Electronic Communications of the EASST
Volume 17 (2009)

Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009 (WowKiVS 2009)

Integriertes Performance-Monitoring von SOA-Anwendungen

Markus Schmid, Jan Schäfer, Reinhold Kröger

12 pages

Guest Editors: M. Wagner, D. Hogrefe, K. Geihs, K. David
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Integriertes Performance-Monitoring von SOA-Anwendungen

Markus Schmid, Jan Schäfer, Reinhold Kröger

Fachhochschule Wiesbaden, Labor für Verteilte Systeme
Kurt-Schumacher-Ring 18, 65197 Wiesbaden
{schmid|jan.schaefer|kroeger}@informatik.fh-wiesbaden.de
http://wwwvs.informatik.fh-wiesbaden.de

Abstract: Der Beitrag stellt einen Ansatz zur Integration von Performance-Monitoring-Aspekten in serviceorientierte Architekturen (SOAs) vor. Der Fokus liegt hierbei auf der konsistenten Instrumentierung bereits existierender SOA-Dienste. Hierfür wird ein im Rahmen des Projekts *Per*formance *Man*agement of *Ent*erprise *A*pplications (PerManEntA)[1] entwickeltes Werkzeug vorgestellt, das eine modellbasierte Instrumentierungsunterstützung mit Hilfe grafischer Marker vorsieht. Der Ansatz ermöglicht die Verwendung einheitlicher Monitoringschnittstellen und -funktionalität für alle in einer SOA kooperierenden Dienste und die konsistente Instrumentierung mit Hilfe teilautomatisierter, werkzeuggestützter Instrumentierung und Quellcodegenerierung. Im Ausblick werden Möglichkeiten zur Integration mit unterschiedlichen Ansätzen für das Service Level Management im SOA-Kontext aufgezeigt.

Keywords: Performance Monitoring, Management, Instrumentierung, ARM, SCA, SLM, SOA

1 Einleitung

Die Realisierung von Geschäftsanwendungen mit Hilfe einer serviceorientierten Architektur (SOA) bietet durch die implementierungsunabhängige Spezifikation von Dienstschnittstellen und die Möglichkeit der Choreografie von Diensten zu Geschäftsprozessen viele Vorteile für den Betreiber. Allgemein wird die Einführung von SOA-basierten Anwendungen mit der daraus resultierenden Flexibilität und Skalierbarkeit der IT-Infrastruktur begründet [HHV06, HK04]. Betrachtet man allerdings die technische Architektur einer SOA-Anwendung als Ganzes, kann man im Vergleich zu traditionellen Anwendungsserver-basierten Multi-Tier-Anwendungen eine deutliche Komplexitätszunahme feststellen.

In komplexen, geschäftskritischen Umgebungen ist ein einheitliches Performance-Monitoring aller Geschäftsprozesse und damit der beteiligten Softwarekomponenten notwendig, um eine zuverlässige Performance-Sicht der Gesamtanwendung bereitzustellen. Neben der eigentlichen Geschäftsfunktionalität muss jeder geschäftskritische SOA-Dienst daher auch eine Schnittstelle für Monitoring-Informationen zur Verfügung stellen. Gleichzeitig kann nur ein konsistentes Monitoring die notwendige Grundlage zur Realisierung eines effektiven *Service Level Managements (SLM)* einzelner Dienste samt unterlagerter Infrastruktur bilden.

[1] Projekt gefördert durch das BMB+F, Förderkennzeichen 1706X07

Idealerweise werden Schnittstellen und die dazugehörige Instrumentierung für das Performance-Monitoring zur Laufzeit bereits während des Entwurfs der einzelnen Dienste vorgesehen und als integrale Bestandteile des Software-Designs betrachtet. Häufig werden jedoch existierende Anwendungen als Dienste in eine SOA-Infrastruktur integriert, sodass auch für bestehende Software-Komponenten ein Weg zur nachträglichen Integration geeigneter Monitoring-Schnittstellen aufgezeigt werden muss. Eine fehlerfreie, konsistente Instrumentierung komplexer Anwendungen ist jedoch kein einfaches Unterfangen, da Instrumentierungscode i.d.R. an unterschiedlichsten Stellen ergänzt werden muss und Anwendungsentwickler in der Regel wenig Erfahrung mit entsprechenden APIs besitzen.

Vor diesem Hintergrund stellt dieser Beitrag einen Ansatz zur IDE-gestützten Integration von Performance-Monitoring-Aspekten in existierende SOA-Dienste vor, der die grafische Auszeichnung von relevanten Messpunkten im Quellcode einer Anwendung ermöglicht und so eine fehlerfreie und konsistente Instrumentierung sicherstellt. Er lässt sich mit einem parallel von uns entwickelten Ansatz zur modellgetriebenen Instrumentierung neuer Dienste (vgl. [SSK08]) zu einer einheitlichen Monitoring-Umgebung für SOA-Dienste verbinden. Diese Monitoring-Umgebung bildet die dann Grundlage für ein Performance-Monitoring auf Ebene von SOA-Anwendungen, welche als Komposition unterschiedlicher Dienste aufgefasst werden können. Gleichzeitig können die durch die Instrumentierung gewonnenen Daten zur Überwachung und zum IT-Management der Implementierung einzelner Dienste verwendet werden.

