KDM-RE: A Model-Driven Refactoring Tool for KDM

Rafael S. Durelli[1], Bruno M. Santos[2], Raphael R. Honda[2],
Márcio E. Delamaro[1] and Valter V. de Camargo[2]

[1]Computer Systems Department University of São Paulo - ICMC
São Carlos, SP, Brazil.

[2]Computing Departament
Federal University of São Carlos - UFSCAR
São Carlos, SP, Brazil.

`{rdurelli, delamaro}@icmc.usp.br`[1],

`{valter, bruno.santos, raphael.honda}@dc.ufscar.br`[2]

Abstract. *Architecture-Driven Modernization (ADM) advocates the use of models as the main artifacts during modernization of legacy systems. Knowledge Discovery Metamodel (KDM) is the main ADM metamodel and its two most outstanding characteristics are the capacity of representing both i) all system details, ranging from lower level to higher level elements, and ii) the dependencies along this spectrum. Although there exist tools, which allow the application of refactorings in class diagrams, none of them uses KDM as their underlying metamodel. As UML is not so complete as KDM in terms of abstraction levels and its main focus is on representing diagrams, it is not the best metamodel for modernizations, since modifications in lower levels cannot be propagated to higher levels. To fulfill this lack, in this paper we present a tool that allows the application of seventeen fine-grained refactorings in class diagrams. The main difference from other tools is that the class diagrams uses KDM as their underlying metamodel and all refactorings are applied on this metamodel. Therefore, the modernizer engineer can detect "model smells" in these diagrams and apply the refactorings.*

1. Introduction

Architecture-Driven Modernization (ADM) is an initiative which advocates for the application of Model Driven Architecture (MDA) principles to formalize the software reengineering process. According to the OMG the most important artifact provided by ADM is the Knowledge Discovery Metamodel (KDM). KDM is an OMG specification adopted as ISO/IEC 19506 by the International Standards Organization for representing information related to existing software systems. KDM is structured in a hierarchy of four layers; *Infrastructure Layer*, *Program Elements Layer*, *Runtime Resource Layer*, and *Abstractions Layer*. We are specially interested in the *Program Elements Layer* because it defines the Code and Action packages which are widely used by our tool. The Code package defines a set meta-classes that represents the common elements in the source-code supported by different programming languages such as: (*i*) `ClassUnit` and `InterfaceUnit` which represent classes and interface, respectively, (*ii*) `StorableUnit` which illustrates attributes and (*iii*) `MethodUnit` to represent methods, etc. The Action package represent behavior descriptions and control-and-data-flow relationships between code elements.

Refactoring has been known and highly used both industrially and academically. It is a form of transformation that was initially defined by Opdyke [Opdyke, 1992] as "a change made to the internal structure of the software while preserving its external behavior at the same level of abstraction". In the area of object-oriented programming, refactorings are the technique of choice for improving the structure of existing code without changing its external behavior [Fowler et al., 2000]. They have proved to be useful to improve the quality attributes of source code, and thus, to increase its maintainability. It is possible to identify several catalogs of refactoring for different languages and the most complete and influential was published by Fowler in [Fowler et al., 2000]. Nowadays, there are researches been carried out about apply refactoring in model instead of source code[Ulrich and Newcomb, 2010]. Nevertheless, although ADM provides the process for refactoring legacy systems by means of KDM, there is a lack of an Integrated Development Environment (IDE) to lead engineers to apply refactorings as such exist in others object-oriented languages. In the same direction, Model-Driven Modernization is a special kind of model transformation that allows us to improve the structure of the model while preserving its internal quality characteristics. Model-Driven Modernization is a considerably new area of research which still needs to reach the level of maturity attained by source code refactoring [Misbhauddin and Alshayeb, 2012].

