Software Construction

RWTH AACHEN UNIVERSITY

Fakultät für Mathematik,
Informatik und
Naturwissenschaften

Lehr- und Forschungsgebiet
Informatik 3
Softwarekonstruktion

Bachelorarbeit

Entwicklung einer generischen Logging Infrastruktur für EMI-basierte Messsysteme

Development of a generic Logging Infrastructure for EMI-based measurement systems

vorgelegt von

Jan Simon Döring

aus Hamburg

am 5. Dezember 2014

Gutachter

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Betreuer

Dipl.-Inform. Matthias Vianden

Hiermit erkläre ich, Jan Simon Döring, an Eides statt, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keinem anderem Prüfungsamt vorgelegt und auch nicht veröffentlicht.

Aachen, 5. Dezember 2014 (Jan Simon Döring)

Danksagung

An dieser Stelle möchte ich mich zunächst bei Herrn Prof. Dr. rer. nat. Horst Lichter bedanken. Zum einen danke ich ihm für die Möglichkeit, diese Bachelorarbeit an seinem Lehrstuhl, dem Lehr- & Forschungsgebiet 3 Softwarekonstruktion, verfassen zu können. Zum anderen danke ich ihm und natürlich Herrn Prof. Dr. rer. nat. Bernhard Rumpe für das Gutachten bzw. Zweitgutachten dieser Arbeit.
Besonderer Dank gilt meinem Betreuer, Dipl.-Inform. Matthias Vianden, dessen herzliche Art, sowie die außerordentliche Betreuung und Ratschläge einen großen Gewinn bedeutete.
Auch möchte ich Dipl.-Inform. Andreas Steffens herzlichst für sein wertvolles Feedback und Anregungen danken!
Unabhängig von dieser Bachelorarbeit danke ich insbesondere meinen Eltern, dich mir während meines Studiums sowohl moralisch als auch finanziell immer zur Seite standen.
Abschließend möchte ich allen nicht namentlich Genannten danken, die in irgendeiner Art und Weise zu dieser Arbeit beigetragen haben. Vielen Dank!

Jan Simon Döring

Kurzdarstellung

Deutsch

Die vorliegende Arbeit beschäftigt sich thematisch mit dem Logging in verteilten Systemen, in der konkreten Ausprägung einer Messinfrastruktur bestehend aus Mikroservices (Enterprise Measurement Infrastructure - EMI). Innerhalb dieser Infrastruktur spielt Logging neben dem Monitoring der einzelnen Services eine zentrale Rolle im Betrieb. Zur Realisierung des verteilten Loggings innerhalb einer EMI wird eine Logging Infrastruktur entwickelt, die neben der Logintegration aus heterogenen Quellen auch eine generische Konfiguration der Logerzeuger ermöglicht. Die integrierten Logs, sowie die Konfigurationen werden zentral von einem, in dieser Arbeit entwickeltem Logging Service verwaltet und dem Anwender mithilfe einer Visualisierungs- & Konfigurationswerkzeug präsentiert, welches ebenfalls in dieser Arbeit entwickelt wurde.

English

This thesis deals with the challenge of logging in distributed systems. More precisly, an approach for logging in an Enterprise Measurement Infrastructure (EMI) consisting of microservices is presented. Since logging (in addition to service monitoring) is crucial for operations of a microservice-based infrastructure, a logging infrastructure is developed in this thesis. The logging infrastructure is capable of integrating logs from heterogenous sources and provides a generic configuration mechanism for configuring log-producing related variabilities. To store and manage the integrated logs and configurations, a Logging Service is developed, which provides data for a visualization and configuration tool that also is developed in this thesis.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Liste der Quelltexte

1 Einleitung

> The most effective debugging tool is still careful thought, coupled with judiciously placed print statements
>
> Brian Kernighan

