Good Practices as a Quality-Oriented Modeling Assistant

Vincent Le Gloahec, Régis Fleurquin, Salah Sadou

Good Practices as a Quality-Oriented Modeling Assistant

Vincent Le Gloahec
Alkante SAS, France
Email: v.legloahec@alkante.com

Regis Fleurquin
INRIA/Triskell,
Campus Universitaire de Beaulieu, France
Email: regis.fleurquin@irisa.fr

Salah Sadou
Valoria laboratory,
Université de Bretagne-Sud, France
Email: salah.sadou@univ-ubs.fr

***Abstract*—In order to ensure the quality of their software development, companies incorporate best practices from recognized repositories or from their own experiences. These good practices are often described in software quality manuals that, in this form, do not guarantee their implementation. In this paper, we propose a framework for the implementation of best practices. We treat the case of modeling activities because they are becoming the main activity of software development processes. Our framework enables on the one hand to describe the good practices and on the other hand to check their application by the designers. We present an implementation of our framework in the Eclipse platform and some examples of use of our approach in the modeling of UML Class diagrams.**

I. Introduction

Good Practices (GPs) are proven processes or techniques that organizations (or persons) have found to be productive and useful to ensure a good level of production quality. Some GPs are found in research papers, reports, guides, books or standards. In the current context, modeling becomes the main activity of the software development process using more and more domain-specific modeling languages (DSML). Software modeling activity, like any other activity, can take advantage of GPs. At every step of the modeling activity, GPs can provide help to build good models faster. As stated by Gratton and Ghoshal [1] and process improvement programs such as CMMI-DEV [2]: *the capacity of an enterprise to prosper is based on its ability to capture, use, and assess good practices.*

Due to a lack of an adequate formalism to document their knowledge, companies that try to capitalize on their knowledge for a particular DSML and activity end up making use of informal documents, often incomplete, poorly referenced, and sometimes scattered. This leads to an inadequate and an ineffective use and sometimes loss of Good Modeling Practices (GMPs). Moreover, documenting GMPs by themselves is not sufficient. If the documented GMPs are not supported by modeling tools, their implementation still depends only on developer skills and good will. Unfortunately, most of modeling tools provide no means to support GMPs. Some tools enforce modeling styles using for example the OCL language. Few modeling tools allow some form of parameterization through adhoc scripting languages (e.g., the J language for the Objecteering platform). Some work [3], [4], and [5] also propose grafting on modeling tools, such as Eclipse or Rational, with extensions allowing the interception of the designer's actions and inspecting the tool's information system to detect some kinds of methodological inconsistencies. But, their expressiveness is still limited, depending on the format of the tool's information system. Consequently, the GMPs expressed using a given modeling language under a given modeling tool are unusable and lost when switching to another modeling tool that supports the same modeling language.

In this paper, we propose tackling the problem of documentation and enactment of GMPs. From identified GMPs common characteristics, we propose a list of requirements to be fulfilled by a language dedicated for GMP specification. We then introduce the abstract syntax and semantic of our GMPs specification language, called GooMod (Sect. 2). This language is usable for GMPs associated with any type of DSML whose abstract syntax is described in a MOF model. Using the Eclipse platform, we developed an editor for this language allowing thus GMP description and a GMP checker that work with any modeler based on the Eclipse platform (Sect. 3). This tool has been designed as a modeling assistant to help designers to build quality models.

II. Good Modeling Practices Description Language

Based on a wide range of well-known good practices, we have identified a list of several required properties for a language that allows the description of good modeling practices. In this section we first show which properties have been identified and then we describe the abstratc syntax and semantic of our GMPs description language, called *GooMod*.

A. Identified Properties

To identify the required properties for a GMPs description language, we conducted a study on best practices in modeling activity. Through literature, we observe three types of good modeling practices: those that are concerned only with the form (style) of produced models, those that describe the process of their design, and those that combine both. As the third type is only a combination of the first two, we limit our study to examples covering the former types. For the first type we found that the best practices for Agile Modeling given in [6] are good examples. In [7], Ramsin and Paige give a detailed review of object-oriented software development

methods. From this review we extracted properties concerning the process aspect of BPs. Here are the identified properties:

1) Identification of the context: to identify the context of a GP, the language must be able to check the state of the model to determine whether it is a valid candidate for the GP or not.

2) Goal checking: to check that a GP has been correctly applied on a model, the language must be able to check that the status of the latter conforms to the objective targeted by the GP. At the GP description language level, this property highlights the same need as the one before.

3) Description of collaborations: a CASE tool alone is able to achieve some parts of a GP's checking. However, some GP cannot be checked automatically and the tool would need the designer's opinion to make a decision. In case of alternative paths, sometimes the tool is in a situation where it cannot determine the right path automatically. Thus, a GP description language should allow interactions with the designer.