In Abschnitt 2 wird zunächst auf die Service Component Architecture, eine Spezifikation zur Strukturierung von SOA-Anwendungen eingegangen, Abschnitt 3 stellt Ansätze zur Quellcode-Instrumentierung vor. Die Abschnitte 5 und 6 beschreiben das zugrundeliegende Instrumentierungsmodell und den Ansatz zur IDE-gestützten Instrumentierung von SOA-Diensten. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick auf Integrationsmöglichkeiten mit bestehenden SOA-Managementansätzen.

2 Service Component Architecture

Eine relativ neue Spezifikation zur Beschreibung statischer Abhängigkeiten zwischen Komponenten innerhalb einer SOA ist die *Service Component Architecture (SCA)*. SCA hat zum Ziel, eine einheitliche Sicht auf SOA-Komponenten bereitzustellen und ihr Zusammenspiel innerhalb der Architektur strukturiert zu modellieren. Die derzeit vorliegende Version 1.0 der SCA-Spezifikation wurde von der *Open SOA Collaboration*[2], einem Zusammenschluss mehrerer im SOA-Umfeld aktiver Unternehmen (u.a. IBM, Sun, Oracle, SAP und BEA) entwickelt. Der weitere Standardisierungsprozess ist in der Zwischenzeit an OASIS übergeben worden.

SCA definiert eine Komponente (*SCA Component*) als ein Element mit beliebig vielen Schnittstellen (*Services*) und Abhängigkeiten (*References*). Ein Service kann durch andere SCA Components referenziert werden, was dann als *Binding* bezeichnet wird. Des Weiteren definiert SCA noch *Properties*, die zur Laufzeit von einer Komponente verwendet werden können.

SCA erlaubt die Gruppierung mehrerer SCA Components mitsamt ihrer Bindings zu einem *SCA Composite*, das sich nach außen wiederum als eine SCA Component darstellt und wie diese über Services, Properties und References verfügt.

[2] Details unter http://www.osoa.org

3 APIs zur Quellcode-Instrumentierung

Zur Quellcodeinstrumentierung von Anwendungen stehen zahlreiche Technologien zur Verfügung. Neben klassischen Logging-Schnittstellen, wie z. B. dem in der Java-Welt verbreiteten *log4j*-API [Gü02], existiert speziell zur detaillierten Vermessung von Vorgängen in verteilten Anwendungen das von der Open Group spezifizierte *Application Response Measurement* (ARM) 4.0 API [Ope03].

ARM unterstützt die Vermessung von mit Start- und Stoppunkten ausgezeichneten Quellcodeabschnitten *(Units of Work)* in verteilten Anwendungen (z.B. zur Antwortzeitmessung). Diese Units of Work werden im ARM-Kontext als *ARM-Transaktionen* bezeichnet und können zusätzlich mit Kontext-Informationen angereichert werden (z.B. CPU-Last zum Messzeitpunkt). Des Weiteren unterstützt ARM die Verknüpfung verteilter Messungen durch die Verwendung von eindeutigen Tokens, sogenannten *ARM-Korrelatoren*, die nachgeordneten Messungen übergeben werden können. Damit lassen sich zum Auswertungszeitpunkt detailliert Hierarchien von Einzelmessungen aufschlüsseln, die z.B. eine Aufruffolge über mehrere Komponenten hinweg nachverfolgbar machen. Für kritische Anwendungsgebiete kann es erforderlich sein, einzelne Dienste und deren Implementierung detailliert zu überwachen (einschließlich der beteiligten Middleware-Plattformen und nachgelagerten Systeme). Zur nahtlosen Integration stellen viele Middleware-Komponenten (z. B. IBM WebSphere, vgl. [Web06]) selbst ARM-Schnittstellen bereit, die dann mit eigenen Messungen verknüpft werden können.

