André van Hoorn, Holger Knoche, Wolfgang Goerigk, and Wilhelm Hasselbring. Model-Driven Instrumentation for Dynamic Analysis of Legacy Software Systems. In Proceedings of the 13. Workshop Software-Reengineering (WSR '11), pages 26–27. May 2011.

Model-Driven Instrumentation for Dynamic Analysis of Legacy Software Systems*

André van Hoorn[1], Holger Knoche[2], Wolfgang Goerigk[2], and Wilhelm Hasselbring[1]

[1] Software Engineering Group, University of Kiel, 24098 Kiel

[2] b+m Informatik AG, 24109 Melsdorf

Abstract

Dynamic analysis requires the instrumentation of application code with monitoring probes. This paper presents an approach to generate instrumentation artifacts from models augmented with analysis directives. Special emphasis is put on how to add monitoring instrumentation by means of aspect-oriented programming (AOP) to programs written in legacy languages.

1 Introduction

Dynamic analysis is a valuable tool for re-engineering, as it provides information about the runtime behavior of software systems [1] vital to tasks such as architecure reconstruction. However, it requires the instrumentation of the application code with monitoring probes. Moreover, the collected monitoring data needs to be parsed, transformed, and aggregated with respect to the desired architecture-level analysis goals.

This paper describes our approach for automating these steps based on analysis directives specified in architectural or other domain-specific models. We employ aspect-oriented programming [2] to add monitoring probes to programs written in current or legacy languages. The approach uses our Kieker framework for continuous monitoring and analysis of software systems [5]. The examples in this paper focus on the Visual Basic 6 (VB6) programming language, as this language is used in one of the case studies of our DynaMod project for model-driven modernization of software systems [4].

The remainder of this paper is structured as follows: Sect. 2 describes our approach for model-driven instrumentation of software systems. Sect. 3 focuses on the AOP framework for legacy languages which is used to add monitoring instrumentation (Sect. 4). Conclusions are drawn in Sect. 5.

2 Model-Driven Instrumentation

Software systems can be represented on different layers of abstraction from code-centric representations like source code and abstract syntax trees (ASTs) over architectural models employing architecture description languages (ADLs) to models written in domain-specific languages (DSLs) close to the problem domain. Model-driven software development (MDSD) [3] provides techniques and tools to generate implementation artifacts from higher abstraction models employing model-to-model and model-to-code transformations. In order to capture legacy code, parsers and static analysis techniques extract models from source code.

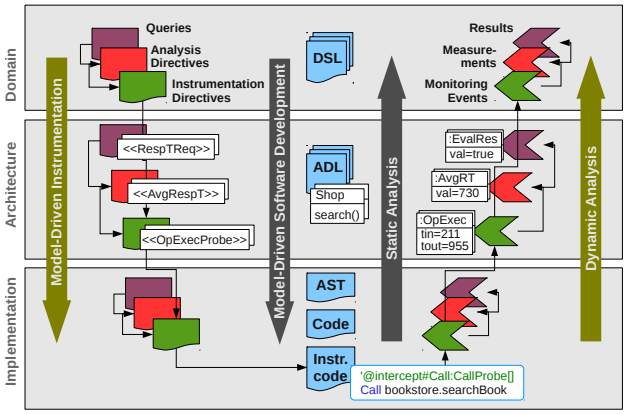

Figure 1: Overview of the approach

To automate the instrumentation of code for dynamic analysis, we propose the approach depicted in Fig. 1. An abstract model extracted by static analysis is augmented with *queries*, for instance, whether the response time of a specific method does not exceed 500 ms in 90 % of all cases. These queries are transformed into *analysis directives* (in this case, to calculate the 0.9-percentile of the operation's response time), which are in turn concretized by *instrumentation directives* specifying the means necessary to gather the data required by the analysis directives.

This approach allows not only to generate the required instrumentation from augmented models using well-understood MDSD techniques. It also enables the generation of the necessary infrastructure to automatically aggregate the *monitoring events* observed during the program execution into the *results* of the respective queries.

3 AOP for Legacy Languages

One shortcoming of manual instrumentation is that monitoring aspects are mixed with application functionality, having a negative impact on source code understandability and maintainability. This problem

*This work is supported by the German Federal Ministry of Education and Research (BMBF) under grant number 01IS10051.

can by solved by the use of aspect-oriented programming (AOP), introduced by Kiczales et al. [2]: application functionality and so-called cross-cutting concerns, such as instrumentation, are kept in separate program artifacts and combined automatically to form the executable application code.

For modern programming platforms such as Java or .NET, several AOP implementations exist which are already widely used. Many of the monitoring probes in our Kieker framework, for instance, benefit from AOP capabilities. As we are interested in monitoring legacy systems, we decided to develop a generic approach that introduces AOP capabilities to legacy languages, which can then be used to weave monitoring instrumentation into the source code.

