be understood by customers as well.

2 Current Payroll Reporting Customization Process

Model-Driven Development principles and technologies are not yet fully adopted in Agresso. In the current development process of ABW, models are mainly used for analysis to describe database structures and relationships. The models are created by database specialists based on their understanding of the domain, user requirements and system requirements. The models are reflected as manipulating the database, mostly in the form of C# and C++ code. There is no MDD support for the development of functional and non-functional requirements, which include a lot of complex business logic and rules.

Agresso reporting tools allow users to build complex queries by directly manipulating table columns and relationships. Since the data models of ABW are not modeled for reporting purpose but for the financial process, most of the joints and filters are used for creating a desired report query. From this point of view, it is not easy to abstract out some concepts that are required for reporting, by only looking at table definitions. Table relationships are complex in ABW, and it is hard for users to grasp the business logic and create an advanced reporting in a short time. Thus, Agresso consultants sometimes need to help customers to customize their reporting. Usually the consultants are required to get an understanding of the customers' domain and requirements in a short period of time. Extra programming efforts might also be made for some customized reports.

The payroll module is the most complex and variable component of ABW. A lot of business logic and rules are involved in the payroll domain, and there are numerous variations within and between organizations. For instance, payroll constitutions for a sales representative and for a research scientist could be very different, thus the differences will be reflected in their payroll reporting structures. *Payment* and *Deduction* (P&D) are the basic components of payroll in ABW's payroll management system. They are defined with codes in ABW by Agresso customers. The payroll of each person is a sum of the P&D values. The P&D values can be calculated through the payroll transactions according to the person's status, such as his/her position, working department, working hours, employment type and etc. ABW can retrieve all the calculated P&D values associated with the human resources from the database.

For payroll reporting, organizations need to produce the different views of payroll information, such as personal actual or predicted salary in a certain period, or human resource cost for a certain department or for a project, or average salary of certain groups of employees. Every single P&D value is not that interesting for reporting, but the aggregation of groups of P&Ds. It is impossible to provide a standard way to group P&Ds out of the box by Agresso, because P&D are defined differently in each organization and P&D grouping might also be varying from time to time even for one organization. Customization of payroll reporting is required.

Fig. 1 illustrates the current workflow of customizing ABW payroll reporting module. Agresso consultants firstly get to know the problem domain of customers' in several ways, i.e. go through internal documents or courses. Then the customers provide the consultants with examples of payroll reporting structures as expected output of the customization. Currently, ABW reporting tools do not provide a solution for P&D grouping customization. Therefore the consultants sometimes need manually to produce SQL scripts based on user requirements, domain specific knowledge and experience on ABW reporting tools. The SQL scripts are fed into the ABW database to generate an intermediate customized reporting table. This table serves as input into ABW reporting tool which generates the final representation of payroll reports for end users.

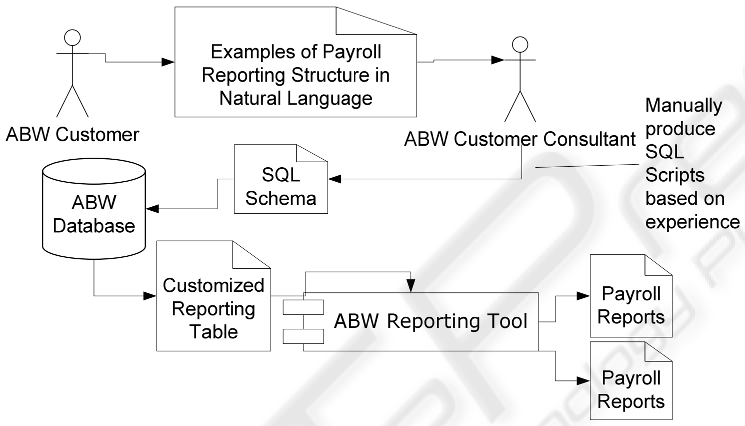

ABW Customer
Examples of Payroll Reporting Structure in Natural Language
ABW Customer Consultant
Manually produce SQL Scripts based on experience
SQL Schema
ABW Database
Customized Reporting Table
ABW Reporting Tool
Payroll Reports
Payroll Reports

Fig. 1. Current workflow of payroll reporting customization.