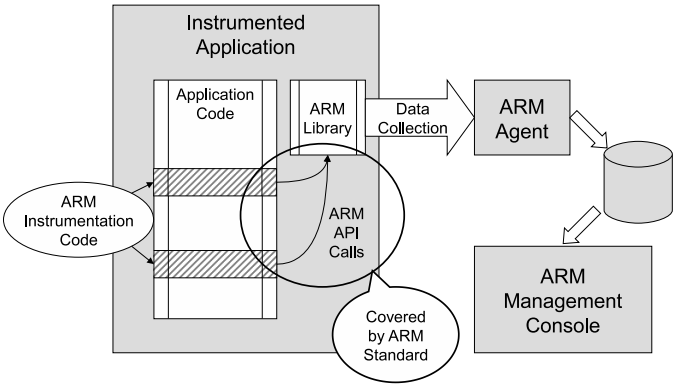

Abbildung 1: ARM-Instrumentierung

Abbildung 1 zeigt, auf welche Weise Zeitmessungen in einer ARM-instrumentierten Anwendung realisiert werden. Die im ARM-Stardard spezifizierten Start- und Stoppanweisungen im Quellcode der instrumentierten Anwendung resultieren in API-Aufrufen einer herstellerspezifischen ARM-Bibliothek. Diese führt die eigentlichen Zeitmessungen durch und leitet die gesammelten Daten an einen zentralen ARM-Agenten weiter. Hier können dann Ergebnisse verteilter Messungen mithilfe der ARM-Korrelatoren verknüpft werden, sodass eine anwendungsübergreifende Auswertung erfolgen kann. Der ARM-Standard beschränkt sich auf die Spezifikation des ARM-APIs; eine Standardisierung von ARM-Agenten und Auswertungswerkzeugen ist nicht vorgesehen. Durch einfachen Tausch der verwendeten ARM-Bibliothek kann eine bestehende ARM-Instrumentierung in die Monitoring-Umgebung eines anderen Anbieters überführt werden, ohne dass Anpassungen am Quellcode vorgenommen werden müssen. Eine sogenannte *Null-Bibliothek* kann zur Anwendung hinzugebunden werden, wenn keine Auswertung von Messungen erwünscht ist.

4 Performance-Monitoring von SOA-Diensten

4.1 Überblick

Abbildung 2 zeigt beispielhaft den typischen Aufbau einer SOA-Anwendung. Auf oberster Abstraktionsebene werden in einer SOA Workflows als technische Realisierung einzelner Geschäftsprozesse definiert, die bei ihrer Abarbeitung auf die Geschäftsschnittstellen unterschiedlicher Dienste zugreifen. Diese sind in der Grafik hell gekennzeichnet. Neben der eigentlichen Geschäftsfunktionalität können SOA-Dienste auch weitere Funktionen bereitstellen, in der Abbildung erkennt man eine dunkel gekennzeichnete Schnittstelle zum Abruf von Performance-Kenngrößen.

Außerhalb des SOA-Fokus befindet sich dabei die eigentliche Implementierung eines Dienstes. Hierbei kann es sich beispielsweise um eine existierende Anwendung handeln, die zur SOA-Integration um eine standardisierte Geschäftsschnittstelle erweitert wurde. Häufig existieren in Firmen oder Abteilungen (in der Grafik als administrative Domänen bezeichnet) bereits individuelle Monitoring- und Managementansätze zur Überwachung solcher Anwendungen. Derartige Lösungen zeichnen sich aber typischerweise durch einen Mangel an Standardisierung aus und eignen sich deshalb nicht für die Bereitstellung einheitlicher Performance-Informationen über Dienst- und Domänengrenzen hinweg.

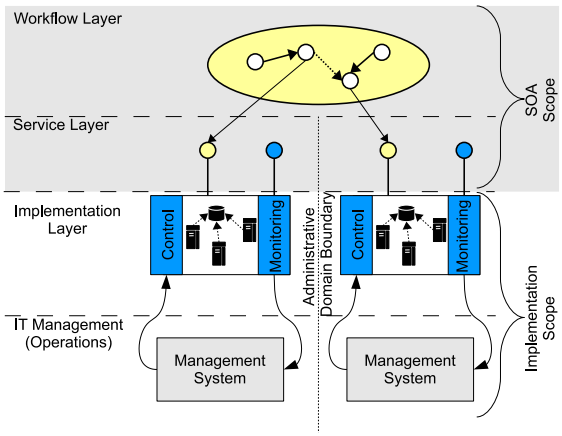

Abbildung 2: Monitoring und IT-Management in SOA-Anwendungen

Die in Abschnitt 3 vorgestellte ARM-Technologie kann durch die Unterstützung für verteilte Messungen dazu beitragen, in einem solchen Umfeld wertvolle dienstübergeifende Performance-Daten zu liefern, die die Grundlage für eine detaillierte Bewertung der Performance eines SOA Workflows bilden können. Im Vergleich zu Zeitmessungen innerhalb einer entsprechend modifizierten Workflow-Engine erlaubt der hier vorgestellte Ansatz deutlich feingranularere Messungen, die ggf. auch um implementierungsspezifische Informationen angereichert werden können. Gleichzeitig kann die ARM-Instrumentierung auch für das Performance-Monitoring im Rahmen des IT-Managements einer geschäftskritischen Dienst-Implementierung als Datenquelle dienen.

4.2 Einheitlicher Zugriff auf Performance-Daten

Um die Ergebnisse einer ARM-Instrumentierung über Dienst-Grenzen hinweg nutzen zu können, muss eine einheitliche Schnittstelle zum Abrufen von Kenngrößen bereitgestellt werden, die die durch Performance-Instrumentierung erhobenen Kenngrößen mithilfe einer *Monitoring-Komponente* nach außen hin bereitstellt.

Da der ARM-Standard kein einheitliches API für das Auslesen der Kenngrößen spezifiziert, ist zur Realisierung der Monitoring-Komponente die Nutzung einer herstellerspezifischen Schnittstelle unumgänglich. In der praktischen Umsetzung des hier vorgestellten Ansatzes wurde die kommerzielle ARM-Implementierung der Firma tang-IT [tICG04] eingesetzt, für die eine Java-basierte Client-API zur Verfügung steht. Unter Nutzung dieser API konnte eine generische Monitoring-Komponente entwickelt werden, die lediglich mit einer geeigneten Konfiguration zum Auslesen der erzeugten Messungen versehen werden muss.