In order to enable Model-Driven Modernizations in the context of ADM, refactorings for the KDM metamodel are required. In this context, in a parallel research line of the same group, we developed a catalogue of refactorings for the KDM [Durelli et al., 2014]. We argue that devising a refactoring catalogue for KDM makes this catalogue language-independent and standardized. However, the KDM metamodel was not created with the goal of being the basis for diagrams, as is the case of UML metamodel. Thereby, in order to make possible to apply fine-grained refactoring in the KDM metamodel, it is necessary to devise a way to view the KDM instance graphically. Furthermore, although there exist tools, which allow the application of refactorings in class diagrams, none of them uses KDM as their underlying metamodel. As UML is not so complete as KDM in terms of abstraction levels and its main focus is on representing diagrams, it is not the best metamodel for modernizations, since modifications in lower levels cannot be propagated to higher levels

Hence, the main contribution of this paper is the provision of a plug-in on the top of the Eclipse Platform named **K**nowledge **D**iscovery **M**odel-**R**efactoring **E**nvironment (KDM-RE). This plug-in can be used to lead engineers to apply refactorings in KDM, which are based on seventeen well known refactorings[Fowler et al., 2000]. The IDE as well as the adapted catalogue are based on our experience as model-driven engineering. Also, by using this plug-in the modernizer engineer can visualize the Code package as a UML class diagram, allowing engineers to detect model smells in that diagram. One hypothetical case study was developed in order to exemplify the use of the plug-in.

This paper is organized as followed: Section 2 provides the background to fully understand our *plug-in* - Section 3 depicts information upon the *plug-in* KDM-RE and an case study - in Section 4 there are related works and in Section 5 we conclude the paper with some remarks and future directions.
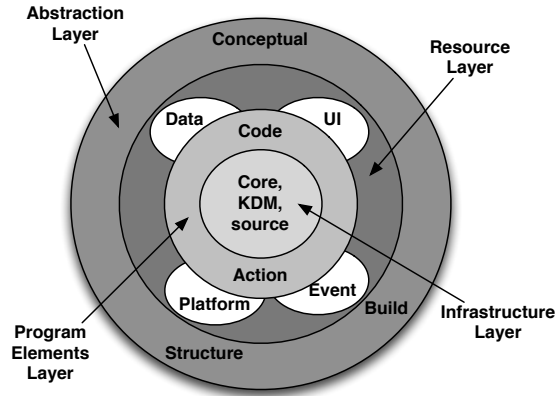
Figure 1. Layers, packages, and separation of concerns in KDM (Adapted from [OMG, 2012])

2. ADM and KDM

OMG defined ADM initiative [Perez-Castillo et al., 2009] which advocates carrying out the reengineering process considering MDA principles. In other words, ADM is the concept of modernizing existing systems with a focus on all aspects of the current systems architecture and the ability to transform current architectures to target architectures by using all principles of MDA [Ulrich and Newcomb, 2010].

To perform a system modernization, ADM introduces Knowledge Discovery meta-model (KDM). KDM is an OMG specification adopted as ISO/IEC 19506 by the International Standards Organization for representing information related to existing software systems. According to Pérez-Castillo et al., [Perez-Castillo et al., 2009] the goal of the KDM standard is to define a meta-model to represent all the different legacy software artifacts involved in a legacy information system (e.g. source code, user interfaces, databases, etc.). The KDM provides a comprehensive high-level view of the behavior, structure and data of legacy information systems by means of a set of meta-models. The main purpose of the KDM specification is not the representation of models related strictly to the source code nature such as Unified Modeling Language (UML). While UML can be used to mainly to visualize the system "as-is", an ADM-based process using KDM starts from the different legacy software artifacts and builds higher-abstraction level models in a bottom-up manner through reverse engineering techniques.

As outlined before, the KDM consists of four abstraction layers: (*i*) *Infrastructure Layer*, (*ii*) *Program Elements Layer*, (*iii*) *Runtime Resource Layer*, and (*iv*) *Abstractions Layer*. Each layer is further organized into packages, as can be seen in Figure 1.