Inhalt

"Hallo Welt", das klassische Einführungsbeispiel in eine Programmiersprache, verdeutlicht bereits, dass seit Beginn der Softwareentwicklung Entwickler & Anwender einen Mechanismus benötigen, zur Laufzeit einer Anwendung Ereignisse textuell zu protokollieren und diese über das Ausgabemedium nachzuvollziehen.
Mit steigender Komplexität der Softwaresystem steigt auch die Bedeutung dieses Protokollierungsmechanismus, im Folgenden als Logging bezeichnet, der als Grundlage für verschiedenste Aufgaben verwendet werden kann. Die Nützlichkeit eines protokollierten Ereignisses wird hierbei über die protokollierten Informationen und deren Weiterverwendungsgrad in der Analyse definiert. Da die Zuständigkeiten für die Definition eines zu protokollierenden Ereignisses und für die Analyse des protokollierten Ereignisses häufig auf unterschiedliche Rollen, nämlich den Entwickler und den Anwender, verteilt sind, kennt der Entwickler die Anforderungen an die zu protokollierenden Informationen nicht, sodass die Verwendungsmöglichkeiten des Loggings bezüglich der Analyse sehr variieren.
Unter der Annahme, dass die benötigten Informationen protokolliert werden, kann Logging für verschiedenste Verwaltungsaufgaben eines Softwaresystem wie die Anomalie Erkennung [FLWL09, LFY+10], Profiling [NKN12, SCH+11] und Reporting verwendet werden. Da Logging eine Möglichkeit bietet das Systemverhalten nachzuvollziehen [FLL+13] und zu protokollieren, kann Logging außerdem zur Fehlersuche & -Behandlung verwendet werden bzw. diese Vorgänge beschleunigen [GKG+09]. Insgesamt wird die Wichtigkeit des Loggings für die Systementwicklung & -wartung nach einer Studie von Qiang Fu et al. [FZH+14] von 96% der befragten Teilnehmer als sehr hoch eingestuft.
Daher ist es entscheidend die Nützlichkeit des Loggings für Analysezwecke auch in modernen Softwaresystem zu gewährleisten, die häufig keine monolithischen Systeme mehr darstellen, sondern als verteilte Systeme konzeptioniert werden. Voraussetzung für die Nützlichkeit des Loggings stellt neben den protokollierten Daten, der einfache Zugriff auf alle protokollierten Ereignisse dar. Während in monolithischen Systemen diese Voraussetzung meistens implizit erfüllt ist, ist die Integration aller protokollierten

Ereignisse in verteilten Systemen eine Herausforderung.
Neben dieser Integrationsproblematik führen die vielseitigen Verwendungsmöglichkeiten des Logging zu einer weiteren, Stakeholder spezifischen Herausforderung. Während die Betreiber eines Softwaresystems aus Performancegründen beispielsweise nur an der Protokollierung von wichtigen Ereignissen interessiert sind, benötigen Entwickler zur Fehlersuche möglichst viele Informationen aus allen protokollierbaren Ereignissen. Die Granularität des Logging muss sich also zur Laufzeit konfigurieren lassen, was durch die Synchronisationsproblematik von Konfigurationen in einem verteilten System ebenfalls eine Herausforderung darstellt.
Die am Lehr- und Forschungsgebiet Informatik 3 der RWTH Aachen entwickelte Enterprise Measurement Infrastructure (EMI) -Referenzarchitektur ist genau mit diesen Herausforderungen konfrontiert. Grob beschrieben, stellt eine EMI ein verteiltes System aus dedizierten Mikroservices zur Datenintegration, Metrik-Berechnung und Visualisierung dar. Da die EMI-Referenzarchitektur die Schwierigkeiten des verteilten Logging noch nicht addressiert, soll in dieser Arbeit ein Mechanismus für das verteilte Logging innerhalb einer EMI entwickelt werden.