The current process of customization hinders both the payroll reporting system itself and its developers: for the system, the tailor-made SQL scripts are difficult to reuse since it meets specific needs, and it may be incompatible with later releases of the system; for the customer consultants in Agresso, there is a lack of a conceptual view of payroll data structure in the current ABW and they need to have extensive knowledge of ABW and SQL language to manipulate directly towards the underlying database.

The two main goals of introducing APRiL are: 1) to externalize and visualize the payroll reporting information using models, and 2) to ease the development of ABW reporting tool in handling the complex variability of payroll structure, so as to reduce error-susceptible manual work.

3 Agresso Payroll Reporting Language (APRiL)

APRiL is designed to address the problems in the current customization process of ABW reporting tool. As illustrated in Fig. 2, APRiL serves as an extra abstraction level between payroll reporting and the underlying database. Therefore Agresso consultants would be able to use APRiL to model the specific payroll data structure in its graphical or tree editor rather than manipulate on the database directly. Furthermore, automatic generation of the SQL scripts from the model allows the reduction of human efforts and thus makes the system less error-prone.

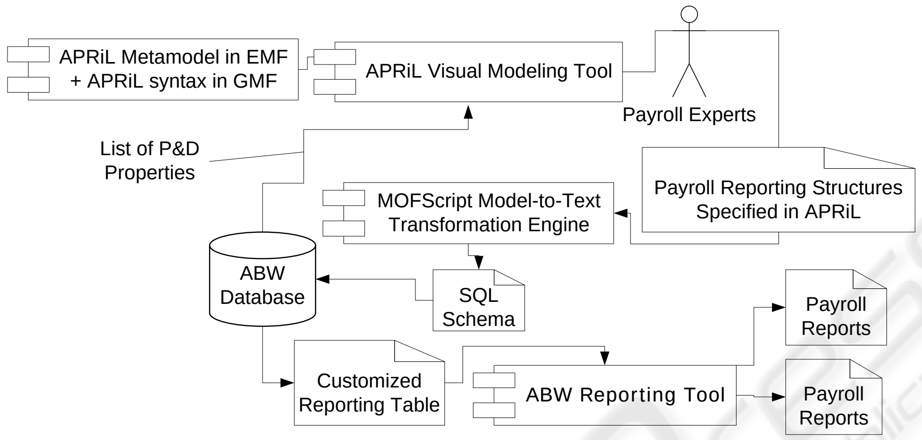

APRiL Metamodel in EMF + APRiL syntax in GMF
APRiL Visual Modeling Tool
Payroll Experts
List of P&D Properties
MOFScript Model-to-Text Transformation Engine
Payroll Reporting Structures Specified in APRiL
ABW Database
SQL Schema
Payroll Reports
Customized Reporting Table
ABW Reporting Tool
Payroll Reports

Fig. 2. Future (APRiL) workflow for payroll reporting customization.

We shall give a brief description of the APRiL language, the implemented supporting tools and the implemented model transformations. An introduction of the technologies used in the development of APRiL is also given.

3.1 APRiL Description

APRiL is defined as an Ecore metamodel in the Eclipse Modeling Framework (EMF) [3]. By specifying a metamodel for APRiL, EMF generates a tree editor automatically for defining payroll reporting data structures. A simplified metamodel of APRiL is given in Fig. 3.

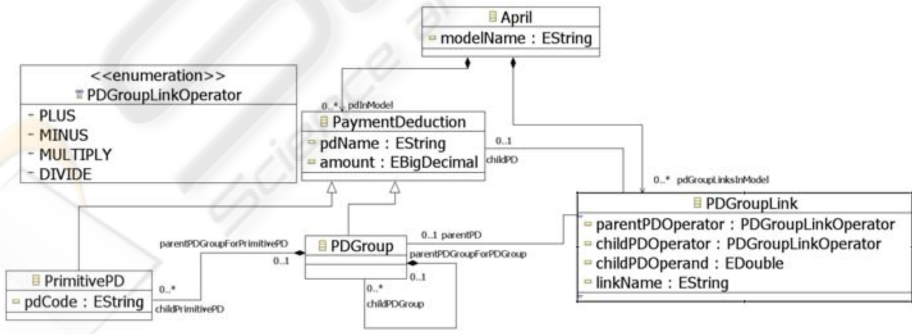


Fig. 3. The APRiL prototype metamodel.