Each package defines a set of meta-model elements whose purpose is to represent a certain independent facet of knowledge related to existing software systems. We are specially interested in the *Program Elements Layer* because it defines the Code and Action packages which are widely used by our catalogue. The Code package defines a set meta-classes that represents the common elements in the source code supported by different programming languages. In Table 1 is depicted some of them. This table identifies KDM meta-classes possessing similar characteristics to the static structure of the source code. Some meta-classes can be direct mapped, such as Class from object-oriented language,

which can be easily mapped to the `ClassUnit` meta-class from KDM.

Table 1. Meta-classes for Modeling the Static Structure of the Source-code

Source-Code Element	KDM Meta-Classes
Class	ClassUnit
Interface	InterfaceUnit
Method	MethodUnit
Field	StorableUnit
Local Variable	Member
Parameter	ParameterUnit
Association	KdmRelationShip

3. Refactoring for KDM by means of KDM-RE

This sections describes KDM-RE. In Figure 2 we depicted the main window of our *plug-in*. For explanation purpose, we highlight two main regions, i.e., ⓐ, and ⓑ. It supports 17 refactorings adapted to KDM. These refactorings are based on some fine-grained refactorings proposed by Fowler [Fowler et al., 2000]. All the refactorings are shown in Table 2. We chose the Fowler's refactorings because they are well known, basic and fine-grained refactorings. Please, not that KDM-RE uses MoDisco[1] once it provides an extensible framework to transform an specific source-code to KDM models. In Figure 2 is presented

Table 2. Refactorings Adapted to KDM

Rename Feature	Moving Features Between Objects	Organing Data	Dealing with Generalization
Rename ClassUnit	Move MethodUnit	Replace data value with Object	Push Down MethodUnit
Rename StorableUnit	Move StorableUnit	Encapsulate StorableUnit	Push Down StorableUnit
	Extract ClassUnit	Replace Type Code with ClassUnit	Pull Up StorableUnit
Rename MethodUnit	Inline ClassUnit	Replace Type Code with SubClass	Pull Up MethodUnit
		Replace Type Code with State/Strategy	Extract SubClass
			Extract SuperClass
			Collapse Hierarchy

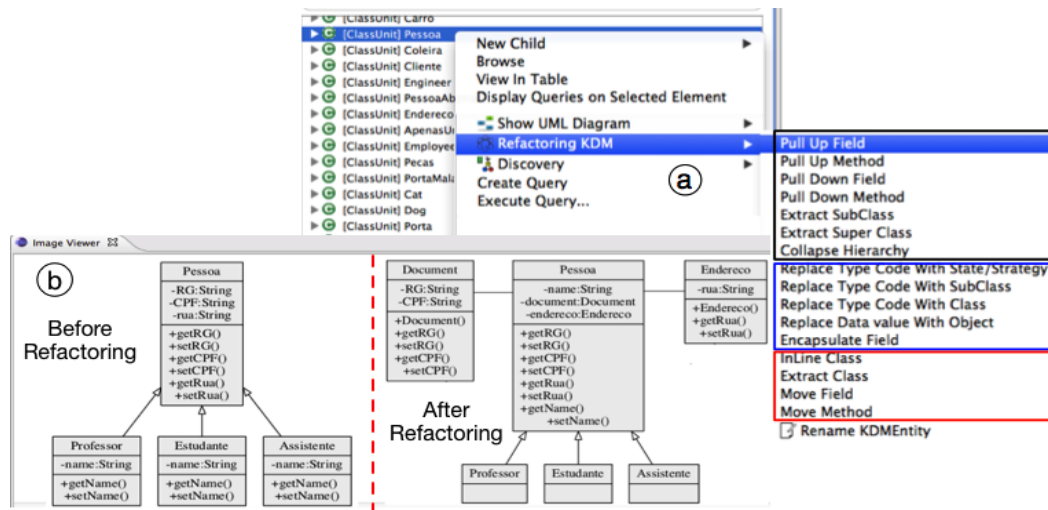**Figure 2. Snippets KDM-RE's Interface**

just a snippet of KDM-RE. Starting from the popup menu named "Refactoring KDM", in

---

[1] http://www.eclipse.org/MoDisco/

this model browser, see Figure 2ⓐ, either the software developer or software modernizer can interact with the KDM model and choose which refactoring must be carried out in the KDM. In the region ⓐ can be seen all 17 refactorings that have been implemented in KDM-RE. For illustration purposes only we drew rectangles to separate the refactorings into three groups. The black rectangle represents refactorings that deal with generalization, the blue rectangle stand for refactorings to organize data and the red one symbolize refactoring to assist the moving features between objects.

