Abstract: Domain-specific modeling language (DSML) allows domain-experts to play a vital role in software development lifecycle, making them programmers/modelers of new systems. Although, there are reports of numerous DSMLs and their advantages, there are some obstacles working against the more widespread adoption of DSMLs in practice. One of them is a lack of supporting tools in most of reported DSMLs, which would assist modelers and make them more efficient. This paper presents DSML called Sequencer, where debugging facilities were integrated in the development environment. Debugging support, such as different execution modes, steps, breakpoints, animations, variable views, stack traces and others have been developed for the Sequencer.

Key-Words: programming language, debugging, breakpoints, data acquisition, measurement systems, vehicle testing

1 Introduction

Software development is becoming more and more demanding and laborious process, since it requires the collaboration of programming engineers as well as users (domain-experts) who understand the problem domain. Usually such development process is time-consuming and expensive. The ideal software development would be if the domain-expert could write its own programs. However, this is not possible with using traditional tools of software development, because users often do not possess knowledge to program in general-purpose programming languages (GPLs). The opposite approach is to develop a domain-specific language (DSL) [1] that increases the level of abstraction and connects the concepts from the problem and solution domain. Domain experts, who have a skill of the problem domain but no formal computer knowledge, can write their own domain-specific programs to solve a specific need in their domain [2]. Moreover, DSML, which use visual notation instead of textual ones, remain more expressive at a higher abstraction level than DSLs [3, 4].

DSML languages are very suitable for the construction of complex measurement systems, where physical data are captured and the conversion of these results into digital form is performed [5]. Some measurements can be very demanding and require a precise measurement procedure. The procedure is usually executed manually or automatically by using previous developed programs. To improve flexibility, productivity and also to emphasize the essential details, DEWESoft developed a DSML called Sequencer [6]. It enables domain-experts to program their own measurement procedures and change them during measurements without any help from programming engineers. In the Sequencer measurement procedure can be constructed in a textual (DSL) or visual manner (DSML). Sequencer is used in various fields from automotive industry, electrical engineering to aerospace industry. In this paper, the debugging facilities of the Sequencer are presented in the automotive domain, where the quality of the car and its parts are subjects under testing procedure. In this domain the Sequencer has already been successfully applied in the industry (e.g. TÜV, a German independent consultant organization that validates the safety of products). For instance, Figure 1 shows the measurement during double lane-change maneuver.

However, sequences are becoming more and more complex and domain-experts have problem detecting bugs in the models. Bugs may occur due to programming errors or measurement (hardware) errors. To facilitate sequence construction, domain-expert must be empowered to test measurement procedures. In this paper, we will show how this can be achieved by using our tool, called Ladybird, through which domain-experts can monitor and alter the state of the running model. The paper is divided as follows. Section 1 briefly presents benefits and motivation for our debugging tool Ladybird. Section 2 includes problem formulation and problem scenarios the domain-experts had in the past with the Sequencer. Section 3 gives explanation about
the debugging tool. Use of debugging support in the Sequencer is presented in Section 4. Finally, Section 5 provides concluding remarks and summarizes the main features of the model debugger for the measurement systems DEWESoft.

A DSML debugger is situated in a similar position as a DSL debugger - instead of checking programs, a debugger is used to debug models. Mannadiar, et al., had similar problems with debugging models and proposed a conceptual mapping between concepts in problem and solution domain [8]. In the Sequencer we solely focus on imperative DSML debuggers. Our debugging tool detects modeling bugs at runtime in the visual manner.

2 Problem Formulation

System complexity increases the number of programming bugs in the process of software development. The tools were developed for that purpose (for instance, debuggers, test engines and profilers) which help to easily and effectively find and resolve programming bugs. In GPLs, debugging is one of the most important programmers’ activities and many believe that this is also true for program development with DSLs [7]. However, developing DSL debugger from scratch can be very expensive. Therefore, Wu et al., proposed a grammar-driven technique to build a DSL debugger, where the debugger could be generated automatically with minimal additional effort, by reusing the existing GPL debugger. However, their approach is applicable only when a DSL is implemented using source-to-source translation, where a line of a DSL code is consecutively translated to 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.

DSL languages can be divided into three categories: imperative, declarative and hybrid [7]. An imperative DSL language is centered on assignment expressions and control flow statements, which allows changing the content of cells in memory at runtime. A declarative DSL language is based on stating the relationship between inputs and outputs. A hybrid DSL language is a mixture of the first two. Each DSL category requires different debugging approaches (e.g., algorithmic debuggers, declarative debuggers, event-based debuggers, assertion checkers, debugging queries).

