Software-Betriebs-Leitstände für Unternehmensanwendungslandschaften

Simon Giesecke, Matthias Rohr und Wilhelm Hasselbring
Graduiertenkolleg „TrustSoft"*
Software Engineering Group
Carl von Ossietzky Universität Oldenburg
{giesecke,rohr,hasselbring}@informatik.uni-oldenburg.de

Abstract: In Kontrollzentren für Telefon-, Verkehrs- oder Energieversorgungsnetzen werden Leitstände verwendet, um dem Betriebspersonal einen schnellen Überblick über die Netzarchitektur und deren gegenwärtige Eigenschaften (z.B. Auslastung) zu bieten. Leitstände sind ein grundlegender Bestandteil von Kontrollzentren z.B. für Energieversorgungsnetze. Für Softwaresysteme sind solche Überwachungs- und Steuerungssysteme bislang wenig verbreitet. Leitstände können bei der Systemüberwachung und dem Erkennen und Beheben von Störungen helfen, da Betriebsdaten im Zusammenhang überblickt werden können. Wir charakterisieren Software-Betriebs-Leitstände mit Hilfe einer Taxonomie von Software-Leitständen und beschreiben Anforderungen an solche Leitstände, beispielhaft für JavaEE-basierte Systeme.

1 Einleitung

Große, verteilte Softwaresysteme sind zunehmend das zentrale Element in vielen Unternehmen. Teil- oder Gesamtsystemausfälle sind zu vermeiden oder schnell zu beheben, um schwerwiegende Konsequenzen wie z.B. Vertragsstrafen, Dienstunterbrechungen oder Kundenabwanderung zu verhindern. Leider ist es praktisch nicht möglich, komplexe Softwaresysteme frei von Fehlern zu entwickeln [Lyu96], so dass Werkzeuge zur Systemüberwachung und Steuerung nötig sind, um schnell Probleme entdecken und isolieren sowie Problemursachen finden und beheben zu können.

Als Vorbild können Steuerungssysteme für Telekommunikations-, Verkehrs- oder Energieversorgungsnetze dienen, da auch hier hohe Verlässlichkeitsanforderungen [HR06] bestehen und die Struktur denen von verteilten komponentenbasierten Softwareanwendungen ähnelt. Für die Überwachung und Steuerung dieser Netze gibt es in Kontrollzentren großformatige Darstellungen der Netzstruktur und dessen aktuellen Zustandes (auch als Großbildprojektion, Mosaiktafel, siehe Abbildung 1). Somit können Administratoren schneller Probleme erkennen, diagnostizieren und ggf. über integrierte Schaltflächen reagieren. Bei Industrie- und Versorgungsunternehmen sind diese auch als SCADA (Supervisory Control And Data Acquisition) bezeichneten Systeme seit langem verbreitet und durch eine

*Diese Arbeit wurde gefördert durch die Deutsche Forschungsgemeinschaft, GRK 1076/1.

Abbildung 1: Leitstand eines Kraftwerks [Wik06]

Vielzahl von Standards (z.B. IEC 61850 [IEC05], ICCP-TASE.2 [IEC98]) unterstützt.

Für Softwaresysteme auf Anwendungsebene gibt es solche Überwachungs- und Steuerungsdarstellungen bisher nur vereinzelt; in der Regel wird eine Softwareanwendung als „Black Box“ betrachtet. Oftmals haben Unternehmen weder vollständige Beschreibungen der Anwendungsarchitektur, noch eine einheitliche Überwachung auf Anwendungsebene. Gerade die Verbindung von Architekturbeschreibung und Laufzeitverhalten ermöglicht bei Leitständen die effiziente Überwachung und Diagnose, da erst hierdurch lokal erfasste Messwerte in einen Zusammenhang gebracht werden. Ein Grund für das Fehlen von Softwarearchitekturbeschreibungen sind oft die kurzen Weiterentwicklungsphasen, bei denen bestehende Dokumentation nicht mitgepflegt wird. Überwachungsfunktionalität (*monitoring*) wird oftmals erst nach und nach hinzugefügt, was zu vielen heterogenen und unverbundenen Überwachungskomponenten führt.

