

# Instrumentierung zum Monitoring mittels Aspekt-orientierter Programmierung

Thilo Focke<sup>1</sup>, Wilhelm Hasselbring<sup>2</sup>  
Matthias Rohr <sup>\*</sup>2 und Johannes-Gerhard Schute<sup>1</sup>

<sup>1</sup> EWE TEL GmbH, Cloppenburger Str. 310, 26133 Oldenburg

<sup>2</sup> Graduiertenkolleg TrustSoft, Abteilung Software Engineering, Universität Oldenburg

**Zusammenfassung** Das Monitoring großer, kontinuierlich laufender Softwaresysteme liefert wichtige Daten zu deren Überwachung und Fehlerdiagnose. Wenn die Wartbarkeit der zum Monitoring nötigen Instrumentierung und der Softwareapplikation selbst kritisch ist, verbietet sich ein manuelles Einbringen des Messcodes. Aspekt-orientierte Programmierung (AOP) ermöglicht die isolierte Programmierung von Querschnittsbelangen und das automatisierte Integrieren in den Applikationscode per Annotationen. Unser Beitrag berichtet über Erfahrungen mit AOP zur Instrumentierung für Performance-Monitoring in einem verteilten Kundenportalsystem eines Telekommunikationsanbieters. Insbesondere bedarf der durch AOP erhoffte Wartbarkeitsvorteil einer kritischen Untersuchung.

**Einleitung** Der Betrieb von softwareintensiven, geschäftskritischen Softwaresystemen benötigt Monitoring, um die Überwachung und Analyse des Laufzeitverhaltens zu ermöglichen. Hierfür muss zusätzliche Funktionalität in das Softwaresystem integriert werden, um Messdaten zu erfassen und an Datenbanken, Protokolldateien oder Überwachungssysteme zu übermitteln. Selbst nach systematischem Entwurf eines Laufzeitmonitorings (wo, zu welchem Zweck, wie und was soll erfasst werden, vgl. [1]) stellt sich die Frage, wie der bestehende Applikationscode erweitert werden kann, ohne dass zukünftige Änderungen an der Applikation oder am Monitoring erschwert werden. Ein hierfür viel versprechender Ansatz ist durch Aspekt-orientierte Programmierung (AOP) gegeben, welche Querschnittsbelange wie Monitoring in so genannte Aspekte kapselt und zur Lauf- oder Kompilierzeit in markierte Methoden einwebt.

**Monitoring von Laufzeitverhalten** Unter Monitoring versteht man das Erfassen und Aufzeichnen (teilweise auch Analysieren und Überwachen) des Laufzeitverhaltens von Komponenten, wie beispielsweise Softwarekomponenten, Diensten oder Betriebssystemprozessen (vgl. [2]). Das Monitoring eines Softwaresystems kann auf verschiedensten Ebenen durchgeführt werden.

Für die unteren Systemebenen existieren recht ausgereifte Monitoring-Mechanismen (z.B. Plattform-, Netzwerk- und Container-Monitoring). Hingegen sind Lösungen eher selten, die gezielt selbstentwickelte Anwendungen betrachten. Somit erfordert oft auch das Monitoring auf dieser Applikationsebene eigene Instrumentierungsansätze. In geschäftskritischen Anwendungen ist es wichtig, Systemzustände nachvollziehen zu kön-

---

\* Diese Arbeit wurde unterstützt von der Deutschen Forschungsgesellschaft (DFG), GRK 1076/1

nen. Die protokollierten Informationen sind im Falle eines Systemabsturzes für Administratoren und auch Entwickler oft die einzige Hilfe, um den Ablauf eines Programms reproduzieren zu können.

**Instrumentierung durch Aspekt-orientierte Programmierung** Mittels Code-Instrumentierung kann bestehender Programmcode in einem zusätzlichen Verarbeitungsschritt um Code erweitert oder manipuliert werden. Der naivste Ansatz des Einbringens von Monitoring-Code ist das manuelle Instrumentieren von Performance-kritischem Programmcode durch das Einfügen von entsprechendem Pre- und Postcode. Dies führt allerdings zu einem in der Applikation verstreuten, redundanten Programmcode. Da ein System üblicherweise häufigen Änderungen unterliegt, wird die Wartbarkeit des Systems beeinträchtigt und die Fehleranfälligkeit wird erhöht. Es werden daher Techniken benötigt, die die zu überwachenden Applikationsteile automatisch instrumentieren, ohne die Wartbarkeit des Systems zu beeinträchtigen. Zwar bieten Objekt-orientierte Programmiersprachen eine Modularisierung des Programmcodes durch die Kapselung zusammengehöriger Funktionalitäten in unabhängige Klassen mit klar definierten Schnittstellen. Allerdings lassen sich so genannte Crosscutting-Concerns (dt. Querschnittsbelange), wie Monitoring, nur schlecht in eigenen Klassen kapseln, da sie sich oft direkt auf andere Methodenaufrufe beziehen und an vielen Stellen eingebunden werden müssen.