1.1 Ziele der Arbeit

In dieser Arbeit soll eine Lösung für das verteilte Logging innerhalb einer EMI entwickelt werden. Diese Lösung soll nicht nur alle protokollierten Ereignisse der einzelnen Services zentral verfügbar machen, sondern auch einen Mechanismus zur zentralen Konfiguration des Protokollierungsvorgangs der einzelnen Services anbieten. Hierfür wird eine Logging Infrastruktur entworfen, die zum einen Mechanismen für die Ereignis-Protokollierung anbietet, sowie für die Ereignis-Sammlung und -Speicherung zuständig ist und zum anderen die Logging Konfiguration der Services ermöglicht. Insgesamt soll diese Logging Infrastruktur eine flexible Logging Plattform bieten, die als Grundlage für verschiedenste, auf einer Analyse der protokollierten Daten basierenden Aufgaben dient.
Logging Strategien als große Herausforderung des Loggings, also was und zu welchem Ereignis protokolliert werden soll, liegen dabei außerhalb des Fokus dieser Arbeit. Eine ausführliche Darstellung der Logging Strategien ist in den Arbeiten von Qiang Fu et al. [FZH+14] und Anton Chuvakin et al. [CP10] zu finden.
Primär soll die Logging Infrastruktur sicherstellen, dass Ereignisse variabel protokolliert werden können und zentral gesammelt werden.
Zur Darstellung der Logging Infrastruktur führt Kapitel 2 zunächst die benötigten Grundlagen, sowie die Systemumgebung ein. Dem klassischen Softwareentwicklungsprozess folgend, definiert Kapitel 3 die Anforderungen an die zu entwickelnde Lösung. Diese Anforderungen werden in Kapitel 4 verwandten Arbeiten gegenübergestellt, um die Notwendigkeit einer EMI spezifischen Neu- bzw. Weiterentwicklung zu analysieren. Ausgehend von diesem Wissen wird die Logging Infrastruktur in Kapitel 5 entworfen und die einzelnen Komponenten in den darauffolgenden Kapiteln detaillierter beschrieben. Anschließend evaluiert Kapitel 9 die

vorgestellte Lösung und Kapitel 10 liefert schließlich eine Zusammenfassung und bietet einen Ausblick.

2 Grundlagen

Eine gute Theorie ist das
Praktischste, was es gibt

Gustav Robert Kirchhoff

Inhalt

2.1 Zentrale Begriffe

Um Missverständnissen vorzubeugen und klare Definitionen zu schaffen, werden in diesem Abschnitt die zentralen Begriffe zum Thema Logging eingeführt.

2.1.1 Log

Basierend auf den Definitionen des National Institute of Standars & Technology (NIST) [KS06] und A. Chuvakin [CSP13] werden in dieser Arbeit Log und Logeintrag folgendermaßen definiert.

Definition 1. *Ein Log ist eine Sammlung von Logeinträgen.*

Definition 2. *Ein Logeintrag besteht aus textuellen Informationen zu einem spezifischen Ereignis, das innerhalb eines Systems aufgetreten ist. Die Informationen umfassen typischerweise einen Zeitstempel und weitere ereignisspezifische Daten. Als Artifakt eines Softwaresystemes besitzt ein Logeintrag einen Lebenszyklus der typischerweise aus 5 Phasen (Erzeugung, Übermittlung, Speicherung, Analyse und Bereinigung) besteht.*

In der Praxis werden der Begriff des Logs und des Logeintrages häufig synonym verwendet. Um Unklarheiten zu vermeiden, werden in dieser Arbeit beide Begriffe entsprechend ihrer Definition verwendet. Das NIST [KS06] unterscheidet drei verschiedene Kategorieren, denen sich Logs und damit deren Logeinträge zuordnen lassen.

Sicherheitssoftwareslogs Logeinträge eines Sicherheitssoftwarelog umfassen hauptsächlich computersicherheitsrelevante Informationen, die beispielsweise von Antivirus-Software erzeugt wurden.

Betriebsystemlogs Logeinträge eines Betriebsystemlogs enthalten verschiedenste Informationen. Hauptfaktor der Kategorisierung ist, dass das korrespondierende Ereignis des Logeintrags ein System Ereignis war, also innerhalb einer Betriebssystemkomponenten aufgetreten ist, beispielsweise der Start eines Services.

Anwendungslogs Wie Betriebsystemlog-Einträge werden Anwendungslog-Einträge über das korrespondierende Ereignis charakterisiert. Ist der Ereignisauslöser eine Anwendung, wird der entsprechende Logeintrag als Anwendungslog-Eintrag bezeichnet.