Überblick In diesem Beitrag schlagen wir zunächst eine Taxonomie von Software-Leitständen vor (Abschnitt 2) und definieren den Begriff des *Software-Betriebs-Leitstands*. Anschließend diskutieren wir die von uns ermittelten Anforderungen an Software-Betriebs-Leitstände im allgemeinen (Abschnitt 3) sowie exemplarisch für Unternehmensanwendungslandschaften, die vorrangig auf JavaEE basieren (Abschnitt 4).

2 Taxonomie

Leitstände sind generell technische Einrichtungen zur Unterstützung des Ablaufs von Prozessen. Außerhalb der Softwarewelt kommen zahlreiche Typen von Leitständen zum Einsatz: Produktions-/Fertigungsleitstände, Kraftwerksleitstände, (Strom-)Netzleitstände, Gebäude-, Verkehrs- und Notrufleitstände.

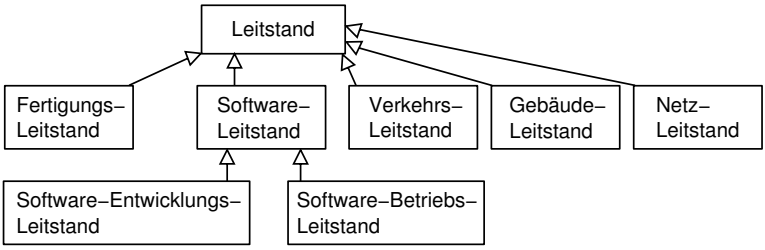

Abbildung 2: Softwareleitstände-Taxonomie

Unter einem Softwareleitstand verstehen wir zunächst einen Leitstand zur Unterstützung von Prozessen, die in Zusammenhang mit Software-intensiven Systemen stehen. Diese Prozesse lassen sich zunächst differenzieren in die Entwicklung und den Betrieb von Software, entsprechend lassen sich auch Software-Entwicklungs-Leitstände und Software-Betriebs-Leitstände identifizieren. Dies ist zusammenfassend in Abbildung 2 dargestellt.

3 Anforderungen an Software-Betriebs-Leitstände

3.1 Verbesserung der Verfügbarkeit durch Software-Betriebs-Leitstände

Gegenüber dem Status Quo soll die Investition in einen Software-Betriebs-Leitstand die Wirtschaftlichkeit des Betriebs eines Software-intensiven Systems verbessern. Kosteneinsparungen werden insbesondere durch die Erhöhung der Verfügbarkeit, mithin der Reduktion der Ausfallkosten, erreicht. Die Verfügbarkeit wird üblicherweise über die Formel

$$\text{Verfügbarkeit} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

definiert, wobei MTTF für die mittlere Zeit bis zu einem Ausfall *(mean time to failure)* und MTTR für die mittlere Zeit bis zum Abschluss der Reparatur *(mean time to repair)* steht [MIO87].

Durch eine Reduktion der MTTR wird also die Verfügbarkeit erhöht. Die Zeit zum Abschluss der Reparatur setzt sich zusammen aus der Zeit vom Eintreten des Ausfalls bis zu seiner Entdeckung (MTTFD, *mean time to failure detection*), der Zeit bis zur Lokalisierung der Ausfallursache (MTTFL, *mean time to fault localization*) und der Zeit zur Durchführung der Reparatur (MTFR, *mean time for repair*), d.h. es ergibt sich

$$\text{Verfügbarkeit} = \frac{\text{MTTF}}{\text{MTTF} + (\text{MTTFD} + \text{MTTFL} + \text{MTFR})}$$

Ein Software-Betriebs-Leitstand soll nun zunächst helfen, die Zeit zur Entdeckung eines Ausfalls zu reduzieren. Im Anschluss hilft er auch bei der Analyse der Ausfallursache und somit auch zur Reduktion der Zeit zur Durchführung der Reparatur. Des Weiteren kann