4) Process definition: a process defines a sequence of steps with possible iterations, optional steps, and alternative paths. A GP description language should allow processes to be defined with such constructs.

5) Restriction of the modeling language: several good practices based on modeling methodologies suggest a gradual increase in the number of manipulated concepts (e.g., each step concerns only a subset of the modeling language's concepts). Thus, the GP description language should allow the definition of this subset for each step.

The documentation of a GP associated with a modeling language requires a description that is independent of any tool; indeed, a GP is specific to a language. It describes a particular use of its concepts. It should not assume modes of interaction (buttons, menus, etc.) used by an editor to provide access to these concepts. Therefore, a GP must be described in a way that can be qualified as a Platform Independent Model (PIM) in Model-Driven Engineering (MDE) terminology. Ignoring this rule would lead Quality Assurance Managers (QAM) to redocument the GPs at each new version or tool change. The language we introduce in this section, called *GooMod*, offers a way to document GPs independently of any editor (language of PIM level).

B. Abstract syntax of the GooMod Language

The abstract syntax of the *GooMod* language is given in Fig. 1. The process part of a GP is described as a weakly-connected directed graph. In this graph, each vertex represents a coherent modeling activity that we call a step. Arcs (identified by *Bind* in our metamodel) connect pairs of vertices. Loops (arcs whose head and tail coincide) are allowed, but not multi-arcs (arcs with the same tail and same head).

A step is associated with four elements: its context, its associated modeling style, the set of language concepts usable during its execution, and a set of actions. The context is a first-order formula evaluated on the abstract syntax graph of the input model before the beginning of the step. We call this formula a pre-condition. The modeling rule is a first-order formula that is evaluated on the abstract syntax graph

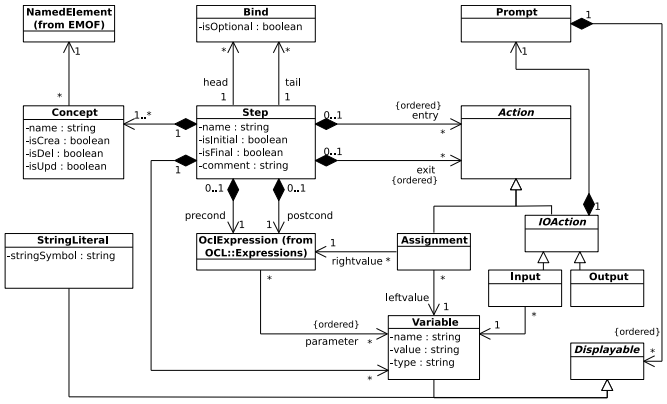

Fig. 1: GooMod Meta-model

of the current model to allow designer to leave from the step. We call this formula a post-condition. The set of the usable language concepts is a subset of the non-abstract metaclass of the asbtract syntax (described in a MOF Model) of the targeted modeling language.

Because some GP require the establishment of a collaboration between the system and the designer, we have included the ability to integrate some actions at the beginning (Entry) and/or at the end (Exit) of a step. The possible actions are: output a message, an input of a value and the assignment of a value to a local variable. Indeed, at each step, it may be necessary to have additional data on the model that only the designer can provide (goal of *Input* action). Conversely, it is sometimes useful to provide designers information that they can not deduce easily from the visible aspect of the model but the system can calculate (goal of *Output* action). Hence, the usefulness of variables associated with steps to hold this data. Thus, actions allow interaction with the designer using messages composed of strings and calculated values.

Steps are also defined by two boolean properties: *isInitial* and *isFinal*. At least one step is marked as initial and one as final in a graph. Finally, an arc can be marked as optional, meaning that its head step is optional.

C. Semantic of the GooMod Language

Semantically, the graph of a GP is a behavior model composed of a finite number of states, transitions between those states, and some *Entry*/*Exit* actions. Thus, a GP is described as a finite and deterministic state machine with states corresponding to the steps of the process part of a GP.

At each step, the elements that constitute it, are used as follows:

1) Before entering the step, the preconditions are checked to ensure that the current model is in a valid state compared with the given step. Failure implies that the step is not yet allowed.
2) If the checking succeeds, then before starting model edits a list of actions (*Entry Action*), possibly empty, is launched. These actions initialize the environment

associated with the step. This may correspond to the initializing of some local variables or simply interactions with the designer.
3) A given step can use only its associated language concepts. In fact, each concept is associated with use type (create, delete, or update).
4) When the designer indicates that the work related to the step is completed, a list of actions (*Exit Action*) will be launched to prepare the step's environment to this end. With these actions the system interacts with the designer to gain data that it can not extract from the model's state.
5) Before leaving the step, the postconditions are checked to ensure that the current model is in a valid state according to the GP rules.