The region ⓑ on Figure 2 shows an UML class diagram. This diagram can be used before to apply some refactorings to assist the modernizer to decide where/when to apply the refactorings. This UML class diagram also can be useful as the modernizer performs the refactorings in KDM model. For instance, changes are reproduced on the fly in a class diagram. We claim that the latter use of this diagram is important once it provides an abstract view of the system, hence, the modernizer can visually check the system's changes after applying a set of refactorings. Furthermore, in the context of modernization usually the source-code is the only available artifact of a legacy system. Therefore, creating an UML class diagram makes, both the legacy system and the generated software to have a new type of artifact (i.e., UML class models), improving their documentation.

3.1. Case Study

In this section, we motivate KDM-RE by analyzing one hypothetical case study. This case study is a small part of the university domain. Figure 2 ⓑ (left side) shows a class diagram used for modeling a small part of the university domain. In an university there are several Persons, more specifically Professors, their Assistants, and Students. Each Person has RG, CPF, and address (of type String). Moreover, classes Professor, Assistant, and Student have an attribute name of type String each. The software modernizer or the software developer found out by looking at the UML class diagram (see Figure 2ⓑ left side) this redundantly, i.e., equal attributes in sibling classes. Therefore, he/she must apply the refactoring "Pull Up Field'. Similarly, he/she also found out by looking at the UML class diagram that one class is doing work that should be done by two or more. For example, he/she found that the attributes RG and CPF should be modularized to a class. Similarly, it is necessary to provide more information about they address, such as number, city, country, etc. Therefore, he/she must apply the refactoring "Extract Class" to the attributes "RG", "CPF" and "rua". Due space limitation it is depicted just the extraction of the attributes "RG" and "CPF". The first step is to select the meta-class that he/she identified as a bad smell, i.e., the meta-class to be extracted into a separate one. This step is illustrated in Figure 3(a).

After selecting the meta-class, a right-click opens the context menu where the refactoring is accessible. After the click, the system displays the "RefactoringWizard" to the engineer, Figure 3(b) depicts the Extract Class Wizard. In this wizard, the name of the new meta-class can be set. Also a preview of all detected `StorableUnits` and `MethodUnits` that can be chosen to put into the new meta-class. Further, the engineer can select if either the new meta-class will be a top level meta-class or a nested meta-class. The engineer also can select if the KDM-RE must create instances of `MethodUnits` to represent accessors methods (gets and sets). Finally, the engineer can set the name of the `StorableUnit` that represent the link between the two meta-classes (the old meta-class and the new one). After all of the required inputs have been made, the engineer can click
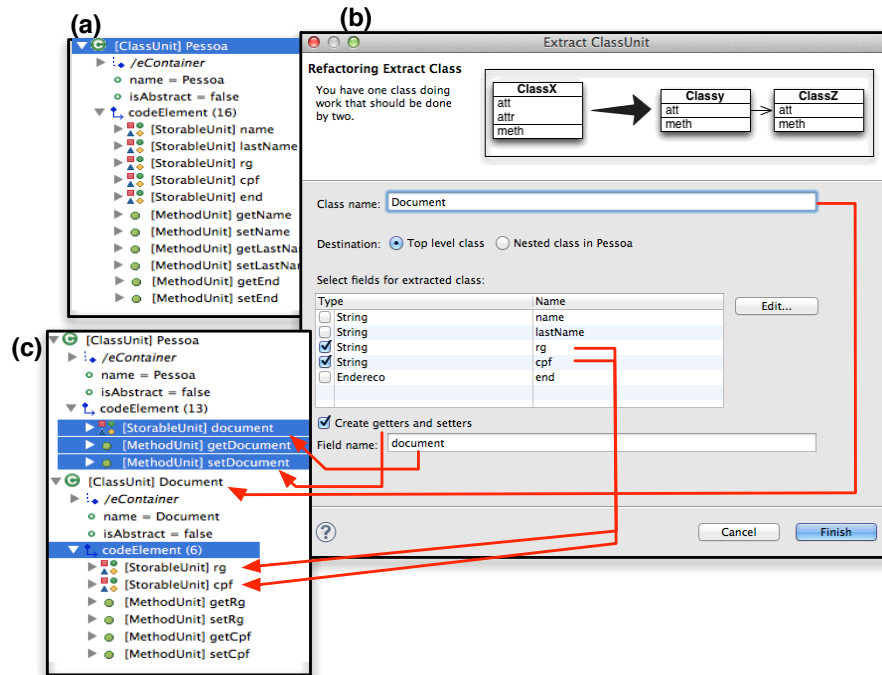
Figure 3. Extract Class Wizard