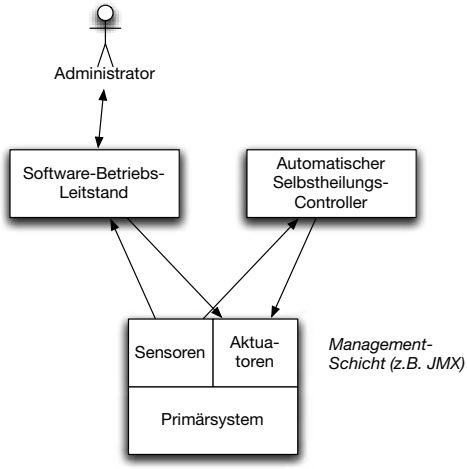

Abbildung 3: Dualität zwischen überwachendem/regelndem System und Primärsystem

ein Software-Betriebs-Leitstand auch bei der Vorhersage von Ausfällen helfen, indem das System auf ungewöhnliches Verhalten hin überwacht wird.

3.2 Allgemeine Anforderungen

Leitstand und Primärsystem In Abbildung 3 ist das Verhältnis zwischen Leitstand und Primärsystem dargestellt. Um einen Leitstand betreiben zu können, muss das Primärsystem mindestens Sensoren aufweisen, die Daten an den Leitstand liefern können. Die Sensoren können Daten über das Laufzeitverhalten (Quality-of-Service [HR06]) erfassen, oder auch eher statische Eigenschaften [FN00] der Komponenten oder ihres Entwicklungsprozesses abfragen und aggregieren. Soll ein erweiterter Leitstand mit Beeinflussungsfunktionalität betrieben werden, müssen weiterhin Aktuatoren vorhanden sein, die am Leitstand vorgenommene Änderungen im System wirksam werden lassen können. Sensoren und Aktuatoren werden in einem System mit einer sauberen Architektur von einer gesonderten Management-Schicht bereitgestellt. Im Falle von JavaEE-basierten Systemen steht hierfür beispielsweise das JMX-Framework [Per02] zur Verfügung. Die Sensoren und Aktuatoren der Management-Schicht können neben einem Software-Betriebs-Leitstand in ähnlicher Weise auch von einem Selbstheilungs-Controller [GCS03] genutzt werden. Der wesentliche Unterschied ist, dass ein Software-Betriebs-Leitstand vorrangig zur Bedienung durch einen menschlichen Administrator gedacht ist, während ein Selbstheilungs-Controller autonom arbeitet. Beide Ansätze können sich auch ergänzen: Der Selbstheilungs-Controller kann vorhergesehene und automatisiert behandelbare Störungen selbstständig regeln, während der Administrator mit Hilfe des Software-Betriebs-Leitstands die übrigen Störungen bewältigt.

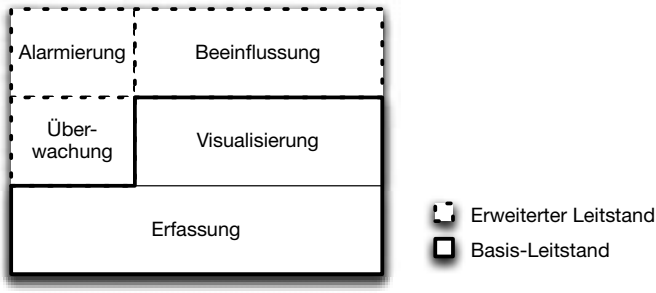

Abbildung 4: Schichtenmodell der Basis- und Erweiterungs-Funktionalität eines Software-Betriebs-Leitstands

Funktionalitätsgruppen In Abbildung 4 ist ein Schichtenmodell der Funktionalität eines Software-Betriebs-Leitstands dargestellt. Wir unterscheiden zunächst zwischen Basis- und Erweiterungs-Funktionalität: Die Basis-Funktionalität muss ein Sekundärsystem bereitstellen, damit von einem Software-Betriebs-Leitstand gesprochen werden kann, die Erweiterungs-Funktionalität ist optional. Zur Basis-Funktionalität gehören zwei Gruppen:

Erfassung Zunächst müssen die Daten von den Sensoren des Primärsystems erfasst und ggf. aufbereitet und aggregiert werden. Es soll auch ein Zugriff auf historische Daten ermöglicht werden.