PaymentDeduction (Payment and Deduction, *P&D*) is the main concept in the payroll reporting domain. The concept *PrimitivePD* represents the *P&D* terms that are defined independent of other *P&D*s and all of them have associated *PdCode* as their unique identifiers in ABW Database. *PDGroup* is the *P&D* term that is defined dependent on other *P&D* terms, namely that a *PDGroup* can consist of several *PrimitivePD* as well as other *PDGroup*s. *PDGroup*s are not defined in ABW database, but only for aggregation purpose in payroll reporting. The hierarchical definition of *PrimitivePD*s and *PDGroup*s makes it possible to model the payroll reporting data structure in a more conceptual and intuitive manner. In APRiL, relationships between P&Ds are able to be modeled explicitly by introducing the concept *PDGroupLink* in the metamodel. The connection lines are not only to visualize the relationship between P&Ds, but also to make it possible for users to set the properties *ChildPDOperator*, *ChildPDOperand* and *ParentPDOperator*, in order to configure the constituents of various P&Ds. We shall exemplify these concepts by defining a real payroll reporting case in APRiL, as shown in Fig. 4.

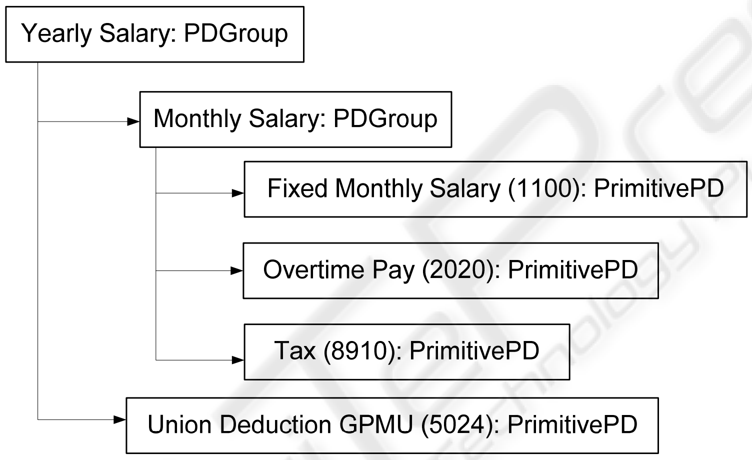

Yearly Salary: PDGroup
Monthly Salary: PDGroup
Fixed Monthly Salary (1100): PrimitivePD
Overtime Pay (2020): PrimitivePD
Tax (8910): PrimitivePD
Union Deduction GPMU (5024): PrimitivePD

Fig. 4. Payroll reporting example.

In this example, *Monthly Salary* as a *PDGroup* is defined as the aggregate of *PrimitivePD*s: *Fixed Monthly Salary*, *Overtime Pay* and *Tax*. *Yearly Salary*, as the topmost *PDGroup*, consists of the aggregate of the *Monthly Salary* for the whole year and a *PrimitivePD*: *Union Deduction GPMU*. The corresponding *PDCode* of each *PrimitivePD* is stated in the parentheses in Fig. 4.

3.2 APRiL Editors

To provide a more intuitive and user-friendly way of editing APRiL models, a graphical editor for APRiL is generated using the Graphical Modeling Framework (GMF) [4] in Eclipse, as shown in Fig. 5.

In GMF, a basic graphical editor based on APRiL metamodel can be created semi-automatically based on the metamodel in three steps: 1) A graphical definition describes the symbols corresponding to the the model elements in the editor; 2) A tool definition defines the tool palette in the editor; 3) A mapping combines all these together. The APRiL editor can then be automatically generated, but there are still opportunities to customize the editor by altering generated editor code manually.