2.1 Problem scenarios

In the following subsection the limitations reported by the Sequencer’s users are defined. The problems regarding realistic applications and requests have been divided into six different categories.

Problem P1a - During the execution the sequence can fall into an infinitive loop. This problem means that the sequence cannot be stopped. Also, it is impossible to skip the current execution block and proceed with another. Moreover, it is impossible to pause/resume the execution of the sequence.

Problem P1b - When domain-experts run very complex sequences, it is difficult to find the bugs, since the bugs can occur in the model or in the measuring equipment. It would be helpful if we could execute the model step-by-step (i.e. one block at a time) which would bring better control over the execution.

Problem P2 - For better and faster understanding of the sequence, the domain-experts would prefer some kind of animation at the sequence runtime.

Problem P3 - In some cases, domain-experts would like to stop the execution before or after specific a block in the model. In some cases, they would also like to stop the sequence at a specific trigger (for example, when a domain-expert presses the brake pedal in a car).

Problem P4 - Some applications are time critical. One example of such an application is testing the water heater, wherein the measurement lasts for a long time.

Problem P5 - For faster testing, domain-experts often want to have the ability to monitor and alter values of a running sequence.

3 Problem Solution

To support domain-experts’ development in the Sequencer, we have provided the debugging tool Ladybird in order to fasten their development of measurement systems. Debugging is supported at the model level and not at the level of synthesized DSML code. It offers features such as model-level breakpoint, step controls and print statements.

Figure 2 shows the architecture and functionalities of the debugger. In general, there are three levels. The DEWESoft software is on the first level. The second level is the Sequencer, which enables the construct of the measurement procedures and it controls DEWESoft and supervises the hardware. On the third level there are
debugging facilities: seven different features, such as execution modes, steps, model animations, breakpoints, print statements, variable views, and stack traces. These seven parts of the Ladybird tool are further described in the sequel.

3.1 Execution Modes and Steps
The sequence can be run in release or debug mode. In release mode the domain-expert can only play the sequence. To be able to use different debugging features, the sequence must be run in debug mode. In this mode one has the ability to run the model continuously or step-by-step. In order to solve the problem P1a, we added the controls “Stop”, “Pause” and “Skip”. The first control allows the domain-expert to terminate the sequence. The second one halts the execution of the sequence. When the sequence execution is halted, the domain-expert can continue its execution with the “Play” control. The control “Skip” allows for the skipping of the current execution block. In addition, with this control we can urge execution, so that some block waiting for a certain action is simply skipped. For example, testing water heaters takes the whole day. Testing in such problem domains is a very wasteful process. With this control some blocks that do not affect the testing procedure, can be simply skipped.

Another feature contains three types of steps: “Step Over”, “Step Into” and “Step Out”, which help to solve the problem P1b. The first control enables the execution of a block at once, skipping the sub-model (in the Sequencer one can define a model and this model can be used elsewhere as a “custom block”). The second control allows changing of the current scope and enables stepping into a sub-model. If you are using “Step Into” and want to exit from the current context, one can use the “Step-Out” control. This control tells the debugger to run until the end of current model (scope) and return one-level higher. Stepping is enabled only in debug mode.

3.2 Animation
Sometimes non-experienced users do not envision the meaning of returned values by the Sequencer. Although they are domain-experts and have domain knowledge, they can not understand the returned values. Therefore, a visual presentation or animation of the current model state can be an useful aid in order to make one’s perception easier and clearer.

Even in our DSML, similar problems occurred, so that a new feature in the form of an animation was developed and thus solved the problem P2. Another purpose of this feature was to help the domain-expert inspect the flow of given model and to understand its behavior. Visualization consists of a static and dynamic view. The first is a model presentation like in the modeling tool (see Figure 3) whereas the dynamic view is an animation at runtime.

3.3 Breakpoints
Very complex and complicated sequences are almost impossible to follow using step-by-step features (the problem P3). Therefore, it is necessary to use other mechanisms. From the GPL languages, these are known as “breakpoints”, which temporarily suspend the execution of a program at a certain statement in programming code. In the Sequencer, the same technique was used where the breakpoint can be applied to blocks in the model. Domain-experts have the ability to specify when or where the breakpoint should interrupt a normal sequence execution.
The suspension can be considered and determined in two ways: the source or data breakpoints. The source breakpoint is attached to the block contained by the model. The feature also supports associating "pass count", which enables domain-experts to specify an arbitrary number that determines how many times the breakpoint can be executed before the debugger stops the execution. Moreover, this breakpoint feature also supports the condition that is an expression that determines whether the execution should be halted. This is useful if we want to halt the execution only if the signal from the measurement returns a certain value.