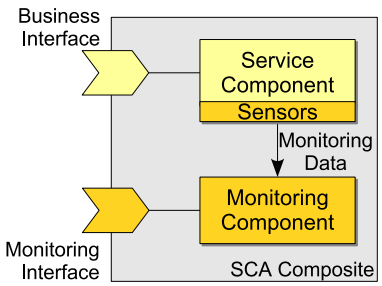

Abbildung 3: Integration von Geschäfts- und Monitoring-Schnittstelle

Aus SOA-Sicht koexistieren Geschäfts- und Monitoring-Schnittstelle einer Dienstimplementierung aber zunächst als unabhängige Komponenten; die auf Implementierungsebene gegebene Abhängigkeit wird nach außen nicht sichtbar. Wie in Abschnitt 2 gezeigt, ist es mithilfe von SCA möglich, die auf Implementierungsebene bestehende Abhängigkeit zwischen Monitoring- und Geschäftskomponente in Form eines SCA Composites zu beschreiben. Hauptvorteil einer SCA-basierten Kopplung von Geschäfts- und Monitoring-Funktionalität einer Komponente ist die Möglichkeit zur transparenten Integration der Monitoring-Funktionalität in bestehende Umgebungen. Ein auf diese Weise instrumentierter Dienst kann von bestehenden Komponenten weiterhin unverändert adressiert werden, da die Geschäftsschnittstelle zur Integration des Monitoring-Anteils nicht modifiziert werden muss. Trotzdem ist über diese Kopplung eine eindeutige Zuordnung der neu erzeugten Monitoring-Schnittstelle zu einer Geschäftskomponente möglich.

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  targetNamespace="http://sample"
  xmlns:sample="http://sample"
  name="Calculator">

  <service name="CalculatorService" promote="CalculatorServiceComponent">
     <interface.java interface="calculator.CalculatorService"/>
        <binding.ws/>
  </service>

  <service name="MonitoringService" promote="MonitoringComponent">
     <interface.java interface="de.fhwiesbaden.cs.vs.MonitoringService"/>
        <binding.ws/>
  </service>

  <component name="CalculatorServiceComponent">
     <implementation.java class="calculator.CalculatorServiceImpl"/>
  </component>

  <component name="MonitoringComponent">
     <implementation.java class="de.fhwiesbaden.cs.vs.MonitoringServiceImpl"/>
  </component>