on the button "Finish" and the refactoring "Extract Class" is performed by KDM-RE.

As can be seen in Figure 3(c) a new instance of `ClassUnit` named "Document" was created - two `StorableUnit` from "Pessoa", i.e., "rg" and "CPF" were moved to the new `ClassUnit` - instances of `MethodUnits` were also created to represent the gets and sets. In addition, the instance of `ClassUnit` named "Pessoa" owns a new instance of `StorableUnit` that represent the link between both `ClassUnits`. Due space limitation the other `StorableUnits` of `ClassUnit` named "Pessoa" are not shown in Figure 3(c). After the engineer realize the refactorings, an UML class diagram is created on the fly to mirror graphically all changes performed in the KDM model, see Figure 2ⓑ right side.

4. Related Work

Borger et al. [Boger and Sturm, 2002] developed a plug-in for the CASE tool ArgoUML that support UML model-based refactorings. The refactoring of class, states and activities is possible, allowing the user to apply refactorings that are not simple to apply at source code level. Van Gorp et al. [Gorp et al., 2003] proposed a UML profile to express pre and post conditions of source code refactorings using Object Constraint Language (OCL) constraints. The proposed profile allows that a CASE tool: (*i*) verify pre and post conditions for the composition of sequences of refactorings; and (*ii*) use the OCL consulting mechanism to detect bad smells such as crosscutting concerns. Reimann et al. [Reimann et al., 2010] present an approach for EMF model refactoring. They propose the definition of EMF-based refactoring in a generic way. Another approach for EMF model refactoring is presented in [Thorsten Arendt, 2013]. They propose EMF Refactor [2], which is a

---

[2]http://www.eclipse.org/emf-refactor/

new Eclipse incubation project in the Eclipse Modeling Project consisting of three main components. Besides a code generation module and a refactoring application module, it comes along with a suite of predefined EMF model refactorings for UML and Ecore models.

5. Concluding Remarks

In this paper is presented the KDM-RE which is a *plug-in* on the top of the Eclipse Platform to provide support to model-driven refactoring based on ADM and uses the KDM standard. More specifically, this *plug-in* supports 17 refactorings adapted to KDM. These refactorings are based on some fine-grained refactorings proposed by Fowler [Fowler et al., 2000]. As stated in the case study the engineer/modernizer by using KDM-RE can apply a set refactorings in a KDM. Also, on the fly the engineer can check all changes realized in this KDM replicated into a class diagram - the engineer can visually verify the system's changes after applying a set of refactorings. In addition, usually the source code is the only available artifact of the legacy software. Therefore, creating an UML class diagram makes, both the legacy software and the generated software to have a new type of artifact (i.e., UML class models), improving their documentation. Also, we claim that as we have defined all refactoring based on the KDM, they can be easily reused by others researchers.