Visualisierung Für die Visualisierung werden die an Messpunkten erfassten Betriebsdaten mit einem Architekturmodell des Primärsystems verknüpft (siehe Abbildung 5). Zur Reduktion des zum Betrieb des Leitstands nötigen Aufwands, werden Änderungen am Architekturmodell optimalerweise automatisch aus dem Primärsystem abgeleitet.

Zur Erweiterungs-Funktionalität gehören mindestens folgende Gruppen:

Beeinflussung Es werden Möglichkeiten zur Veränderung von Systemeigenschaften bereitgestellt. Wünschenswert ist hierbei eine Integration mit der Visualisierungsfunktionalität und dem Architekturmodell (direkte Manipulation [Shn87]). Zur Beeinflussung muss das Primärsystem auch Aktuatoren zur Verfügung stellen. Diese umfassen u.a. Reparatur- und Rekonfigurationsoperationen [MMHR04].

Überwachung Gemäß festgelegter Regeln können kritische Systemzustände automatisch erkannt werden. Solche Regel können auch automatisiert durch statistische Analysen des Langzeitverhaltens gebildet werden (z.B. [KF05]). Diese Erkennung von Ausfällen oder Anomalien unterstützt die oft kritischen Prozesse der Fehlerdiagnose und Reparatur.

Alarmierung Aufbauend auf den Überwachungsfunktionen können anwesende oder auch nicht anwesende Administratoren auf kritische Systemzustände aufmerksam gemacht werden. Eine Fern-Alarmierung per E-Mail, Telefon oder SMS ist denkbar.