</composite>
```

Abbildung 4: Beispiel: Gruppierung von Geschäfts- und Monitoring-Funktionalität

Abbildung 4 zeigt als Beispiel die Gruppierung eines Web Services (*Calculator*) mit der dazugehörigen Monitoring-Schnittstelle, in dem die jeweiligen Komponenten als Java-Klassen referenziert werden. Durch den Ausdruck `<binding.ws/>` wird für die Geschäfts- und die Monitoring-Funktionalität eine entsprechende Web-Service-Schnittstelle bereitgestellt. Somit können instrumentierte SOA-Dienste durch eine SCA-Laufzeitumgebung bereitgestellt werden.

In den folgenden Abschnitten wird genauer auf unseren Ansatz für die Performance-Instrumentierung bestehender SOA-Dienste (vgl. Abschnitte 5 und 6) eingegangen, die die Bereitstellung einer einheitlichen Instrumentierung und somit von Laufzeit-Performance-Kenngrößen für die hier skizzierte Schnittstelle zum Ziel hat.

5 Zugrundeliegendes Instrumentierungsmodell

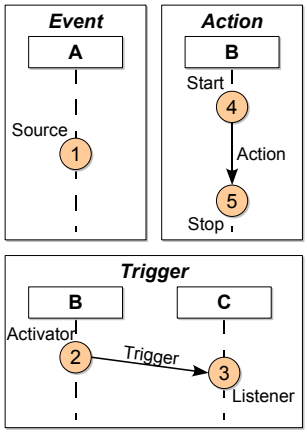

Abbildung 5: Im Modell verwendete Basismuster

Als Grundlage für den in diesem Beitrag vorgestellten IDE-gestützten Instrumentierungsansatz von SOA-Diensten wurde eine Reihe abstrakter *Instrumentierungsmuster* entwickelt. Diese beschreiben im Kontext von SOA-Diensten häufig auftretende Interaktionsmuster, deren Erfassung oder Vermessung zur Laufzeit von Interesse sein können.

Instrumentierungsmuster bestehen aus *Instrumentierungspunkten*, die über Kanten miteinander verknüpft werden. Instrumentierungspunkte markieren dabei Stellen im Programmablauf, an denen z. B. eine Zeitmessung starten oder enden soll. Über die Zuordnung von *Rollen* zu einzelnen Punkten innerhalb eines Musters können diese Funktionen explizit ausgedrückt werden. Instrumentierungsmuster werden in *Basismuster* und *Komplexe Muster* unterteilt. Im Modell verwendete Basismuster sind die in Abbildung 5 dargestellten Muster *Event*, *Trigger* und *Action*.

Ein *Event* entspricht einem unabhängigen Ereignis im Ablauf, das keine Auswirkung auf andere Punkte des Systems hat. Für ein Event kann später z. B. eine Lognachricht im Quellcode generiert werden. Ein *Trigger* beschreibt eine Beziehung zwischen zwei Ereignissen in unterschiedlichen Komponenten, in der das erste Ereignis (Rolle *Activator*) durch sein Auftreten das zweite Ereignis (*Listener*) auslöst. Ein Trigger kann z. B. für die Vermessung einer Einwegnachricht verwendet werden, bei der die Erfassung von Versende- und Empfangszeitpunkt relevant ist. Eine *Action* bezeichnet einen durch einen Ein- und einen Austrittspunkt (*Start* und *Stopp*) begrenzten Bearbeitungsschritt innerhalb einer Komponente. Eine Action kann z. B. für die Messung der Ausführungsdauer eines Algorithmus oder eines Aufrufs einer entfernten Komponente verwendet werden.

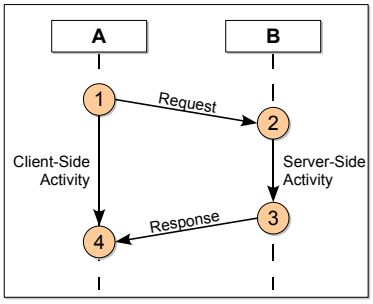

Abbildung 6: Request / Response-Muster

Um anspruchsvollere Instrumentierungsszenarien (z. B. Client- und Server-seitige Vermessung eines RPC-artigen Aufrufs) abzubilden, können die beschriebenen Basismuster zu komplexen Mustern zusammengefügt werden. Ein Beispiel hierfür ist das in Abbildung 6 dargestellte *Request/Response*-Muster, das zur Auszeichnung der Verknüpfung von Zeitmessungen in den Komponenten `A` und `B` genutzt werden kann. In diesem Muster werden zwei Trigger und zwei Actions miteinander kombiniert, um eine typische Client/Server-Interaktion abzubilden.

Mithilfe der hier beschriebenen Instrumentierungsmuster ist es möglich, gängige Performance-Instrumentierungsszenarien auf Modellebene – und damit zunächst unabhängig von einer konkreten Instrumentierungstechnologie oder einem bestimmten API – zu definieren.

Bei der IDE-gestützten Instrumentierung von SOA-Diensten kann sich der Entwickler damit auf einer hohen Abstraktionsebene bewegen, die – durch die Möglichkeit der dienstübergreifenden grafischen Darstellung der Instrumentierungspattern – einen guten Überblick über die erfolgte Instrumentierung gewährleistet.

6 Das IDE-basierte Instrumentierungswerkzeug

Im Rahmen des Projekts *Per*formance *Man*agement of *Ent*erprise *A*pplications[3] (PerManEntA) wurde an der FH Wiesbaden ein Eclipse-basiertes Werkzeug zur Performance-Instrumentierung verteilter Anwendungen entwickelt, das auf dem in Abschnitt 5 beschriebenen Instrumentierungsmodell aufsetzt. Das Werkzeug erweitert die Eclipse-Entwicklungsumgebung um eine Reihe von Plugins, zur grafischen Instrumentierung von Anwendungen.

6.1 Vorgehensweise bei der Instrumentierung

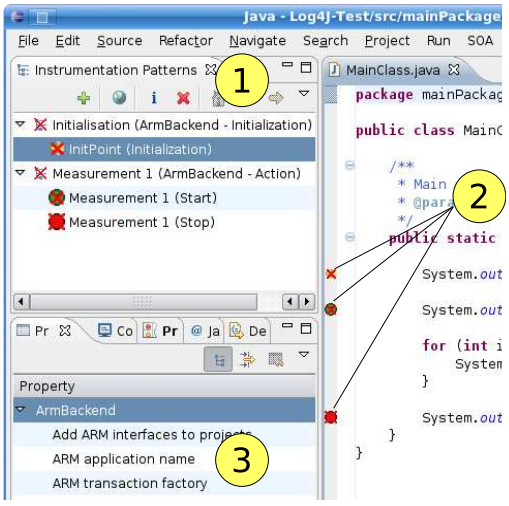

Abbildung 7: Bildschirmausschnitt des Instrumentierungswerkzeugs