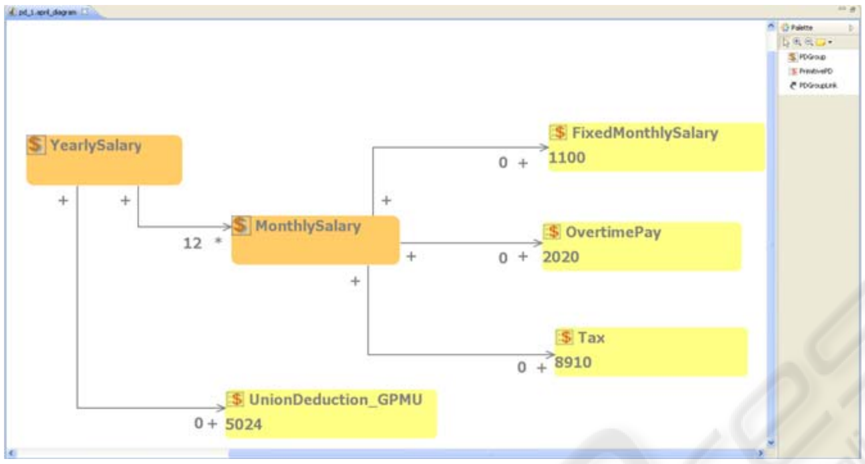


Fig. 5. Payroll reporting example in APRiL graphical editor.

With APRiL tree editor, payroll reporting structures can be modeled by creating an APRiL model element as the container for all other model elements. Relationships between *P&D* terms are defined in the "*properties*" view. While working with the graphical editor, we specify payroll structures by dragging *P&D* terms from the palette onto the APRiL canvas. Relationships between APRiL elements are defined by drawing connection lines between the graphical representations of them. Payroll reporting models created by either the tree editor or the graphical editor (Fig. 5) are equivalent and can be processed by the same code generators.

3.3 APRiL Code Generation

MOFScript [2] is used to generate SQL scripts from APRiL models. MOFScript, as a model-to-text transformation engine, is designed to traverse the model and generate a text such as an SQL script. APRiL models are input to a MOFScript specification that generates SQL scripts which are tailored for specific payroll reporting structures. Fig. 6 shows the generated SQL scripts based on our example.

The following code generation has been done: 1) the PrimitivePDs and PDGroups modeled as tree nodes in the model are transformed into table columns, 2) the PDCodes of the PrimitivePDs are transformed into the value references in SQL case conditions, the properties to specify P&D relationships (ChildPDOperator, ChildPDOperand and ParentPDOperator) are translated as PDGroup constituents' calculations.

```
select t.resource_id, t.client, t.pd,
t.post_id,t.description,t.period,t.voucher_no,t.date_from,t.date_to,t.factor_
att_id,t.dim_value,
t.resource_typ, t.address_type, t.post_id__1 , t.date_from__1 , t.date_to__1,
t.wage_rule__1,  t.scale_id__1,  t.main_position__1, t.parttime_pct__1,
case when pd = '1100' then amount else 0.0 end AS FixedMonthlySalary,
case when pd = '2020' then amount else 0.0 end AS OvertimePay,
case when pd = '8910' then amount else 0.0 end AS Tax,
case when pd = '5024' then amount else 0.0 end AS UnionDeduction_GPMU,
case when pd = '1100' or pd = '2020' or pd = '8910' then amount else 0.0 end
AS MonthlySalary,
((case when pd = '1100' or pd = '2020' or pd = '8910' then amount else 0.0
end) * 12)
+ (case when pd= '5024' then amount else 0.0 end) AS YearlySalary
from datasource t
```

Fig. 6. SQL scripts generated from payroll reporting example.

