

Ein Vorgehensmodell für Performance-Monitoring von Informationssystemlandschaften

Thilo Focke¹, Wilhelm Hasselbring²
Matthias Rohr ^{*2} und Johannes-Gerhard Schute¹

¹ EWE TEL GmbH, Cloppener Str. 310, 26133 Oldenburg

² Graduiertenkolleg TrustSoft, Abteilung Software Engineering
Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg

Zusammenfassung Der Betrieb von softwareintensiven, geschäftskritischen Informationssystemlandschaften benötigt ein Performance-Monitoring um die Überwachung und Analyse von Laufzeitverhaltens zu ermöglichen. Während die rein technische Implementierung von Performance-Monitoring eher unproblematisch ist, bietet sich bisher kein Vorgehensmodell für den systematischen, zielgerichteten Einsatz in komplexen Systemen an. Somit haben die in der Praxis anzutreffenden „ad-hoc“-Realisierungen oftmals eine mangelhafte Effektivität und Wartbarkeit zur Folge. Aus diesem Grund wird in diesem Artikel ein Monitoring-Ansatz vorgestellt, dessen Kern aus einem Vorgehensmodell für die Planung und Integration eines Performance-Monitoring besteht. Damit zusammenhängend wurde eine wiederverwendbare Infrastruktur entwickelt, die die Monitoringdatenintegration für verteilte Systeme leistet. Der vorgestellte Ansatz wurde in einem Telekommunikationsunternehmen evaluiert und die dementsprechend umgesetzte Monitoring-Lösung befindet sich seit Mai 2006 im operativen Betrieb.

1 Einleitung

Enterprise Application Integration zielt insbesondere auf die Unterstützung der Geschäftsprozesse, die in Unternehmen oder zwischen Unternehmen üblicherweise mehrere Informationssysteme betreffen. Dabei ist es nicht nur wichtig, Konzepte für den *Aufbau* derartiger Landschaften von Informationssystemen zu entwickeln. Es ist auch äußerst relevant, Konzepte für den verlässlichen *Betrieb* komplexer IT-Landschaften zu finden. Für unternehmenskritische Systeme müssen Ausfallzeiten möglichst vermieden oder klein gehalten werden.

Ein Ansatz hierfür ist die Überwachung und Langzeitanalyse von Laufzeitverhalten. Hierdurch können automatisch Fehler oder Performance-Probleme entdeckt werden und schnell Hinweise auf dessen Ursachen ermittelt werden. Das Monitoring der Performance (insbesondere Laufzeiteffizienz) liefert somit wertvolle Daten, um unternehmenskritische Systeme hoch verfügbar halten zu können.

Unter Monitoring versteht man das Erfassen und Aufzeichnen (teilweise auch Analysieren und Überwachen) des Laufzeitverhaltens von Komponenten, wie beispielsweise Softwarekomponenten, Dienste oder Betriebssystemprozesse (vgl. [1]). Während

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

zahlreiche Standards (SNMP, WBEM/-CIM) und proprietäre Produkte (HP OpenView, MS Operations Manager, IBM Tivoli) für die Überwachung und das Management von Systemlandschaften, Netzwerken und Hardware existieren, sind Lösungen und insbesondere Vorgehensmodelle für das Monitoring auf *Softwareapplikationsebene* Mangelware. Da jedoch Informationssystemlandschaften immer softwareintensiver werden, ist es nicht mehr zeitgemäß, Softwareapplikationen als monolithischen Block zu beobachten, sondern eine innere Betrachtung nötig, um präzisere Diagnosen zu erhalten. Auch in der Lehre und Ausbildung bleiben die Aspekte des Softwarebetriebs und im speziellen Monitoring und dessen zielgerichteter Einsatz bislang weitestgehend unbeachtet.