Vergleichbar mit Debugger-Breakpoints können relevante Quellcodeabschnitte zur Performance-Instrumentierung markiert werden. Für SOA-Dienste sind hier vorrangig Operationen von Interesse, die über die externe Dienstschnittstelle aufgerufen werden können. Zusätzlich können jedoch auf diese Weise feingranulare Messungen innerhalb der Implementierung spezifiziert werden, um z. B. den Anteil von Datenbankabfragen oder einzelner Algorithmen an der Gesamtausführungsdauer einer Operation bestimmen zu können. Auf Knopfdruck kann für die mit Instrumentierungspunkten und -Patterns ausgezeichneten Stellen dann Instrumentierungscode generiert werden oder zuvor erzeugter Code wieder entfernt werden.

Der Instrumentierungscode wird im Rahmen des Generierungsprozesses direkt in die zu instrumentierende Datei eingefügt. Das Werkzeug er-

[3] Näheres unter http://wwwvs.informatik.fh-wiesbaden.de/projekte/permanenta.html

laubt das Gruppieren von Instrumentierungsmustern zu *Aspekten*, die gemeinsam aktiviert oder deaktiviert werken können, sodass zur Laufzeit nur relevante Teile der vorgenommenen Instrumentierung aktiv gehalten werden müssen. Dadurch lässt sich zum Einen der durch die Instrumentierung verursachte Laufzeit-Overhead minimal halten, zum Anderen können beispielsweise bei der Fehleranalyse oder zum Systemtest durch Zuschalten zusätzlicher Aspekte zeitweise detailliertere Messungen erfolgen.

Abbildung 7 zeigt einen Bildausschnitt des Instrumentierungswerkzeugs, in dem eine Instrumentierungsmusteransicht (1), grafische Instrumentierungspunkte (2) und die Eigenschaftenansicht eines einzelnen Instrumentierungspunktes zu sehen sind.

Nachdem die Performance-Instrumentierung eines SOA-Dienstes wie beschrieben erfolgt ist, kann die Auswertung der Performance-Messungen mithilfe der in Abschnitt 4.2 beschriebenen generischen Monitoring-Komponente erfolgen, auf die aus der SOA über die bereitgestellte Monitoring-Schnittstelle zugegriffen werden kann.

Zum jetzigen Zeitpunkt ist das Werkzeug noch nicht in der Lage, zusätzlich zur Instrumentierung die zur Kopplung von Monitoring- und Geschäftsfunktionalität eines SOA-Dienstes notwendige SCA-Beschreibung zu generieren. An dieser Stelle ist damit derzeit noch eine manuelle Ergänzung der Implementierung existierender SOA-Dienste notwendig.

6.2 Architektur des Instrumentierungswerkzeugs

Die im Rahmen des PerManEntA-Projekts entwickelte Implementierung unterstützt aktuell die Instrumentierung von Java-Quellcode mithilfe von ARM und Log4J. Durch die modulare Architektur des Werkzeugs ist jedoch die Möglichkeit der Erweiterung auf andere Programmiersprachen und Instrumentierungs-APIs gegeben. Zu diesem Zweck sind API-spezifische Teile in eigenen Eclipse-Plugins, den sogenannten *Instrumentierungs-Backends*, gekapselt.

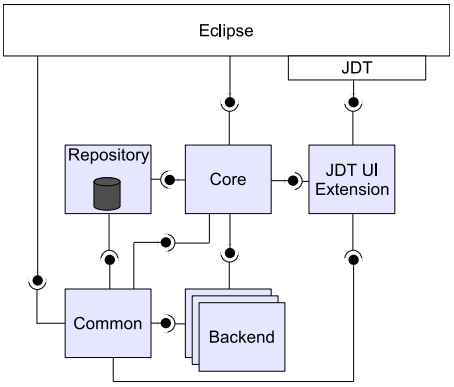

Abbildung 8: Architektur des Instrumentierungswerkzeugs

Abbildung 8 zeigt das Zusammenspiel der einzelnen Teile des Werkzeugs, die jeweils als einzelne Eclipse-Plugins realisiert sind. Zentrales Element der Architektur ist das *Core*-Plugin, das die Initialisierung des Gesamtsystems übernimmt und das Zusammenspiel zwischen der Entwicklungsumgebung und den API-spezifischen Backends koordiniert.

In der *JDT-UI-Extension* sind spezifische Erweiterungen der Eclipse Java Entwicklungsumgebung (JDT) realisiert, hierzu zählen beispielsweise Funktionalität zur Manipulation des Editor-Buffers sowie zum Zugriff auf den durch JDT bereitgestellten Abstract Syntax Tree (AST) einer Java-Klasse.

Das *Common*-Plugin realisiert von Instrumentierungs-API und Programmiersprache unabhängige Funktionalität; hierzu zählen beispielsweise Visualisierung und Manipulation der spezifizierten Instrumentierungsmuster sowie Funktionalität zum Anstoßen der Code-Generierung oder des Entfernens von Instrumentierungscode. Da nicht notwendigerweise jede Instrumentierungs-

technologie alle der in Abschnitt 5 diskutierten Patterns unterstützt, informieren die Backends das Common-Plugin über die von ihnen unterstützten Funktionen. Zur Code-Generierung wird durch den Core das vom Benutzer gewählte Backend aufgerufen.

Die Instrumentierungsinformationen für einzelne Klassen werden durch ein *Repository*-Plugin gekapselt und im XML-Format als separate Datei im Eclipse-Workspace abgelegt.

6.3 Integration mit einem modellgetriebenen Instrumentierungsansatz