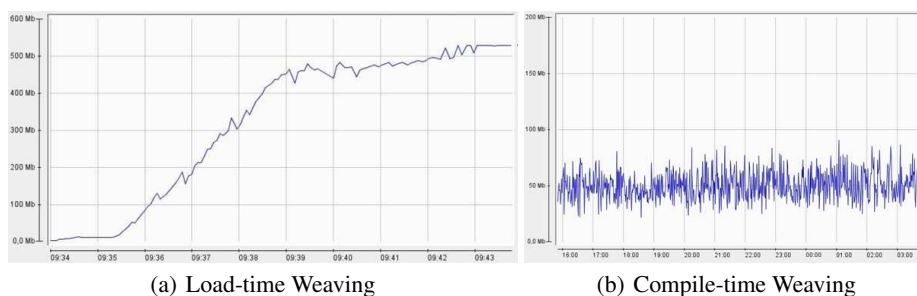
Eine Lösung für diese Problem bietet das Paradigma der Aspekt-orientierten Programmierung [3], indem es die Querschnittsbelange von den Kernfunktionalitäten trennt. Die implementierten Querschnittsbelange werden "Aspekte" genannt und separat von anderen Teilen des Systems entwickelt. Während der Kompilierung oder zur Laufzeit, verwebt der "Aspektweber" (engl. aspect weaver) Aspekte mit dem Programmcode. Eine verbreitetes Werkzeug für AOP in Java stellt AspectJ<sup>3</sup> dar. Seit Java Version 5 können so genannte Annotationen genutzt werden, um Methoden oder Klassen mit zur Laufzeit auslesbaren Meta-Informationen zu erweitern. Durch diese Annotationen kann AspectJ mitgeteilt werden, um welche Aspekte eine Methode oder eine Klasse ergänzt werden soll.

**Einsatzszenario** Bei dem betrachteten Softwaresystem handelt es sich um ein Java-basiertes Kundenportal für die rund 277.000 Kunden (Stand: 31.12.2005) des im nord-deutschen Raum tätigen Telekommunikationsunternehmens EWE TEL. Als eines der größten regionalen TK-Unternehmen Deutschlands bietet EWE TEL eine hohe Vielfalt an Sprach-, Internet- und Datendiensten, die über dieses Portal durch den Kunden angepasst werden können. Konkret handelt es sich um eine mehrschichtige Java-Anwendung, die auf Containern wie dem Servlet Container Apache Tomcat 5.5.12 oder dem Applikationsserver Bea WebLogic 9.1, verteilt ist. Hauptsächlich sollen Auslastungsdaten zur langfristigen Ressourcenplanung und für eine weitere Verbesserung der Systemüberwachung (Fehlererkennung, und Fehlerdiagnose/Engpassanalyse) kontinuierlich gemessen werden. Für das Instrumentierungsverfahren ist die Erhaltung der Wartbarkeit des Softwaresystems die wesentliche Anforderung, da oft neue Anforderungen durch das Kundenportalsystem umzusetzen sind. Des Weiteren soll das nachträgliche Einbringen oder Anpassen von Messpunkten einfach sein, damit sich auf der ganzen Systemlandschaft ein effizientes und einheitliches Monitoring pflegen läßt.

<sup>3</sup> <http://www.eclipse.org/aspectj/>

**Laufzeit- vs. Kompilierzeit-Weben** Die vom Monitoring zu überwachenden Methoden wurden per Annotationen markiert. In der Fallstudie wurden zwei Szenarien des Webens der Aspekte betrachtet. Zunächst wurde das Laufzeitweben (engl. Load-time Weaving, LTW) eingesetzt, bei dem erst zur Laufzeit die geladenen Klassen auf relevante Markierungen überprüft werden. Der Vorteil dieser Strategie ist, dass zur Kompilierzeit kein zusätzlicher Verarbeitungsschritt (durch den Postcompiler von AspectJ) anfällt und somit keine großen Änderungen an den Build-Skripten erforderlich sind. Im zweiten Szenario wurde dennoch das Weben zur Kompilierzeit eingesetzt (engl. Compile-Time Weaving, CTW) da hartnäckige Speicherverbrauchsprobleme mit dem Laufzeitkompi-ler von AspectJ 1.5.0 auftraten (siehe unten).

**Gesammelte Erfahrungen** Das Integrieren von den Messpunkten über Java 5 Annotationen gestaltete sich als sehr einfach und wurde von den Softwareentwicklern des Kundenportalsystems als positives Akzeptanzkriterium herausgestellt. Insbesondere konnten alle am System beteiligten Softwareentwickler ohne vorherigen Lernaufwand Messpunkte einsetzen und entfernen, da keine Kenntnisse in Aspekt-orientierter Programmierung hierfür benötigt werden. Im Vergleich dazu ist das Identifizieren von relevanten Positionen für Messpunkte (wo soll was gemessen werden) wesentlich schwieriger, aber dieses Problem existiert unabhängig von dem in diesem Artikel betrachteten Instrumentierungsansatz (siehe beispielsweise [1]).