Da diese Arbeit wie in Abschnitt 2.3 beschrieben, im Umfeld einer Enterprise Measurement Infrastructure (EMI) stattfindet, wird im Folgenden nicht mehr zwischen den unterschiedlichen Log-Kategorieren unterschieden und ein Log-Eintrag implizit als Anwendungslog kategorisiert, da jeder Logeintrag, der im Rahmen einer EMI auftritt, zur Kategorie der Anwendungslogs gehört.

2.1.2 Logging

Angelehnt an die Logging Definition von [CSP13], wird Logging folgendermaßen definiert.

Definition 3. *Logging ist der Vorgang der Log-Eintrag Erzeugung und Ausgabe in ein Medium, häufig einer Datei, zur Sammlung der Ereignisse in einem Log.*

Grundsätzlich beinhaltet Logging zwei verschiedene Akteure.

Logging Sender Der Logging Sender ist für die Erzeugung und temporäre Verwaltung des Log-Eintrags zuständig. Entweder verschickt er den Log-Eintrag direkt an den Logging Empfänger (**push-basiertes Logging**), oder er verschickt den Log-Eintrag erst auf Anfrage des Logging Empfängers (**pull-basiertes Logging**).

Logging Empfänger Der Logging Empfänger ist zuständig für den Empfang der Log-Einträge vom Logging Senders und die anschließende Ausgabe zwecks Persistierung in ein Medium.

Zur Verbesserung der Nützlichkeit des Loggings bieten viele Logging Mechanismen in Form einer vollständigen, partiellen Ordnung (Log-Level) die Möglichkeit einen Log-Eintrag in seiner Wichtigkeit zu kategorisieren.

2.1.3 Logger

Der Begriff des Loggers wird in dieser Arbeit synonym zum Begriff des Software Loggers verwendet und wird daher folgermaßen definiert.

Definition 4. *Ein Logger stellt die Schnittstelle zum Ausführen eines Logging Vorganges dar. Dabei übernimmt ein Logger häufig sowohl die Rolle des Logging Senders als auch des Logging Empfängers.*

Abhängig vom verwendeten Logging Mechanismus lässt sich aus Granularitäts- & Performancegründen pro Logger ein Schwellenwert definieren, sodass nur Logeinträge, deren Wichtigkeit über dem Schwellenwert liegt, versendet werden.

2.2 Logging Infrastruktur

Eine Logging Infrastruktur bietet grundlegende Möglichkeiten zur Abbildung des Lebenszyklus eines Log-Eintrages. Ausgehend von der Definition des NIST [KS06] und des Lebenszyklus eines Log-Eintrages wird eine Logging Infrastruktur folgendermaßen definiert.

Definition 5. *Eine Logging Infrastruktur stellt Mechanismen zur Erzeugung, Sammlung und Speicherung von Log-Einträgen zur Verfügung und bietet Schnittstellen zur Verwaltung und anschließenden Analyse der gespeicherten Log Einträge.*

Optional ermöglicht die Logging Infrastruktur die Konfiguration von Log-Eintrag erzeugungsspezifischen Parametern, wie z.B. das Log Level.
Unterstützt die Logging Infrastruktur erweiterte Funktionalitäten hinsichtlich der Speicherung von Log-Einträgen (z.B. Log Normalisierung, Log Rotation) oder bietet Analysemöglichkeiten (z.B. Event Korrelation) spricht man von einer **Log Management Infrastruktur**. [KS06]

2.3 EMI

Da die Logging Infrastruktur in eine Enterprise Measurement Infrastruktur (EMI) integriert werden soll, erfolgt in diesem Abschnitt eine Darstellung der EMI-Referenzarchitektur, sowie der bisherigen Logging Situation innerhalb konkreter EMI - Instanzen, um die Anforderungen an die Logging Infrastruktur motivieren und definieren zu können.