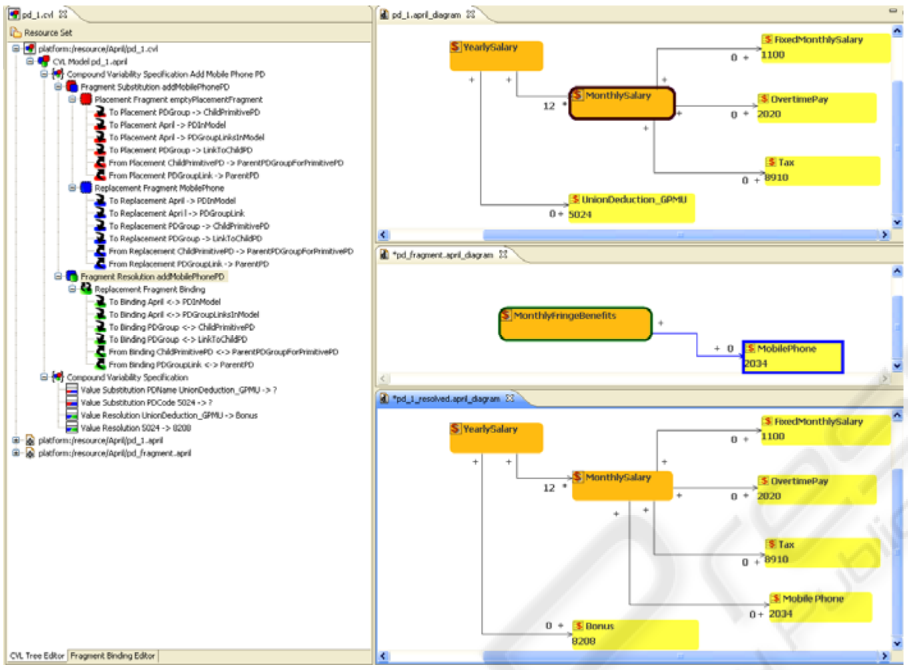
4 Adding Variability Handling to APRiL

We have shown that APRiL is able to reduce manual customization efforts of payroll reporting by applying MDD technologies; however, by identifying the differences and commonalities between APRiL payroll structure models, we are able to automate this process even further.

Haugen et al. [5] propose adding standardized variability handling to DSLs. The basic idea of the approach is to facilitate Software Product Line (SPL) development by separating variability modeling from the base domain modeling. They propose a Common Variability Language (CVL) to model variability while still keeping modeling the base model in DSLs. Several variability models can be defined in CVL for a base model and each variability model can have several resolution models. Each resolution model describes a variant of the base model, which is transformed by executing CVL descriptions.

The payroll structure depicted in the uppermost pane of Fig. 7 can be seen as the base model and the one in the bottom pane as a variant of it. The model variant has one more constituent of *Monthly Salary* called *Mobile Phone*, and *Yearly Salary* has *Bonus* instead of *Union Deduction GPMU*. In order to derive the variant from the base model, we define the variability model that describes the differences between the two and choose resolutions in the CVL editor shown in the left pane of Fig. 7. In general CVL handles variability by categorizing it into value substitution, reference substitution and fragment substitution. As applied here, we use CVL to define an empty fragment of the base model substituted with a fragment in which *Mobile Phone* is included; we also apply value substitution to *PdName* and *PdCode* in order to replace *Union Deduction GPMU* with *Bonus*. The CVL description is then executed using MOFScript and results in a resolved model as shown at the bottom of Fig. 7.

APRiL graphical editor integrates CVL tool support by implementing an interface that provides coloring/highlighting of the elements in the base model according to the selected CVL constructs, as shown in Fig. 7.



```
pd_1.cvl
Resource Set
platform:/resource/April/pd_1.cvl
  CVL Model pd_1.april
    Compound Variability Specification Add Mobile Phone PD
      Fragment Substitution addMobilePhonePD
        Placement Fragment emptyPlacementFragment
          To Placement PDGroup -> ChildPrimitivePD
          To Placement April -> PDInModel
          To Placement April -> PDGroupLinksInModel
          To Placement PDGroup -> LinkToChildPD
          From Placement ChildPrimitivePD -> ParentPDGroupForPrimitivePD
          From Placement PDGroupLink -> ParentPD
        Replacement Fragment MobilePhone
          To Replacement April -> PDInModel
          To Replacement April -> PDGroupLink
          To Replacement PDGroup -> ChildPrimitivePD
          To Replacement PDGroup -> LinkToChildPD
          From Replacement ChildPrimitivePD -> ParentPDGroupForPrimitivePD
          From Replacement PDGroupLink -> ParentPD
      Fragment Resolution addMobilePhonePD
        Replacement Fragment Binding
          To Binding April <-> PDInModel
          To Binding April <-> PDGroupLinksInModel
          To Binding PDGroup <-> ChildPrimitivePD
          To Binding PDGroup <-> LinkToChildPD
          From Binding ChildPrimitivePD <-> ParentPDGroupForPrimitivePD
          From Binding PDGroupLink <-> ParentPD
    Compound Variability Specification
      Value Substitution PDName UnionDeduction_GPMU -> ?
      Value Substitution PDCode 5024 -> ?
      Value Resolution UnionDeduction_GPMU -> Bonus
      Value Resolution 5024 -> 8208
platform:/resource/April/pd_1.april
platform:/resource/April/pd_fragment.april
CVL Tree Editor | Fragment Binding Editor
```

Fig. 7. CVL editor, APRiL editor with CVL integrated showing the base model, the model that includes replacement fragment and the resolved model.