Im Rahmen des Projekts wurde parallel ein modellgetriebener Ansatz zur Instrumentierung neu zu entwickelnder SOA-Dienste entwickelt. Hierbei wird eine grundlegende Instrumentierung der einzelnen Operationen eines auf UML-Ebene modellierten SOA-Dienstes automatisiert im Rahmen des MDA-Codegenerierungsprozesses vorgenommen. Die resultierende Instrumentierung arbeitet dabei mit der gleichen Monitoring-Komponente wie der oben vorgestellte Ansatz. Zusätzlich zur eigentlichen Codegenerierung erzeugt der MDA-Transformationsprozess ein XML-Repository, sodass eine nachträgliche Verfeinerung der Instrumentierung mit dem hier vorgestellten Eclipse-basierten Instrumentierungswerkzeug möglich ist. Der MDA-basierte Instrumentierungsansatz für neu zu entwickelnde SOA-Dienste ist detailliert in [SSK08] dargestellt.

6.4 Weitere Module

Zur Unterstützung des lokalen IT-Managements von Dienst-Implementierungen wurden im Rahmen des Projekts weitere Module zur Auswertung der durch die Instrumentierung erhaltenen Performance-Informationen entwickelt. Wie die in Abschnitt 4.2 vorgestellte Monitoring-Komponente operieren diese auf der durch den ARM-Agenten bereitgestellten Datenbestand und nutzen damit ein herstellerspezifisches API.

Zur Integration der Performance-Messwerte in existierende SLM-Systems wurde ein Modul zur Überwachung von Schwellenwerten entwickelt, mit deren Hilfe komplexe Constraints über Messpunkte und Instrumentierungspatterns spezifiziert werden können. Das Modul agiert gegenüber einer existierenden IT-Managementumgebung als SNMP-Agent und kann so im Bedarfsfall mittels SNMP-Traps über Schwellwertverletzungen informieren. Die Definition entsprechender Constraints und die Überwachung von Schwellwerten ist detailliert in [Tex08] beschrieben.

7 Verwandte Arbeiten

Traditionell werden zur Vermeidung fehleranfälliger manueller Instrumentierung von Anwendungen Compiler-basierte Ansätze verfolgt; hierbei generiert ein Compiler z. B. zusätzlichen Instrumentierungscode für Funktionsaufrufe. Neuere Ansätze versuchen das Anwendungsverhalten indirekt über bereits existierende Sensorik zu bestimmen. Aguilera et al. [AMW+03] beschreiben einen Black-Box-Ansatz zur Performance-Analyse verteilter Anwendungen auf Basis der Auswertung von Kommunikations-Traces und verzichten somit vollständig auf jegliche Anwendungsinstrumentierung. Der Ansatz beschäftigt sich intensiv mit der Rekonstruktion relevanter Aufrufgraphen aus der aufgezeichneten Kommunikation. Im Gegensatz zu unserem Instrumentierungsansatz können Aufrufgraphen jedoch nur mit einer bestimmten Wahrscheinlichkeit

und im Nachhinein ermittelt werden, weiterhin ist die Granularität des Performance-Monitorings auf Dienst-Interaktionen festgelegt – feingranulare Messungen sind somit nicht möglich.

Pinpoint [CKF+02] ist ein System, das sich mit automatischer Problemerkennung in großen, dynamischen Internet-basierten Systemen beschäftigt. Pinpoint stellt eine Reihe instrumentierter Middleware-Komponenten bereit, mit deren Hilfe ein Request bei seinem Lauf durch das System beobachtet werden kann. Pinpoint erlaubt allerdings keine weitere Verfeinerung durch zusätzliche Messungen, das System verwendet einen nicht-standardisierten Messansatz und ist nicht offen für Erweiterungen.

Durch die Verwendung einer einheitlichen Monitoring-Schnittstelle können mit dem in diesem Beitrag vorgestellten Ansatz instrumentierte Dienste prinzipiell in Architekturen integriert werden, die alternative Dienste innerhalb eines Workflows basierend auf ihrer Dienstgüte auswählen. Ein Beispiel für einen solchen Ansatz ist WSQoSX [BGR+05]. Diese Architektur unterstützt ein QoS-abhängiges Binding von Diensten, wobei unter anderem die Antwortzeit eines Dienstes als Auswahlkriterium herangezogen wird. WSQoSX wählt dabei unter mehreren Diensten mit äquivalenter Funktionalität diejenigen aus, mit deren Beteiligung an die Architektur gestellte SLAs erfüllt werden können.

In [MDGA08] wird eine modellbasierte Monitoring-Infrastruktur für Composite-Services vorgestellt, die mit existierenden Managementumgebungen integriert werden kann. Im Unterschied zum hier präsentierten Ansatz konzentrieren sich die Autoren bei der Erfassung von Performance-Kenngrößen auf die Ebene des Workflow-Managementsystems, wodurch es möglich ist, eine Auswahl der zu vermessenden Operationen aus Geschäftssicht zu treffen. Durch die Beschränkung auf die Workflow-Ebene erfolgen Messungen allerdings in deutlich geringerer Granularität, Performance- oder Durchsatzmessungen können bei nachrichtenbasierter (Einweg-)Kommunikation nicht durchgeführt werden. Der hier vorgestellte Ansatz kann aber problemlos mit der in [MDGA08] diskutierten Architektur kombiniert werden, so dass sich die Vorteile beider Ansätze nutzen lassen.

8 Zusammenfassung und Ausblick

