Computers in Industry 65 (2014) 622–635

Contents lists available at ScienceDirect

Computers in Industry

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

Debugging measurement systems using a domain-specific modeling language

Tomaž Kosar [a,*], Marjan Mernik [d], Jeff Gray [c], Tomaž Kos [b]

[a] University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova ulica 17, 2000 Maribor, Slovenia
[b] DEWESoft d.o.o.' Gabrsko 11a, 1420 Trbovlje, Slovenia
[c] University of Alabama, Department of Computer Science, Tuscaloosa, AL 35487, USA
[d] University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova ulica 17, 2000 Maribor, Slovenia

ARTICLE INFO

Article history:
Received 1 March 2013
Received in revised form 14 October 2013
Accepted 21 January 2014
Available online 23 February 2014

Keywords:
Debugging aid
Domain-specific modeling languages
Graphical environments
Usage experience

ABSTRACT

Capturing physical data in the context of measurement systems is a demanding process that often requires many repetitions with different settings. To assist in this activity, a domain-specific modeling language (DSML) called Sequencer has been developed to enable the improved definition of measurement procedures. With Sequencer, the level of abstraction has been raised and sophisticated changes in measurement procedures are now enabled. Although there are numerous DSMLs like Sequencer in the existing literature, there are some obstacles working against the more widespread adoption of DSMLs in practice. One challenge is the lack of supporting tools for DSMLs, which would improve the capabilities of end-users of such languages. For instance, support for debugging a model expressed in a DSML is often neglected. The lack of a debugger at the proper abstraction level limits the domain experts in discovering and locating bugs in a model. In this paper, Sequencer is presented together with debugging facilities (called Ladybird) that are integrated in a modeling environment. Ladybird supports different execution modes (e.g., steps, breakpoints, animations, variable views, and stack traces) that can be helpful during the debugging of a model. Ladybird's primary contribution is in showing the value of error detection in complicated industrial environments, such as data acquisition in automotive testing. The paper contributes to a discussion of the implementation details of DSML debugging facilities and how such a debugger can be reused to support domains other than the measurement context of Sequencer.

1. Introduction

Domain-specific languages (DSLs) [1–3] allow domain experts to play a vital role in the software development process. Empirical evidence has shown that productivity increases with DSL adoption when compared with the traditional code-driven software development process [4,5] that uses general-purpose languages (GPLs), such as Java or C++. The adoption of a DSL raises the level of abstraction [6,7] and connects the concepts from the problem and solution domains [8,9]. Domain experts, who have skills in the problem domain, but may not have formal training in computer science, can write their own domain-specific programs to solve a specific need in their domain [10]. DSLs can be further sub-divided into specification, modeling, and programming languages [11]. In this paper, we focus on domain-specific modeling languages (DSMLs) [12,13], which often use a visual notation rather than textual representation and remain more expressive at a higher abstraction level than GPLs.

In measurement systems, both mechanical equipment and measurement settings have to be tested from various points of view. If traditional software tools based on GPLs are used, this process can be simplified by using the prepared test procedures to speed up the testing process and to analyze the measurement results. However, prepared tests are often insufficient and tests need to be changed, or even developed from scratch. Therefore, it would be a significant contribution to support domain experts with the ability to model measurement procedures on their own. This can be achieved by developing an appropriate DSML, which is very suitable for the construction of measurement systems [14], where physical data are captured and the conversion of these results into a digital form is performed [15]. To improve flexibility and productivity, DEWESoft [16] developed a DSML called Sequencer [17], which enables domain experts to model and evolve their own measurement procedures without any help from programmers. In Sequencer, the measurement procedure can be

* Corresponding author. Tel.: +386 41559205.
E-mail addresses: tomaz.kosar@uni-mb.si (T. Kosar), marjan.mernik@uni-mb.si (M. Mernik), gray@cs.ua.edu (J. Gray), tomaz.kos@dewesoft.si (T. Kos).

0166-3615/$ – see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.compind.2014.01.013

constructed in a textual or visual manner. To the best of our knowledge, specialized measurement systems [18,19] do not allow the construction of measurements to such an extent. Also, existing tools are adjusted specifically to the type of test and are limited in their flexibility and usefulness. These tools (when compared with Sequencer) are limited to one type of hardware vendor, while DEWESoft supports many different types of hardware from various vendors. Moreover, with existing tools it is impossible to hide the unimportant details of the measurements (e.g., Sequencer allows customizations for specific tests, while specialized measurement systems usually support limited flexibility of displayed data during measurements). DEWESoft addresses these shortcomings in Sequencer, where end-users can adjust the measurement and control procedures, while tailoring them to their specific needs by writing an additional sequence (measurement procedure).