Leaving a step, several transitions are possible. These transitions are defined by the *Binds* whose tail is this step. A transition is possible only if the precondition of the head step of the concerned *Bind* is verified by the current state of the model. If several next steps are possible, then the choice is left to the designer. A *Bind* can also be defined as optional. In this case, its tail step becomes optional through the transition it defines. Thus, the possible transitions of the tail step are added to those of the optional step, and so on.

III. An Implementation Example of the GooMod Language

To implement the GooMod language, we developed a complete platform for the management of GMPs, starting from their definition at the PIM level up to their enactment at the PSM level. This section describes both levels and their associated tools.

A. The GooMod Language PIM-level Implementation

In our platform, we first propose a PIM level implementation of the GooMod language described in the previous sections. Thus, GMPs described with GooMod are independent of any CASE tool.

The *GMP Definition Tool* is designed for QAM in charge of the definition of GMPs that should be observed in a company. Our first graphical editor, designed using the Eclipse Graphical Modeling Framework[1] (GMF), allows the representation of GMPs in the form of a process. Such a process is represented by a path in a graph. Each node of the path is a step. At each step of the process, the GMP Definition tool allows for the selection of a subset of manipulated concepts from the target language (the one used by modelers to design their systems, UML for example), as well as the definition of a set of OCL pre- and postconditions, as well as entry and exit actions.

B. PSM-level of the Proposed Platform

At the PSM level, the platform is composed of two parts:

1) GMP Activation Tool: it aims to link a modeling process defined with the GMP Definition tool to a target modeling tool. It applies a given process during the final modeling stage to control the enforcement of GMP.

[1]See Eclipse Modeling Project (http://www.eclipse.org/modeling)

2) Targeted Modeling Tool: this is the end-user modeling tool where the GMP will be performed. This tool is not intended to be modified or altered directly, but will be controlled by an external plugin, which in our case is the GMP Activation Tool.

In our case, the targeted modeler is the GMF UML2.0 editor embedded in the Eclipse Model Development Tools (MDT) project. However, our approach is not limited to UML modelers or more generally speaking to the UML language. Indeed, as the GMP Definition tool uses a target meta-model (the one of the targeted modeling tool) as its input, the definition of the OCL constraints and the selection of editable concepts will be dedicated to the modeling language (any kind of MOF-compliant DSML).

The GMP Activation tool has been designed to interact with GMF-generated editors. If the first feature of GMP Activation is to enact a process and check the elaboration of models, the second feature consists of controlling some parts of the targeted CASE tool. At each step of modeling, only the editable concepts of the current step are active. The Activation tool dynamically activate/deactivate GMF creation tools according to the editable concepts allowed for each step. With this approach, we are able to control any GMF editor within the Eclipse platform.

C. Example: UML Class Diagram Modeling with GooMod

To demonstrate how our proposed platform could be used, consider the following situation: in a company, some developers are not used to UML and when they need to design UML Class diagrams, it takes a significant amount of time and it also often leads to incorrect diagrams. To tackle this problem, the QAM of the company has already identified some good practices that she wants the developers' follow.

With the help of the GooMod language and the GMP Definition tool, the QAM has defined its own methodology using an OMT-based process and for each step of the process, he associated some pre- and post-conditions, entry and exit actions, and editable concepts. A brief description of the method is given hereafter. The process part of the method consists of five steps, with their associated context, actions, and concepts. For the sake of brevity, several details such as OCL pre- and postconditions have been omitted:

1) Create packages: during this initial step, only packages should be created. So, only the concept *Package* from the UML metamodel will be available.
2) Create classes: at this step, only the concept *Class* will be made available. Before to enter this modeling step, it must be verified that at least one package has been added to the diagram. To do so, a pre-condition is added: `Package.allInstances()->notEmpty()`.
3) Adding attributes and operations: for each class, designers should consider adding attributes and maybe some operations if needed. The selected UML concepts for this step are *Property* and *Operation*.
4) Adding relationships: as this modeling step is not necessarily required, the transition that leads to this step is

marked as optional; thus, designers are allowed to skip this step and reach the final step. Designers are allowed to create *Association*, *Generalization*, *Dependency*, and *Realization* relationships.

5) Refactoring: in the final step of modeling, all the concepts manipulated in the preceding steps are made available, thus enabling the refactoring of all elements in the diagram. As a last verification, some metrics can been defined as post-conditions, such as for instance that there should not be more than 20 classes per package:
```
Package.allInstances()->forAll( p:Package
| p.ownedElement->select( e:Element |
e.oclIsTypeOf(Class) )->size() <= 20 ).
```

Even if this example is rather straightforward, its purpose is to show how QAM can use the GooMod language to define, in an independent manner of any tool, their practices. The process part of the presented method has bee build using the graphical notation of the GMP Definition tool (steps and binds), whereas non-graphical properties of the language are defined with the help of editing assistants (pre- and postconditions, entry and exit actions, and editable concepts). Once a good practices description is finished, the model is stored in a XMI format to be used by other tools.