It is important to notice that the application of refactorings in UML class diagrams is not a new research as stated before. However, all of the works we found on literature perform the refactoring directly on the UML metamodel. Although UML is also an ISO standard, its primary intention is just to represent diagrams and not all the characteristics of a system. As KDM has been created to represent all artifacts and all characteristics of a system, refactorings performed on its finer-grained elements can be propagated to higher level elements. This propitiates a more complete and manageable model-driven modernization process because all information is concentrated in just one metamodel. In terms of the the users who uses modernization tools like ours, the difference is not noticeable; that is, whether the refactorings are performed over UML or KDM. However, there are two main benefits of developing a refactoring catalogue for KDM. The first one is in terms of reusability. Other modernizer engineers can take advantage of our catalogue to conduct modernizations in their systems. The second benefit is that, unlikely UML, a catalogue for KDM can be extended to higher abstractions levels, such as architecture and conceptual, propitiating a good traceability among these layers.

We believe that KDM-RE makes a contribution to the challenges of Software Engineering which focuses on mechanisms to support the automation of model-driven refactoring. Future work involves implementing more refactorings and conducting experiments to evaluate all refactorings provided by KDM-RE. Doing so, we hope to address a broader audience with respect to using, maintaining, and evaluating our tools. Currently, KDM-RE generates only class diagrams to assist the modernization engineer to perform refactorings, however, as future work, we intend to: (*i*) extend this computational support to enable the achievement of other diagrams, e.g., the sequence diagram, (*ii*) perform structural check of the software after the application of refactorings; and (*iii*) carry out the assessment tool, as well as refactorings proposed by controlled experiments. A work that is already underway is to check how other parts of the highest level of KDM are affected after the application of certain refactorings. For example, assume that there are two pack-

ages P1 and P2. Suppose there is a class in P1, named C1, and within the P2 there is a class named C2. Assume that C1 owns an attribute A1 of the type C2., i.e., there is an association relationship between these classes of different packages. P1 and P2 represent architectural layers, i.e., P1 = Model and P2 = View. Thus, the relationship that exists is undesirable. When we make a fine-grained refactoring such as moving the attribute A1 of the class C1, it should be reflected to the architectural level, eliminating the existing relationship between the two architectural layers.

6. Acknowledgements

References

Boger, M. and Sturm, T. (2002). Tool-support for model-driven software engineering. In *In Proceedings of Practical UML-Based Rigorous Development Methods*.

Durelli, R. S., Santibáñez, D. S. M., Delamaro, M. E., and Camargo, V. V. (2014). Towards a refactoring catalogue for knowledge discovery metamodel. In *IEEE 15th International Conference on Information Reuse and Integration (IRI)*.

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (2000). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

Gorp, P. V., Stenten, H., Mens, T., and Demeyer, S. (2003). Towards automating source-consistent uml refactorings. In *International Conference on UML - The Unified Modeling Language*, pages 144–158. Springer.

Misbhauddin, M. and Alshayeb, M. (2012). Model-driven refactoring approaches: A comparison criteria. In *Sofware Engineering and Applied Computing (ACSEAC), 2012 African Conference on*.

OMG (2012). Object Management Group (OMG) Architecture-Driven Modernisation. Disponível em: *http://www.omgwiki.org/admtf/doku.php?id=start*. (Acessado 2 de Agosto de 2012).

Opdyke, W. F. (1992). *Refactoring Object-Oriented Frameworks*. Ph.D. Thesis, University of Illinois.

Perez-Castillo, R., de Guzman, I. G.-R., Avila-Garcia, O., and Piattini, M. (2009). On the use of adm to contextualize data on legacy source code for software modernization. In *Proceedings of the 2009 16th Working Conference on Reverse Engineering*, WCRE '09, pages 128–132, Washington, DC, USA. IEEE Computer Society.

Reimann, J., Seifert, M., and Abmann, U. (2010). Role-based generic model refactoring. In *In ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013)*. Springer.

Thorsten Arendt, Timo Kehrer, G. T. (2013). Understanding complex changes and improving the quality of uml and domain-specific models. In *In ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013)*.

Ulrich, W. M. and Newcomb, P. (2010). *Information Systems Transformation: Architecture-Driven Modernization Case Studies*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.