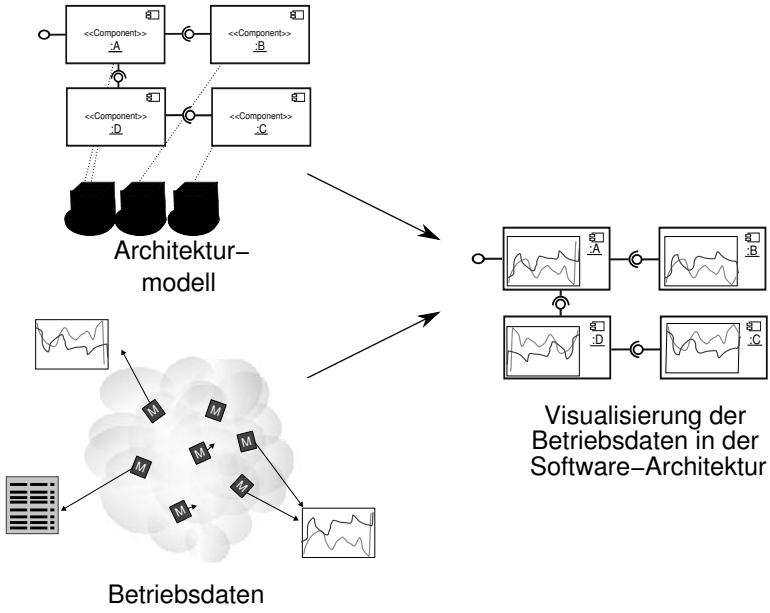

Abbildung 5: Verknüpfung von Architekturbeschreibung und Betriebsdaten

4 Überwachung und Beeinflussung von Informationssystemen

Eine Unternehmensanwendungslandschaft wird in der Praxis nie homogen ausschließlich aus Anwendungen und Komponenten bestehen, die die gleiche Implementierungstechnologie einsetzen, sei es JavaEE (Java Enterprise Edition [Sun05b], vormals J2EE) oder eine andere. Wir betrachten beispielhaft einen Leitstand für Unternehmen, die zumindest für ihre Eigenentwicklungen vorwiegend JavaEE einsetzen und somit die JavaEE-Anwendungen eine herausragende Bedeutung innerhalb ihrer Anwendungslandschaft einnehmen. Gleichwohl sollen auch JavaEE-fremde Komponenten berücksichtigt werden können um der Heterogenität realer Anwendungslandschaften Rechnung zu tragen. Es muss dabei in Kauf genommen werden, dass über Nicht-JavaEE-Komponenten nur eingeschränkte Informationen erfasst werden können (sensorische Restriktion) und/oder die Beeinflussungsmöglichkeiten reduziert sind (aktuatorische Restriktion).

Ein wesentlicher Vorteil von JavaEE ist hierbei, dass bereits standardisierte Schnittstellen wie JMX [Per02] zur Verfügung stehen. So können sich Anwendungsentwickler zunächst auf die Datenerfassung und Integration von Steuerungsfunktionalität in den einzelnen Anwendungskomponenten konzentrieren, während die JMX-Implementierung des Anwendungsservers im wesentlichen automatisch die Infrastruktur zum dezentralen Zugriff auf die Sensorik und Aktuatorik anbietet. Separate Managementschnittstellen von Komponenten und Ausführungscontainern (siehe Abbildung 6) ermöglichen eine Trennung von den Schnittstellen der Geschäftslogik. Somit wird die Wartbarkeit und Erweiterbarkeit der Anwendungsarchitektur nicht beeinträchtigt.

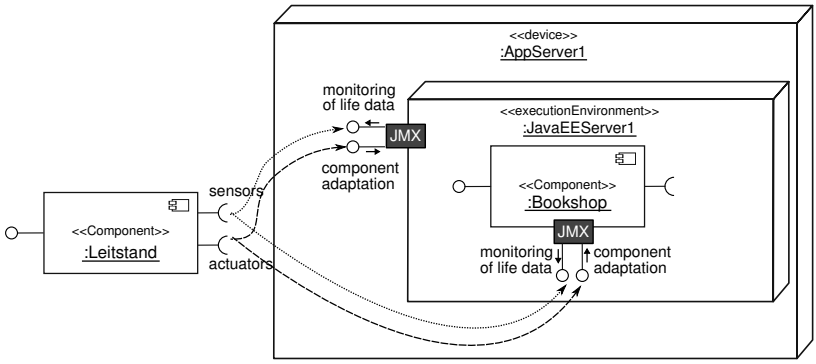

Abbildung 6: Management-Schnittstellen (hier Java Management Extensions, JMX) für die Steuerung und Erfassung des Software-Betriebs.

Unter Windows können .NET-basierte Anwendungen mit der Windows Management Instrumentation [LM02] ähnliche Schnittstellen nutzen. Auch für Web Services stehen standardisierte Management-Schnittstellenspezifikationen zur Verfügung [Sun05a].

5 Ausblick

Es soll ein Leitstand-Prototyp entwickelt werden, der aktuelles Systemverhalten (z.B. Antwortzeiten und Speicherverbrauch von Komponenten) mit einer Systemarchitekturbeschreibung (z.B. UML 2) in einem Modell verbindet. Die Architekturbeschreibung soll dabei automatisch zur Laufzeit aktualisiert werden, um das eingangs beschriebene Problem der veralteten Dokumentation zu vermeiden. Einzelne Komponenten des überwachten JavaEE-Systems können über Schnittstellen wie JMX Messdaten und Steuerungsmechanismen anbieten, welche ebenfalls automatisch in das Architekturmodell eingebunden werden. Der wesentliche Beitrag des Leitstand-Prototypen wird es sein, diesen fehlenden Architekturmodellbezug herzustellen und aufrecht zu halten. Eine Visualisierungskomponente wird mehrere Sichten auf dieses kombinierte Modell in einer grafischen Oberfläche anbieten. Damit werden die aktuelle Systemarchitektur, das Laufzeitverhalten und von Komponenten angebotene Steuerungsmöglichkeiten zusammengefasst.