Da in der Praxis oft das Bewusstsein existiert, dass ein Monitoring auf Softwareapplikationsebene benötigt wird, aber da kein ganzheitlicher Ansatz zur Verfügung steht, lässt sich vielerorts ein selbst entwickeltes, wenig dokumentiertes, „ad-hoc“ Monitoring vorfinden. Das ad-hoc Monitoring stellt den naivsten Monitoring-Ansatz dar. Hierbei wird „aus dem Stegreif“, per Hand der Applikationscode um Standardausgabebefehle oder Aufrufe von Logging-Frameworks erweitert. Es ist nicht selten, dass weder eine Trennung des Logging-Codes von der restlichen Anwendungslogik vorgenommen wird, noch systematisch geplant und dokumentiert wird, wo, zu welchem Zweck, wie und was überhaupt mit dem Monitoring erfasst werden soll. Somit ist die Effektivität und Wartbarkeit, wie bei auch jeder anderen unsystematischen Softwareentwicklung, üblicherweise mangelhaft.

Um die Probleme des „ad-hoc“ Monitorings zu vermeiden ist ein ganzheitlicher Monitoring-Ansatz nötig. Der in diesem Artikel beschriebene *Monitoring-Ansatz* zum Performance-Monitoring besteht aus:

- einem *Monitoring-Vorgehensmodell* als Anleitung für die phasenweise, zielgerichtete und Planung, Implementierung und Integration eines Performance-Monitoring und
- einer *Monitoring-Infrastruktur* als vorimplementiertes Framework aus Standardkomponenten des Monitorings für verteilte Java EE-basierte Systeme.

Der Beitrag des Monitoring-Vorgehensmodells ist, dass ein zielgerichtetes Vorgehen angeboten und als Folge das Risiko einer Eigenentwicklung vermindern wird. Weiterhin wird eine spätere Rekonstruktion des Entwicklungsprozesses eines Monitorings und der zu Grunde liegenden Entscheidungen ermöglicht. Zuletzt soll es auf Grund von erhöhter Systematik einheitlichere Ergebnisse erzielen lassen, welches speziell bei großen Systemen wichtig ist, bei denen mehrere Entwickler an der Realisierung eines Monitorings beteiligt sind. Die Monitoring-Infrastruktur kapselt allgemeine Elemente des Monitorings in wiederverwendbaren Komponenten, um den Entwicklungsaufwand zu reduzieren.

Der Artikel ist wie folgt aufgebaut: Zunächst wird der aus dem Vorgehensmodell und der Infrastruktur bestehende Monitoring-Ansatz (Abschnitt 2 und 3) vorgestellt, wobei der Schwerpunkt auf den Phasen des Vorgehensmodells liegt. In Abschnitt 4 wird die Anwendung in einem realen System eines Telekommunikationsunternehmens beschrieben. Zum Ende dieses Artikels (Abschnitt 5) erfolgt eine Zusammenfassung sowie ein Ausblick über zukünftige Erweiterungen.

2 Das Vorgehensmodell

Vorgehensmodelle organisieren Entwicklungsprozesse, indem empfohlene Methoden und Techniken in Phasen gegliedert werden. Ein spezielles Vorgehensmodell für Performance-Monitoring kann im Gegensatz zu allgemeinen Softwareentwicklungsmodellen (z.B. Spiralmodell, Wasserfallmodell), die typischen Fragestellungen gezielter und somit effizienter adressieren als ein allgemeines Softwareentwicklungsmodell. Zusätzlich vermag es Entwicklungsfehler zu vermeiden, indem es als Checkliste dient. Abbildung 1 zeigt das Vorgehensmodell mit seinen fünf Phasen für den Entwurf, die