5 Related Work

After decades of experimentation, MDD has been applied in many real-world case studies in different domains which result in significant productivity improvement in product development. DSLs improve software development productivity by raising the abstraction level beyond programming concepts. DSLs only contain language constructs that are relevant concepts in the problem domain world, but not in the code world. It follows the domain semantics, allowing the domain experts perceive themselves work directly with domain concepts. Most domain modeling approaches, such as FAST [8], DFR [9] and GenVoca [1], emphasize a DSL as an important factor to leverage product development in SPL.

Kelly et al. [6] document a case study on how to apply MDD approach to simplify and ease software development of insurance products by defining its tailored DSL. In this case, an insurance expert, a nonprogrammer, draws models to define insurance products in the graphical editor of the DSL, and then the code generator produces the required data and code for a J2EE web site.

Haugen et al. [5] and Svendsen et al. [7] introduce their Train Control Language (TCL) as a DSL of railway signaling for ABB, Norway. TCL allows train experts to compose the station drawing and then generate different representations for various purposes, such as interlocking table, source code for Programmable Logic Controllers (PLCs). In such a safety critical domain, TCL provides more possibilities to ensure security validation. CVL approach is also applied to produce various station variants in this work.

6 Conclusions and Future Work

This paper has presented a DSL for payroll reporting domain (APRiL) and its tool support. The big advantage of the approach is that APRiL provides a clear separation of the concerns in the payroll reporting domain and reporting tool development. For Agresso customer consultants who use APRiL, it makes them able to view the desired data in a conceptual view and to perform less error-prone customization; even for end users who are only specialists in the payroll domain, APRiL makes them capable of customizing ABW reporting tool by themselves without seeking help from the consultants or knowing any technical details of ABW reporting tool.

APRiL has been developed as a prototype and only been used in payroll reporting for the time being. The preliminary experiments have been promising and we are looking into extending the scope of the APRiL approach. We will explore the possibility to proceed with the following work in the future:

- Integration with ABW reporting process. We will explore how to migrate APRiL tool to .NET based ABW system.
- Generalize APRiL a DSL for reporting with various purposes in ABW. Tree hierarchy is a very common structure for reporting, i.e. organizational hierarchy, grouping hierarchy, subtotal and roll-up. Thus we can put some efforts on generalizing APRiL as a Tree Hierarchy Modeling Language for reporting. Then this DSL can be applied to most of the ABW reporting tools for different purposes.

Acknowledgements

The presented work has been developed within the MoSiS project ITEA 2 – IP06035. The APRiL has been developed by SINTEF in cooperation with Agresso R&D, Norway.

References

1. Batory, D., Chen, G., Robertson, E., Wang, T.: Design Wizards and Visual Programming Environments for Genvoca Generators. IEEE Transactions on Software Engineering. 26, 441-452 (2000)
2. Eclipse: Mofscript User Manual. Http://Www.Eclipse.Org/Gmt/Mofscript/Doc/Mofscript-User-Guide.Pdf. (2008)
3. eclipse.org, “Eclipse Modeling Framework,” vol. 2007
4. GMF, “Eclipse Graphical Modeling Framework (Gmf).” http://www.eclipse.org/modeling/gmf/
5. Haugen, O., Møller-Pedersen, B., Oldevik, J., Olsen, G.K., and Svendsen, A., “Adding

Standardized Variability to Domain Specific Languages," SPLC 2008, Limerick, Ireland, (2008)

6. Kelly, S. and Tolvanen, J.-P.: Domain-Specific Modeling: Enabling Full Code Generation. John Wiley & Sons, Inc., (2008)
7. Svendsen, A., Olsen, G.K., Endresen, J., Moen, T., Carlson, E., Alme, K.-J., and Haugen, O., "The Future of Train Signaling," Model Driven Engineering Languages and Systems (MoDELS 2008), Tolouse, France, (2008)
8. Weiss, D., C. T. R: Software Product-Line Engineering. Addison Wesley Longman, (1999)
9. White, S., "Software Architecture Design Domain," 2nd Integrated Design and Process Technology Conference, (1996)