Fig. 2 illustrates how aspects are applied to a piece of VB6 code. In this case, the source code is augmented with aspect-oriented directives which are included as comments and have the form `@intercept#<Advice>:<Aspect>[<data>]`. The example includes two advices: `Call` to execute aspects before and after specific procedure calls, and `Execution` for doing this for every execution of a procedure. Depending on the type of advice, aspects must implement special interfaces—in this case, they must provide `before` and `after` procedures which are called before and after the respective call or execution. When the interface methods are called, data corresponding to the advice and activated join point (in this case, `IcptOpCall` and `IcptOpExec`) is passed to the aspect. This allows, for example, to integrate monitoring instrumentation for monitoring executions, as detailed in the following Sect. 4.

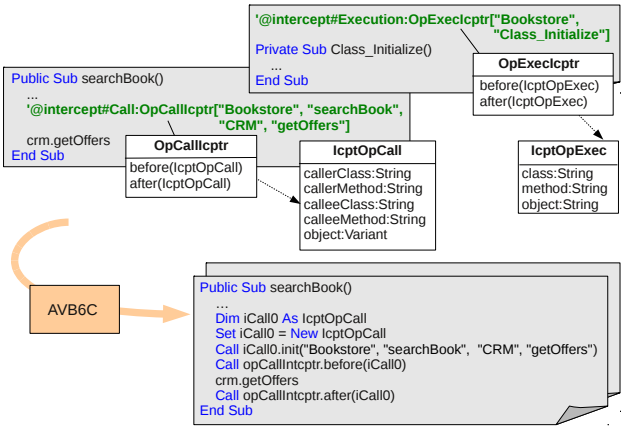

Figure 2: AOP weaving example for VB6

4 Monitoring Instrumentation

To avoid a complete reimplementation of the monitoring and logging functionality for each targeted legacy language, we use the Java-based Kieker monitoring component whenever possible. This requires the implementation of language-specific probes and a way to pass the monitoring data to the Java monitoring component. In order to add monitoring support to native Windows applications, for instance, we developed a DLL providing access to the existing Kieker monitoring core.

```
Public Sub before(icptOpExec As IcptOpExec)
   Dim tin As Variant
   tin = monitoringCtrl.currentTimeNanos
   icptOpExec.object = tin
End Sub

Public Sub after(icptOpExec As IcptOpExec)
   Dim r As OperationExecutionRecord
   Dim tin, tout As Variant
   Set r = New OperationExecutionRecord
   tin = icptOpExec.object
   tout = monitoringCtrl.currentTimeNanos
   Call r.initFields(icptOpExec.class, icptOpExec.class, icptOpExec.class, tin, tout)
   Call monitoringCtrl.writeRecord(r)
End Sub
```

Figure 3: AOP-based Kieker monitoring probe for VB6

As an alternative to manual instrumentation, we use the AOP framework described in the previous Sect. 3 to weave Kieker monitoring probes into source code of legacy systems. Fig. 3 shows a VB6 monitoring probe that monitors executions of procedures employing the afore-mentioned Kieker DLL.

5 Conclusions

As a proof of concept, we have implemented crucial parts of the described approach. In our future work, we plan to extend its functionality in terms of programming languages (e.g., Cobol, Natural, and Structured Text (ST) for SPS-based embedded systems); abstraction layers and models for instrumentation and analysis, particularly based on ASTs and DSLs; metrics, statistical functions, monitoring probes, as well as corresponding AOP features (e.g., for supporting loops and branches). Moreover, we will systematically study the monitoring overhead, as performed for the Java platform in our earlier work [5].

References

[1] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Transactions on Software Engineering*, 35(5):684–702, 2009.

[2] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proc. Europ. Conf. Object-Oriented Programming (ECOOP '97)*, volume 1241 of *LNCS*, pages 220–242. Springer, 1997.

[3] T. Stahl and M. Völter. *Model-Driven Software Development – Technology, Engineering, Management.* Wiley & Sons, 2006.

[4] A. van Hoorn, S. Frey, W. Goerigk, W. Hasselbring, H. Knoche, S. Köster, H. Krause, M. Porembski, T. Stahl, M. Steinkamp, and N. Wittmüss. DynaMod project: Dynamic analysis for model-driven software modernization. In *Proc. 1st Int'l Workshop on Model-Driven Software Migration (MDSM '11)*, volume 708 of *CEUR Workshop Proceedings*, pages 12–13, 2011.

[5] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. Continuous monitoring of software services: Design and application of the Kieker framework. TR-0921, Dept. Comp. Sc., Univ. Kiel, Germany, Nov. 2009.