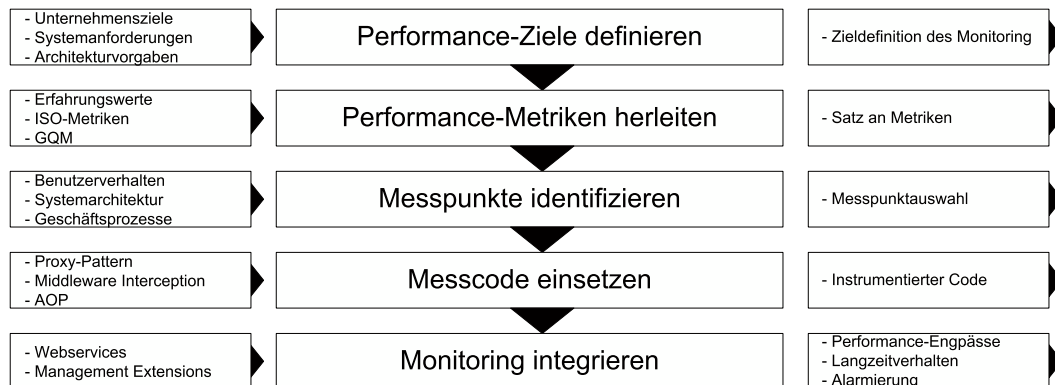


Abbildung 1. Vorgehensmodell für ein Performance-Monitoring

Implementierung und die Integration von einem Performance-Monitoring in ein bestehendes Softwaresystem. Die Phasen werden in den folgenden Unterabschnitten im Detail beschrieben werden. Es ist noch zu erwähnen, dass Rückschritte und Iterationen bei der Anwendung des Vorgehensmodells möglich sind, da das Monitoring gegebenenfalls angepasst werden muss, wenn sich die Systemarchitektur oder die Monitoring-Anforderungen ändern oder konkretisieren. Insbesondere sind Anpassungen nach der ersten Erprobungsphase zu erwarten, da oftmals erst anhand von ersten Messergebnissen festgestellt werden kann, welche Messpunkte und Metriken sich zum effizienten Erreichen der spezifizierten Ziele eignen. Daher sollte die Implementierung eine hohe Wartbarkeit besitzen, um Anpassungen des Monitorings zu unterstützen.

2.1 Performance-Ziele definieren

Die Datenerfassung für ein effektives und effizientes Monitoring muß *zielgerichtet* erfolgen. Dafür müssen die Ziele klar spezifiziert und dokumentiert werden. Zudem ist dies wichtig, um Nachvollziehbarkeit zu gewährleisten und um die Wartbarkeit des Softwaresystems nicht zu gefährden.

Bei der Identifikation der Monitoring-Ziele (z.B. Überwachung, Optimierung), können beispielsweise Unternehmensziele, Systemanforderungen oder Architekturvorgaben als Grundlage dienen. Zu späteren Zeitpunkten sollte stets überprüft werden, ob die Ziele erreicht wurden.

2.2 Performance-Metriken herleiten

Im zweiten Schritt werden Performance-Metriken mit der Goal-Question-Metric-Methode (GQM) [2] bestimmt und verbreitete Metriken aus der Literatur (z.B. [3] und [4]) eingesetzt. GQM leitet Metriken von den Zielen (Goals) über Fragen (Questions) her. Die Fragen sind eine Verfeinerung der Ziele um das Zielverständnis zu verbessern und spätere Überprüfungen zu ermöglichen. Auf Basis der Fragen werden zuletzt die Metriken bestimmt. Werden die Metriken sorgfältig ausgewählt, kann vermieden werden, dass unnötige oder falsche Daten erhoben werden.

2.3 Messpunkte identifizieren

Wir verstehen unter einem *Messpunkt* den Ort in der Anwendungsarchitektur, der für die Datenerfassung mit Anwendungscode erweitert wird.

Die Systemarchitekturbeschreibungen können markante Architekturpunkte als Kandidaten für Messpunkte oft schon erkennen lassen. Genauere Architekturanalysen können in Kombination mit Benutzerverhaltensprofilen erfolgen, indem Vorhersageverfahren für die Zuverlässigkeit oder Performance für eine Sensibilitätsanalyse benutzt werden, um kritische Architekturelemente zu finden.

Alternativ können auch Techniken wie Lasttests oder Profiling uninteressante Punkte für Performancemessungen ausschließen, insbesondere da erfahrungsgemäß lediglich ein kleiner Teil des Programmcodes für den Großteil der Gesamtausführungszeit einer Software-Anwendung verantwortlich ist (vgl. [5]). Die Anzahl der Messpunkte sollte einen Kompromiss zwischen Datenqualität und Monitoring-Overhead bilden, der ausgehend von den in den vorherigen Phasen ermittelten Zielen und Fragen gefunden werden kann.