Das in diesem Beitrag geschilderte Vorgehen zur Integration von Performance-Monitoring-Funktionalität in den Designprozess von SOA-Diensten ist ein erster Schritt hin zur modellgestützten Integration weiterer Management-Funktionalität für derartige Komponenten. Mithilfe des IDE-basierten Instrumentierungswerkzeugs kann eine Performance-Instrumentierung bereits existierender Dienste erfolgen oder eine bereits vorhandene Instrumentierung weiter verfeinert werden.

Gleichzeitig bildet der Ansatz die Grundlage für ein integriertes, dienstübergreifendes Performance-Monitoring in SOA-Anwendungen. Um Abhängigkeiten zwischen Messungen über Dienstegrenzen hinweg verfolgen zu können, müssen ARM-Korrelatoren beim Aufruf eines Dienstes an diesen übergeben werden. [STTK07] und [Sch06] beschreiben die prinzipielle Vorgehensweise zur transparenten Übertragung von ARM-Korrelatoren zwischen unterschiedlichen Middleware-Komponenten. Zur Realisierung einer integrierten Performance-Monitoring für SOA-Anwendungen soll diese Vorgehensweise mit dem in dieser Arbeit vorgestellten Ansatz zusammengeführt werden.

Durch Erweiterung des bestehenden Instrumentierungsmodells erscheint auch die Integra-

tion einer universelleren Dienst-Management-Funktionalität möglich, wie sie beispielsweise in [SK08] für einzelne SOA-Dienste und Workflows vorgeschlagen wird. [SK08] beschreibt einen Ansatz zum dezentralen Performance-Management in SOAs, bei dem auf Workflow- und Dienstebene angesiedelte Management-Komponenten zur Durchsetzung Performance-bezogener Dienstgüteanforderungen kooperieren. Der hier vorgestellte Ansatz zur Performance-Instrumentierung kann dabei als Datenquelle zur Dienstüberwachung eingesetzt werden. Hierbei wäre jedoch eine engere Verknüpfung von Modell und generiertem Quellcode durch bidirektionalen Abgleich von Änderungen wünschenswert, da dies die Flexibilität bei der Entwicklung von Anwendungen erhöhen würde.

Literatur

[AMW+03] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, A. Muthitacharoen. Performance Debugging for Distributed Systems of Black Boxes. In *Proc. of ACM Symp. on Operating Systems Principles (SOSP'03)*. 2003.

[BGR+05] R. Berbner, T. Grollius, N. Repp, O. Heckmann, E. Ortner, R. Steinmetz. An approach for the Management of Service-oriented Architecture (SoA) based Application Systems. In *Enterprise Modelling and Information Systems Architectures, Proceedings, 2005*. Pp. 208–221. Oktober 2005.

[CKF+02] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, E. Brewer. Pinpoint: Problem Determination in Large, Dynamic Internet Services. In *Proceedings of the Int. Conf. on Dependable Systems and Networks (DSN'02)*. 2002.

[Gü02] C. Gülcü. Log4j Manual. 2002. http://logging.apache.org/log4j/docs/manual.html.

[HHV06] A. Hess, B. Humm, M. Voß. Regeln für serviceorientierte Architekturen hoher Qualität. *Informatik Spektrum* 6, 2006.

[HK04] T. Heimann, R. Kappes. Mit SOA aus der Kostenfalle. *IT Management*, November 2004.

[tICG04] tang IT Consulting GmbH. tang-IT ARM. 2004. http://www.tang-IT.com/arm/.

[MDGA08] C. Momm, T. Detsch, M. Gebhart, S. Abeck. Model-Driven Development of Monitord Web Service Compositions. In Harroud et al. (eds.), *Proceedings of the 15th Annual Workshop of the HP Software University Association*. Pp. 93–104. Juni 2008.

[Ope03] The Open Group. Application Response Measurement. 4.0 edition, Oktober 2003. http://www.opengroup.org/arm.

[Sch06] J. Schaefer. An Approach for Fine-Grained Web Service Performance Monitoring. In Eliassen and Montresor (eds.), *Distributed Applications and Interoperable Systems: 6th IFIP WG 6.1 International Conference, DAIS 2006, Bologna, Italy, June 14-16, 2006, Proceedings*. Pp. 169–180. Springer Verlag, June 2006.

[SK08] M. Schmid, R. Kroeger. Decentralised QoS-Management in Service Oriented Architectures. In Meier and Terzis (eds.), *Distributed Applications and Interoperable Systems: 8th IFIP WG 6.1 International Conference, DAIS 2008, Oslo, Norway, June 4-6, 2008*. Pp. 44–57. Springer Verlag, Juni 2008.

[SSK08] M. Schmid, J. Schaefer, R. Kroeger. Ein MDSD-Ansatz zum QoS-Monitoring von Diensten in Service-orientierten Architekturen. *PIK - Praxis der Informationsverarbeitung und Kommunikation* 31 / 4, 2008.

[STTK07] M. Schmid, M. Thoss, T. Termin, R. Kroeger. A Generic Application-Oriented Performance Instrumentation for Multi-Tier Environments. In *10th IFIP/IEEE International Symposium on Integrated Network Management (IM2007)*. Pp. 304–313. IEEE, Mai 2007.

[Tex08] A. Textor. Monitoring unternehmenskritischer Anwendungen unter Verwendung modellbasierter Performance Constraints. B.Sc. Thesis, FH Wiesbaden, FB Design Informatik Medien, September 2008.

[Web06] WebSphere Application Server Manual – Getting performance data from request metrics. 2006.
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.base.doc/info/aes/ae/tprf_rqenable.html