Aufgrund der besonderen Eigenschaften von Software ist es nicht sinnvoll, Entwicklung und Betrieb von Individualsoftware scharf zu trennen: In der Praxis lässt sich keine fehlerfreie Software entwickeln, was zu besonderen Anforderungen an die Fehlererkennung führt, gleichzeitig aber auch das Potenzial zur flexiblen Fehlerbeseitigung birgt. Bekanntlich entfällt der Großteil der Software-Entwicklungskosten auch nicht auf die initiale Entwicklung, sondern auf spätere Wartungstätigkeiten. Wartung schließt aber auch die Aufrechterhaltung des ordnungsgemäßen Betriebs mit ein. Hierfür ist ein integrierter Leitstand hilfreich, der die Funktionalität eines Software-Entwicklungs-Leitstands und eines Software-Betriebs-Leitstands vereinigt und um weitere Funktionalität ergänzt.

Literatur

[FN00] Norman E. Fenton und Martin Neil. Software metrics: roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, Seiten 357–370. ACM Press, 2000.

[GCS03] David Garlan, Shang-Wen Cheng und Bradley Schmerl. Increasing System Dependability through Architecture-based Self-repair. In R. de Lemos, C. Gacek und A. Romanovsky, Hrsg., *Architecting Dependable Systems*, Jgg. 2677 of *Lecture Notes in Computer Science*, Seiten 23–46. Springer-Verlag, Berlin, 2003.

[HR06] W. Hasselbring und R. Reussner. Toward Trustworthy Software Systems. *IEEE Computer*, 39(4):98–99, April 2006.

[IEC98] IEC. *Fieldbus Technology – Systems Integration, Networking, and Engineering*, 1998. IEC Standard 60870-6 TASE.2.

[IEC05] IEC. *Communication Networks and Systems in Substations*, 2005. IEC Standard 61850.

[KF05] Emre Kiciman und Armando Fox. Detecting Application-Level Failures in Component-based Internet Services. *IEEE Transactions on Neural Networks*, 16(5):1027–1041, September 2005.

[LM02] Matthew Lavy und Ashley Meggitt. *Windows Management Instrumentation (WMI)*. Sams, 2002.

[Lyu96] Michael R. Lyu, Hrsg. *Software Reliability Engineering*. McGraw-Hill, New York, 1. Auflage, 1996.

[MIO87] John D. Musa, Anthony Iannino und Kazuhira Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, New York, 1. Auflage, 1987.

[MMHR04] Jasminka Matevska-Meyer, Wilhelm Hasselbring und Ralf H. Reussner. On A Software Architecture Description supporting Component Deployment and System Runtime Reconfiguration. In Jan Bosch, Clemens Szyperski und Wolfgang Weck, Hrsg., *Proceedings of the Ninth International Workshop on Component-Oriented Programming (WCOP'04)*, Juni 2004.

[Per02] J. Steven Perry. *Java Management Extensions*. O'Reilly, 2002.

[Shn87] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. In R. M. Baecker und W. A. S. Buxton, Hrsg., *Readings in Human-Computer Interaction: A Multidisciplinary Approach*, Seiten 461–467. Morgan Kaufmann, 1987.

[Sun05a] Sun. Web Services Specifications for Systems Management. Bericht, Sun Microsystems, 2005. `http://developers.sun.com/techtopics/webservices/management/`.

[Sun05b] Sun Microsystems. *Java Platform, Enterprise Edition 5 (Java EE 5) Specification*, 2005. JSR-000244, Proposed Final Draft.

[Wik06] Wikipedia. Leitstand — Wikipedia, The Free Encyclopedia, 2006. [Online; letzter Zugriff 28.7.2006].