2.4 Messcode einsetzen

In dieser Phase wird die Programmlogik für das Monitoring implementiert und mit der zu überwachenden Applikation auf technischer Ebene verbunden. Grundsätzliche lassen sich für das Verbinden zwei Klassen von Verfahren unterscheiden: (1.) Verfahren die den Quellcode der Applikation erweitern (empfohlen z.B. Aspektorientierte Programmierung (AOP) [6] oder ähnliche Techniken; nur im Ausnahmefall manuelle Codeerweiterung), oder (2.) die sogenannte Middleware Interception. AOP verknüpft Monitoringlogik mit Programmlogik zur Kompilier- oder Laufzeit, Middleware Interception beobachtet den Nachrichtentransfer zwischen den Applikationskomponenten auf der tieferen Middlewareebene.

2.5 Monitoring integrieren

Nachdem der Messcode mit der Applikation verbunden ist, erfolgt noch eine Integration der Messpunkte in eine Monitoring-Infrastruktur. Diese ist dafür verantwortlich, die einzelnen Messdaten von den verteilten Messpunkten zusammenzubringen und zu verwalten. Dies kann beispielsweise so aussehen, dass ein zentrales, separates System alle Messpunkte regelmäßig abgefragt, die Daten persistent speichert und für Drittanwendungen zur Verfügung stellt (z.B. für Langzeitanalysen, Alarmierungen).

3 Die Infrastruktur

Die Monitoring-Infrastruktur setzt sich zum einen aus einem Performance-Monitor und zum anderen aus einer Integrationsanwendung zusammen. Der Performance-Monitor stellt dabei eine Komponente dar, die vom Applikationsserver aufgerufen wird, um Monitoring auf Applikationsebene zu aktivieren. Das Monitoring erfolgt dabei automatisch an durch Annotationen definierten Messpunkten in der zu überwachenden Applikation.

Die Integrationsanwendung stellt eine entkoppelte Anwendung dar, die die Performance-Daten der auf mehreren Servern verteilten Performance Monitore verbindet. Weiterhin ist sie in der Lage, die Performance Monitore in regelmäßigen Abständen zu kontaktieren und dessen Daten persistent zu speichern, um sie möglichen Drittanwendungen bereitzustellen. Letztere können diese Daten beispielsweise für Alarmierungen oder Langzeitanalysen weiterverwenden.

Bei der Implementierung des Performance Monitors kam die aspektorientierte Java-Erweiterung AspectJ sowie die Management-Erweiterung JMX zum Einsatz. Mit Hilfe von AspectJ wird an den definierten Messpunkten zusätzliche Funktionalität „eingewebt“, welche die Performance-Messungen durchführt.

4 Einsatz bei einem Telekommunikationsunternehmen

Im Rahmen der Evaluationsphase wurde der entwickelte Monitoring-Ansatz bestehend aus dem Vorgehensmodell und der Infrastruktur an einem Java-basierten Portalsystem für rund 277.000 Kunden (Stand: 31.12.2005) des im norddeutschen Raum tätigen Telekommunikationsunternehmens EWE TEL getestet. Als eines der größten regionalen TK-Unternehmen Deutschlands bietet EWE TEL eine hohe Vielfalt an Sprach-, Internet- und Datendiensten. Zu EWE TEL gehört ebenfalls die Bremer Marke nord-Com.

Im Evaluationsszenario wurde der Servlet Container Apache Tomcat 5.5.12 sowie der Applikationsserver Bea WebLogic 9.1 eingesetzt. Bei dem angesprochenen Kundenportalsystem handelte es sich um eine verteilte Anwendung, bei der Messpunkte in Applikationen gesetzt wurden, die auf unterschiedlichen Applikationsservern installiert waren. Der Messcode wurde dabei in einem ersten Evaluationsszenario zur Laufzeit mit Hilfe des so genannten Load-time Weaving (LTW) von AspectJ eingewebt. In einem weiteren Szenario kam Compile-time Weaving (CTW) zum Einsatz, bei dem Messcode bereits während des Build-Prozesses eingewebt wurde, da LTW in Kombination mit RMI-Kommunikation, aufgrund eines bis dahin ungelösten konzeptionellen Problems von AspectJ 1.5.0, massive Speicherprobleme zutage brachte.