**Abbildung 1.** Heap Memory Usage

Das Einweben des Monitorings mittels LTW verlief in verschiedenen Prototypen problemlos, führte allerdings im Testbetrieb mit dem gesamten Portalsystem nach einiger Zeit zu Speicherüberläufen (siehe Abbildung 1(a)). Zudem stiegen die Startup-Zeiten sowohl des Apache Tomcat 5.5 als auch des Bea Weblogic 9.1 um etwa 30-40 %, und der während des Startup-Prozesses in Anspruch genommene Speicher stieg ebenfalls um 30-40 % an. Das Problem konnte zusammen mit Mitgliedern des AspectJ-Entwicklerteams in Verbindung mit RMI-Kommunikation und LTW in AspectJ 1.5.0 gebracht werden. Die Java Virtual Machine registriert für jeden RMI-Aufruf einen eigenen Classloader. Dieser muss Klassen nachladen, die für den verteilten Aufruf auf der Client- und Server-Seite nötig sind. Dies ist normalerweise ein leichtgewichtiger Prozess, der die Java Virtual Machine nicht weiter belastet. Im Evaluationsszenario wurde bei jedem RMI-Aufruf ein neuer Load-time weaving-Vorgang durch AspectJ gestartet.

Zwar war dieses Verhalten den AspectJ-Entwicklern bekannt, allerdings war das Ausmaß der Konsequenzen anscheinend weniger bewusst, da dieses Problem nur in hinreichend komplexen Anwendungen auftritt, die beispielsweise Konzepte wie Enterprise-Java-Beans (EJB) einsetzen. Da zu diesem Zeitpunkt keine Lösung für dieses konzeptionelle Problem von AspectJ 1.5.0 gefunden werden konnte, musste auf Compile-Time Weaving umgestellt werden, und damit eine Erweiterung der Applikationsbuild-Skripte erfolgen.

Bei der Verwendung von Compile-time weaving trat durch das Performance-Monitoring an etwa 25 Messpunkten kein nennenswerter Performance-Overhead auf und der Heap-Speicherverbrauch lag bei 50 MB (siehe 1(b)), statt bei 30 MB ohne AspectJ. Allerdings mussten für CTW die Build-Skripte aller zu überwachenden Teil-Applikationen um AspectJ-Kompileraufrufe erweitert werden. Dadurch werden zusätzliche AspectJ-Kenntnisse von den jeweiligen Entwicklern benötigt und eine Rekompilierung und ein Redeployment aller Applikationen ist bei Änderungen der Aspekte erforderlich.

**Zusammenfassung** Die Funktionalität des Performance-Monitoring konnte im Sinne der Aspekt-orientierten Programmierung erfolgreich als eigener Aspekt gekapselt werden. Das Nutzen der Aspekte per Java-Annotationen im bestehenden Programmcode wurde von am System beteiligten Entwicklern als sehr einfach und wartungsfreundlich beschrieben. Insbesondere wurden keine Kenntnisse über Aspekt-orientierte Programmierung hierfür benötigt, sobald einmal beim LTW die Middleware angepasst war, bzw. beim CTW die Applikations-Build-Skripte. Demnach erfüllt die verwendete Technologie für Aspekt-orientierte Programmierung im Kontext von Monitoring hohe Wartbarkeitsanforderungen und konnte sich im Praxiseinsatz für die Instrumentierung mit Performance-Monitoring grundsätzlich bewähren. Allerdings war das Load-time weaving in der vorliegenden Version von AspectJ noch nicht technisch ausgereift und es mußte auf Compile-time weaving ausgewichen werden. Diese Lösung wird seit Mai 2006 im operativen Betrieb eingesetzt.

Es muss darauf hingewiesen werden, dass Aspekt-orientierte Programmierung auch einige Gefahren bietet. Obwohl positive Erfahrungen im Hinblick auf Monitoring gesammelt werden konnten, stellen nur wenige Funktionalitäten so klare Querschnittsbelange dar. Das Kapseln von Funktionalität in Aspekte, die zudem noch, wie bei AspectJ, spezielle Syntax verwenden, kann auch schnell die Verständlichkeit des Anwendungscodes verschlechtern. Weitere mögliche Nachteile entstehen durch die erforderlichen Erweiterungen an den Middleware-Start-Skripten (beim LTW) oder Build-Skripten (beim CTW), wie beispielsweise ein zusätzliches Potential für Konfigurationsfehler und Wartbarkeitsprobleme.

## Literatur

1. Focke, T., Hasselbring, W., Rohr, M., Schute, J.G.: Ein Vorgehensmodell für Performance-Monitoring von Informationssystemlandschaften. In: Tagungsband Enterprise Application Integration EAI'06. (2006)
2. IEEE Standards Board: IEEE standard glossary of software engineering terminology—IEEE std 610.12-1990 (2002)
3. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In Akşit, M., Matsuoka, S., eds.: Tagungsband ECOOP'97. Springer-Verlag, Berlin, Heidelberg, and New York (1997) 220–242