However, sequences may become complex such that domain experts face several challenges when trying to detect bugs in the models. Bugs may occur due to specification errors (e.g., semantic errors) or measurement errors (e.g., hardware malfunction). To facilitate sequence construction, the domain expert must be empowered with dedicated tools to improve measurement procedures. Usually, DSML tools other than model compilers (i.e., transformations from models to some other artifact) are most often not available. However, the utility of a DSML is seriously diminished if the supporting tools (e.g., a debugger) needed by a software developer are not available. In this paper, we describe debugging features in a tool called Ladybird, which is integrated in the modeling environment Sequencer [17]. The debugging facilities of Ladybird (e.g., execution modes, steps, breakpoints, animations, print statements, variable view, and stack traces) enable end-users to simultaneously watch multiple models, and during the execution, monitor and alter the state of a running model. Sequencer, as well as features of Ladybird, has been applied to automotive domain,[1] where the quality of the car and its parts are subjected to testing procedures.

The remainder of this paper is organized as follows. Section 2 describes related work on DSMLs, debuggers, and measurement systems. Section 3 highlights the DEWESoft system and gives some details about specific domains where the measurement system has been used. Section 4 explains the architecture of Sequencer and introduces the details of the domain-specific modeling language and the modeling tool. Ladybird, Sequencer's debugger, is discussed in Section 5 and implementation details of debugging support are presented in Section 6. A demonstration on a real case scenario is illustrated in Section 7. Discussion follows in Section 8. Finally, Section 9 provides concluding remarks and summarizes the main features of the Ladybird model debugger for the DEWESoft measurement system.

2. Related work

This section provides an overview of related work in the area of model debuggers and measurement systems.

2.1. An overview of model debuggers

The benefits of using a DSML in software engineering are still not realized fully because the software development lifecycle using DSMLs is often not supported by the appropriate tools. As observed in the case of functional languages, there were several factors that led to the resistance of functional languages in mainstream development: the lack of debuggers and profilers, inadequate support by Integrated Development Environments (IDEs), and poor interoperability [20]. These same factors can all be considered contributing factors for the software industry's resistance to DSMLs. Hence, it is crucial that more work is devoted to DSML tooling [21]. In particular, we are interested in DSML debugging tools. We believe that in the future, DSML debuggers will be generated automatically from metamodels. However, we need to gather enough experience and knowledge from the development of specific DSML debuggers in order to generalize the case to support automatic generation. Our work in this paper is a step toward that direction. Despite the fact that metamodeling tools (e.g., MetaEdit+ [12,22], GME [23], EMF [24]) do not automatically generate DSML debuggers, we will briefly describe GME tool and its capabilities. The situation is better for textual DSLs, where some tools have integrated DSL debuggers (e.g., MPS [25]).

The GME [23] is a metamodeling tool that is similar in purpose to MetaEdit+ [12,22], but differs in its specific use. The metamodel in GME is depicted with a UML class diagram [26] showing elements of the DSML and how they can be associated with each other. Numerous DSMLs have been built using GME (e.g., POSAML [27], PICML, CQML, CUTS [28]). It is important to note that while these tools (e.g., MetaEdit+, GME) are advantageous in constructing DSMLs, they do not support model debugging at a level of abstraction that can be used by most domain experts. Hence, debugging can be performed at the code level only, but not at the model level. At most, one can use the modeling tool API for viewing and manipulating a model's internal representation, which is not sufficient for most end-users.

Developing a DSL debugger from scratch can be very expensive. Therefore, Wu et al. [29] proposed a grammar-driven technique to build a DSL debugger, where the debugger could be generated automatically with minimal additional effort by reusing an existing GPL debugger. However, their approach is applicable only when a DSL is implemented using source-to-source translation of a textual language, where a line of DSL code is consecutively translated into many lines of GPL code. By keeping track of the DSL code to GPL code translation, a GPL debugger can be reused, but debugger actions like "step into" and "step over" have to be reimplemented. Wu et al.'s framework has been extended to generate DSL testing tools automatically, but with similar limitations [30].

In the case of DSML debugging, relevant works are rare. Mannadiar and Vangheluwe proposed a conceptual mapping of debugging concepts from programming languages to DSMLs [31]. Language primitives (e.g., print statements, assertions, and exceptions) and debugger primitives (e.g., execution modes, steps, runtime variable I/O, breakpoints, jumps, and stack traces), which are commonly found debugging facilities in programming languages were mapped into a DSML debugger, forming a starting point for DSML debugger development. As a proof of concept, Mannadiar and Vangheluwe prototyped debugging concepts into a successor of the ATOM^3 tool [32]. They also pointed out two different facets of DSML debugging: the debugging of model transformations, and the debugging of domain-specific models. In our work, special attention will be given to the latter, but initial work in the former is represented by Hibberd et al. [33]. In our work, debugging concepts from [31] have been used in the implementation of the Ladybird debugging tool described in this paper.

Blunk et al. [34] presented an approach for modeling debuggers for a DSML. Their approach requires a metamodel-based description of the abstract syntax of the language (e.g., defined in the Eclipse Modeling Framework – EMF [35]). Debugging is then described with operational semantics at the metamodeling level, where possible runtime states are modeled as part of a DSL metamodel and transitions are defined as a model-to-model transformation. Blunk et al. demonstrated their approach on breakpoints by stepping over lines of a DSL designed for voice

[1] Sequencer has been applied successfully in the automotive industry (e.g., General Motors).