In this example, the UML modeler used by the designers is the standard UML2 modeler provided in the Eclipse platform. As this editor is based on GMF, our GMP Activation tool is able to automatically adapt itself to this modeler. When developers start designing UML Class diagrams, they first load the GooMod model defined by the QAM, and launch the controlled editing process. The full case study presented in this paper is provided as a screencast at http://www-valoria.univ-ubs.fr/SE/goomod.

IV. Related Work

Several works suggest introducing processes and tools that facilitate storage, sharing, and dissemination of GP within companies (e.g. [8], [9]). They advocate in particular the use of real repositories allowing various forms of consultation, thus facilitating research and discovery of GPs. However, the GP referred to by these systems are documented and available only through textual and informal. It is therefore impossible to make them productive in order to control their use within CASE tools.

To the best of our knowledge, there is no other work on the definition of rigorous languages for documenting the modeling GP. However this field can benefit from works concerned with method engineering [10] and software development process [11]. With CASE tools, several works suggest encouraging, even requiring, the respect of certain GP. The domain that had produced good results in recent years is the one that focuses on the automation of GP concerning detection and correction of inconsistencies. These include, in particular, the work presented in [5], [4], and [12]. They propose adding extensions to modeling tools, such as Eclipse or Rational, that are able to intercept the actions of developers and inspect the information system of the tools in order to detect the occurrence of certain types of inconsistency. These works involve GP with a much smaller granularity than those we are dealing with.

V. Conclusion

The quality of a software system is highly influenced by the quality of the process used to develop and maintain it. The belief in this premise is seen worldwide in quality movements, as evidenced by the ISO/IEC body of standards or by the CMMI process improvement maturity model. The latter model describes an evolutionary improvement path, from immature processes to disciplined, mature processes with improved quality and effectiveness. All our works tackle the requirements emphasized in those quality models, in a limited but more and more important scope in today development processes: modeling activities.

This paper is a first step in this direction. A language (GooMod), that can be used by quality engineers, allowing the rigorous documentation of a high variety of good modeling practices in a formalism independent of any "CASE Tools" is described. To control and facilitate the use of these practices in projects, the framework we proposed is easily adaptable for any editor based on GMF (because most of the used modeling editors are Eclipse plug-ins). Thus, designers build models that comply with these good practices. Moreover, the GooMod language is powerful enough to define practices dedicated to collect valid data at some points of the modeling activities.

References

[1] L. Gratton and S. Ghoshal, "Beyond best practices," *Sloan Management Review*, no. 3, pp. 49–57, 2005.

[2] SEI, "CMMI for Development, Version 1.2," Software Engineering Institute, Tech. Rep., 2006. [Online]. Available: http://www.sei.cmu.edu

[3] A. G. Cass and L. J. Osterweil, "Process support to help novices design software faster and better," in *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 2005, pp. 295–299.

[4] A. Egyed, "Uml/analyzer: A tool for the instant consistency checking of uml models," *Software Engineering. ICSE 2007. 29th International Conference on*, pp. 793–796, 2007.

[5] X. Blanc, I. Mounier, A. Mougenot, and T. Mens, "Detecting model inconsistency through operation-based model construction," in *ICSE '08: Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 511–520.

[6] S. W. Ambler, *The Elements of UML(TM) 2.0 Style*. New York, NY, USA: Cambridge University Press, 2005.

[7] R. Ramsin and R. F. Paige, "Process-centered review of object oriented software development methodologies," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1–89, 2008.

[8] G. Fragidis and K. Tarabanis, "From repositories of best practices to networks of best practices," *Management of Innovation and Technology, 2006 IEEE International Conference on*, pp. 370–374, 2006.

[9] L. Zhu, M. Staples, and I. Gorton, "An infrastructure for indexing and organizing best practices," in *REBSE '07: Proceedings of the Second International Workshop on Realising Evidence-Based Software Engineering*. IEEE Computer Society, 2007.

[10] B. Henderson-Sellers, "Method engineering for OO systems development," *Commun. ACM*, vol. 46, no. 10, pp. 73–78, 2003.

[11] OMG, "Software Process Engineering Meta-Model, version 2.0 (SPEM2.0)," Object Management Group, Tech. Rep., 2008. [Online]. Available: http://www.omg.org/docs/formal/08-04-01.pdf

[12] A. Hessellund, K. Czarnecki, and A. Wasowski, "Guided development with multiple domain-specific languages," in *MoDELS*, 2007, pp. 46–60.