Der Monitoring-Ansatz mit Integration der Messpunkte durch CTW befindet sich seit Anfang 2006 erfolgreich im operativen Betrieb und liefert Performance-Daten, die zur Langzeitüberwachung, Alarmierung und Ressourcenbedarfsplanung genutzt werden.

5 Zusammenfassung und Ausblick

In diesem Artikel wurde ein Vorgehensmodell sowie eine Implementierung in Form einer Infrastruktur für ein Performance-Monitoring von Informationssystemlandschaft-

ten vorgestellt. Im Vorfeld dieser Arbeit konnte kein adäquates Vorgehensmodell für die systematische Integration eines Performance-Monitoring in der Literatur gefunden werden. Sicherlich gibt es zahlreiche Vorgehensmodelle, die für die Softwareentwicklung eingesetzt werden (Wasserfallmodell, Spiralmodell), welche aber nicht auf kritische Aspekte von Monitoring eingehen. Daher wurde ein konkretes Vorgehensmodell für Performance-Monitoring entwickelt, welches vorgefertigte Phasen für die Entwicklung eines Monitorings anbietet. Im Gegensatz zum Vorgehensmodell lassen sich in der Literatur einige ähnliche Monitoring-Infrastruktur-Architekturen finden (z.B. [4,7,8,9,10]). Eine alternative AOP-basierende Monitoring-Infrastruktur ist durch das Glassbox Framework gegeben [11].

Das in diesem Artikel vorgestellte Vorgehensmodell sowie die Infrastruktur konnte sich im Praxiseinsatz bewähren und werden in einem großen Kundenportalsystem eines regionalen TK-Unternehmens zur Langzeitüberwachung, Alarmierung und Ressourcenbedarfsplanung eingesetzt. Eine Erweiterung des Ansatzes um eine Protokollierung von Fehlernachrichten ist in der Entwicklung. Als Langzeitvision sind zudem einfache Selbstheilungsmechanismen denkbar, wie sie z.B. von IBMs Autonomic Computing Kampagne [12] propagiert werden, um im Fehlerfall einem Systemausfall entgegenwirken könnten.

Literatur

1. IEEE Standards Board: IEEE standard glossary of software engineering terminology—IEEE std 610.12-1990 (2002)
2. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. In Marciniak, J.J., ed.: Encyclopedia of Software Engineering. Volume 1. John Wiley & Sons (1994) 528–532
3. ISO/IEC Standard: Software Engineering – Product Quality – Part 2: External Metrics. ISO Standard 9126-2, ISO/IEC (2003)
4. Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. 1 edn. John Wiley & Sons (1991)
5. Boehm, B.W., Papaccio, P.N.: Understanding and controlling software costs. IEEE Transactions on Software Engineering **14**(10) (1988) 1462–1477
6. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In Akşit, M., Matsuoka, S., eds.: Proceedings European Conference on Object-Oriented Programming. Volume 1241. Springer-Verlag, Berlin, Heidelberg, and New York (1997) 220–242
7. Harkema, M., Quartel, D., Gijsen, B.M.M., van der Mei, R.D.: Performance monitoring of java applications. In: Proceedings of the 3rd International Workshop on Software and Performance (WOSP-02), New York, ACM Press (2002) 114–127
8. Li, J.: Monitoring of component-based systems. Technical Report HPL-2002-25R1, Hewlett Packard Laboratories (2003)
9. Hoffman, B.: Monitoring, at your service. ACM Queue **3**(10) (2005) 34–43
10. Kreger, H.: Java management extensions for application management. IBM Systems Journal **40**(1) (2001) 104–129
11. Ron Bodkin: Glassbox Inspector. (2006) <https://glassbox-inspector.dev.java.net/>, Letzter Besuch: 05. Juni 2006.
12. Kephart, J., Chess, D.: The vision of autonomic computing. IEEE Computer **36**(1) (2003) 41 – 50