The second type is data breakpoints. Here, the execution stops when a value of local or global variable is changed. For example, in the brake test, the sequence would be stopped by pressing the brake pedal in the car. In that case, the analog signal from the brake sensor jumps from zero to five volts.

3.4 Print Statements
In some domains, the duration of the execution is very long (e.g. water heater). In such cases, debugging in the form of steps and breakpoints is difficult (problem P4). It is thus necessary to find other solutions that have no direct impact on the execution itself. To this end, we use a known GPL technique: “print statements”. They are generally used to output local and global variables and to verify that blocks are executed at certain points of execution. That information can be observed in the output window and can be stored directly to the text file. After a long test, the stored values can be studied by domain-experts.

3.5 Variable View and Stack Traces
In order to solve the problem P5, in the Sequencer there is an option of changing values of local and global data variables (also called channels) at the runtime. We can monitor various measured signals, such as analog, digital, counter, and the signals that come from different data buses (e.g., CAN, FlexRay, ARIC 429 and MIL-STD-1553).

In addition, this feature allows one to see the model level of the currently executed block. The domain-expert can see which models have led the execution into its current state. The feature is called “stack traces” and becomes visible when the execution is suspended.

4 An Example
In Figure 4 debugging mode in the Sequencer is shown. For the purpose of demonstration a double-lane test change is used. It is used to identify the vehicle dynamic.

The vehicle should be driven from the initial lane to another lane in parallel, and back to the initial line, without exceeding lane boundaries. The result is to analyze and determine the road holding ability and handling the vehicle characteristics. In this test it is very important to observe and record all measurement signals (e.g. velocity) at a high sampling rate and all should be synchronized together.

The constructed sequence of maneuver is defined with a few steps. The first step is to load project and setup files. After that, the sensors parameters must be set. The sequence continues with the entering car’s length and width which are used to calculate the precision of driving. The maneuver can be run only by test drivers that have a lot of experience. During the measurements, the test driver can observe on the monitor the following parameters: enter, leave and average velocity, vehicle roll, acceleration, and absolute distance between cones and car as shown in Figure 1. The double-lane change is started when a certain condition (e.g. velocity should be 80 km/h or higher) is reached. When the measurement is finished, the average speed between enter and leave velocity should be calculated. If the test turns out to give unexpected or inadequate results, it can be repeated, so that the system returns to a previous point. At the end of the test procedure, the overall measured values can be analyzed.

When all the hardware and sequences are prepared, we want to start with debugging when the car starts moving and then step-by-step execute the testing procedure. In addition, we can observe the values of speed and the initial condition to start with a normal execution procedure (not stepping). Since the application is time critical we want current blocks, and the values of variables to be stored in the text file.

In our case we propose the following solution for the detection of potential bugs. For the detection and termination of sequences we can use the data breakpoint where we can set the condition that the sequence would stop when the value of the variable “velocity” is higher than 10 km/h. After that we execute step-by-step sequence and monitor variables until the velocity is higher than the enter speed. Then we add the print block and set it to store in the file name of current executed block and the value of the current variable. Later, we can observe in this file what could be a potential error of the measurement. Figure 4 shows the example described above. The first part of the screenshot marks the steps controls. The second part in Figure 4 shows the animation where the domain-expert can observe progressing of model execution. Watch list, output view, call stack and data breakpoints are shown on the third part. On the right side of the screen the domain-expert can observe the execution and working of the measurement system (part four).
5 Conclusion

This paper presents the debugging features in Sequencer, a DSML integrated in measurement systems DEWESoft. The modeling environment offers control features, such as model step-wise execution, break point functionality and model behavior animation. With the ladybird tool, it is also possible to simultaneously watch multiple models, and during their operation, monitor and alter local and global variables. Now the domain-experts can create and trace measurement procedure faster and more efficiently.

Acknowledgements

This work was partly sponsored by the European Social Foundation. Moreover, we would like to thank DEWESoft co-workers who contributed to a discussion of DSML problems.

References: