Information and Software Technology 58 (2015) 338–354

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A framework to identify primitives that represent usability within Model-Driven Development methods

Jose Ignacio Panach [a,*], Natalia Juristo [b], Francisco Valverde [c], Óscar Pastor [c]

[a] Escola Tècnica Superior d'Enginyeria, Departament d'Informàtica, Universitat de València, Avenida de la Universidad, s/n, Burjassot, 46100 Valencia, Spain
[b] Universidad Politécnica de Madrid, Campus de Montegancedo, 28660 Boadilla del Monte, Spain
[c] Centro de Investigación en Métodos de Producción de Software – ProS, Universitat Politècnica de València, Camino de Vera, s/n, 46022 Valencia, Spain

ARTICLE INFO

Article history:
Received 20 January 2014
Received in revised form 4 July 2014
Accepted 5 July 2014
Available online 22 July 2014

Keywords:
Model-Driven Development
Usability
Conceptual model

ABSTRACT

Context: Nowadays, there are sound methods and tools which implement the Model-Driven Development approach (MDD) satisfactorily. However, MDD approaches focus on representing and generating code that represents functionality, behaviour and persistence, putting the interaction, and more specifically the usability, in a second place. If we aim to include usability features in a system developed with a MDD tool, we need to extend manually the generated code.
Objective: This paper tackles how to include functional usability features (usability recommendations strongly related to system functionality) in MDD through conceptual primitives.
Method: The approach consists of studying usability guidelines to identify usability properties that can be represented in a conceptual model. Next, these new primitives are the input for a model compiler that generates the code according to the characteristics expressed in them. An empirical study with 66 subjects was conducted to study the effect of including functional usability features regarding end users' satisfaction and time to complete tasks. Moreover, we have compared the workload of two MDD analysts including usability features by hand in the generated code versus including them through conceptual primitives according to our approach.
Results: Results of the empirical study shows that after including usability features, end users' satisfaction improves while spent time does not change significantly. This justifies the use of usability features in the software development process. Results of the comparison show that the workload required to adapt the MDD method to support usability features through conceptual primitives is heavy. However, once MDD supports these features, MDD analysts working with primitives are more efficient than MDD analysts implementing these features manually.
Conclusion: This approach brings us a step closer to conceptual models where models represent not only functionality, behaviour or persistence, but also usability features.

1. Introduction

The Model-Driven Development (MDD) paradigm [20] states that all the analysts' effort must be gathered in the conceptual model and the system is implemented by means of transformation rules that can be automated. In other words, the MDD paradigm distinguishes between conceptual models (where analysts work) and the code that implements the system (which can be generated with as much automation as possible from the conceptual model).

Nowadays, there are several tools which implement the MDD paradigm, such as WebRatio [2], UWE [19], NDT [9] and OO-Method [29,28], among others. All these tools are very powerful to represent and generate the system functionality, behaviour and persistency by means of conceptual models. However, in most MDD methods, there is a lack of expressiveness to represent usability features [1,24]. Nowadays, if these features are to be included in systems developed by these MDD methods, the generated code needs to be changed manually. These manual changes involve some disadvantages:

- Changes in the code can be inconsistent with the characteristics expressed in the conceptual model.
- Every time we regenerate the code from the conceptual model, the manual changes to the code must be applied.
- Understanding the code to enhance the system usability can be difficult for the analyst.

* Corresponding author.
E-mail addresses: joigpana@uv.es (J.I. Panach), natalia@fi.upm.es (N. Juristo), fvalverde@pros.upv.es (F. Valverde), opastor@pros.upv.es (Ó. Pastor).

http://dx.doi.org/10.1016/j.infsof.2014.07.002
0950-5849/

In order to overcome all these problems, we propose including usability features in a conceptual model similarly to what it is currently done with functionality, behaviour and persistency in most MDD methods [18,34]. This proposal is a step forward to incorporate software systems characteristics not combined to date in MDD methods. Note that the target audience of our proposal are analysts that work frequently with MDD tools, since they are the persons that tweak the code to support usability features nowadays. Our approach does not deal with benefits or disadvantages of the MDD paradigm versus a traditional method or how to improve the learnability of novice users with MDD tools.

In the past, many SE authors considered usability as a non-functional requirement [7]. Recently, however, some authors have identified several usability features that are strongly related to functionality [4,11,16]. We focus on these features, since they affect not only interface but also the architecture, and are hard to deal with unless they are considered from the early stages of development. The contribution of our work is the definition of a process to represent functional usability features in a conceptual model in such a way that a model compiler can automatically generate their code.

The benefits of incorporating functional usability features in a MDD method through conceptual primitives are [35,36]:

- Unambiguously defined functional usability features. This is an essential characteristic for performing model-to-model and model-to-code transformations.
- Reduced development effort with respect to including usability features by hand, since functional usability features are added to the system code by a model compiler.
- Evolutions of usability requirements need to be applied to the conceptual model only. Therefore, system will be able to evolve more easily.

Our proposal to include usability features is valid for any MDD method. However, it has been necessary to select a specific MDD method to fully define our proposal. We have chosen OO-Method [29,28], since it is supported by a commercial tool that is being regularly used to develop real systems by a company (INTEGRANOVA) [6]. Such MDD tool generates fully functional systems from a conceptual model. Another advantage of the MDD method used as benchmark for our research is that its conceptual model is abstract enough to straightforwardly add new primitives that represent usability features.

This paper is the ongoing work of two previous publications: [25,26]. Ref. [25] offers a first draft of the idea to represent functional usability features in a conceptual model. The contribution of this paper with regard to the previous one consists of: (1) A more detailed definition of the procedure to include functional usability features in a conceptual model; (2) A proof of concept with different usability features in a real MDD tool. Ref. [26] is a poster that introduces a short description of an experiment to analyze the benefits of including functional usability features in a system. The contribution of this paper with regard to the previous one consists of: (1) an exhaustive description of the design, threats and results of the experiment to know whether or not users' satisfaction and users' efficiency improves after including functional usability features in the systems; (2) a comparison of effort to include functional usability features in a MDD method manually with the effort to include them through conceptual primitives.

The paper is structured as follows. Section 2 introduces the usability and MDD background necessary to understand our proposal. Section 3 describes our proposal for adding usability features to a MDD method. Section 4 illustrates the application of our proposal to a specific MDD method. Section 5 discusses an experiment to evaluate user satisfaction improvement applying our proposal. Section 6 studies the improvement of the efficiency of analysts working with functional usability features represented as conceptual primitives versus including them manually. Section 7 describes related work. Finally, Section 8 presents some conclusions.

2. Background

The **MDD paradigm** aims to develop software using a conceptual model that abstractly represents the system under development [20]. This conceptual model is the input for a model compiler that generates the code implementing the system. Usually, this generation is performed by transformation rules that are applied automatically. A MDD conceptual model is divided into different views or models. View stands for the set of formal elements that describe something that has been built for a purpose. For example, there can be a view to represent the user interaction, another view to represent system functionality and another view to represent information persistence. Views are composed of conceptual primitives. Conceptual primitives are modelling elements that have the capability of abstractly representing an aspect of the system. Examples of conceptual primitives are class diagram classes, class attributes and services, etc. The system is generated from the conceptual model by a model compiler. The level of automation for code generation is more or less powerful depending on the MDD method.

Usability is a very broad concept. According to ISO 9241-11 [14], usability is "*the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specific context of use*". Human–Computer Interaction (HCI) literature provides many different recommendations to improve software system usability. HCI recommendations can be classified into three groups [16]:

- Usability recommendations with impact on the user interface (UI). They refer to presentation issues which imply slight modifications of the UI design (e.g. buttons, pull-down menus, colours, fonts, layout).
- Usability recommendations with impact on the development process. To follow these advices the development process needs to be tuned. For example, recommendations designed to reduce the user cognitive load state that software development should implicate users.
- Usability recommendations with high impact on architectural design. They involve building certain functionalities into the software in order to improve user-system interaction. This set of usability recommendations are referred to as functional usability features (FUF). Examples of such features are cancel, undo and feedback facilities. Unless these features are considered from the early stages of the software development process, it takes a lot of rework to build them into a software system [4]. We focus our approach on this group of recommendations.

Table 1 shows a summary of FUFs, the mechanisms into which they are divided and their goals. We have selected four mechanisms to illustrate here our approach (shaded in grey in Table 1). This choice is based on the usefulness of the mechanisms for the examples used in this paper.

As shown in [17], a full description and elicitation guidelines for each and every FUF can be found at http://www.grise.upm.es/sites/extras/2/. FUFs were derived from interaction patterns described in the literature as [40,42,31]. FUFs contribute a detailed description of how usability features affect the system architecture, whereas interaction patterns only define how usability features affect the system interface. Another difference between FUFs and interaction