2.3.1 EMI - Referenzarchitektur

Aufgrund verschiedener Einschränkungen existierender Messsysteme [LVS14], wurde am Lehr- und Forschungsgebiet Informatik 3 der RWTH Aachen eine Referenzarchitektur für Messsinfrastrukturen (EMIs) entwickelt, die diese Einschränkungen durch die Verwendung dedizierter Mikroservices adressiert. Insgesamt soll eine EMI Instanz die Vermessung externer, heterogener Datenquellen ermöglichen und in Form von Metriken visualisieren.
Die EMI-Referenzarchitektur (siehe Abbildung 2.1) stellt eine 5-Schichten Architektur dar. Die unterste Schicht der Datenanbieter (Data Provider) repräsentiert die externen Datenquellen und ist daher kein fester Bestandteil der Messinfrastruktur, wird aber zur Bereitstellung der Daten benötigt und muss damit in jeder Messinfrastruktur vorhanden sein. Die bereitgestellten Daten der Datenanbieter werden mithilfe der darüberliegenden Schicht der Datenadapter (Data Adapter) in die konkrete Messinfrastruktur importiert

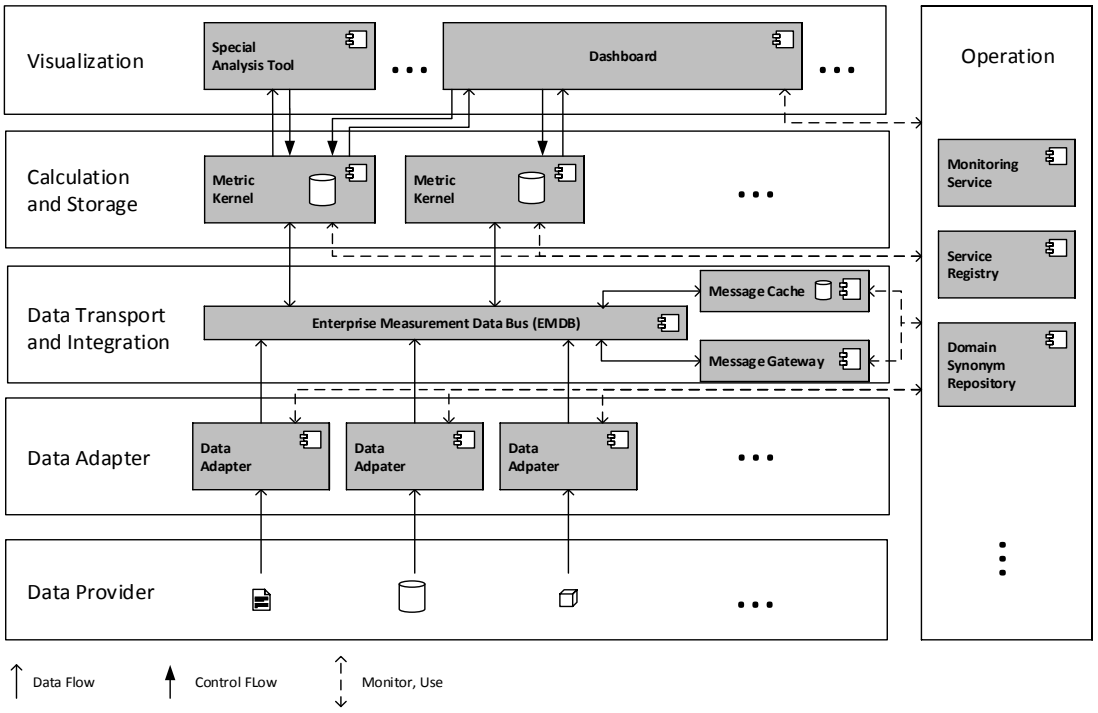

Abbildung 2.1: EMI Referenzarchitektur (aus [LVS14])

und verarbeitet, also normalisiert und in interne Datentypen konvertiert. Die Datenadaptierung erfolgt dabei entweder push, pull oder invoke-push basiert [LVS14], d.h. entweder stellt der Datenanbieter dem Datenadapter seine Daten zu einem selbstbestimmten Zeitpunkt zur Verfügung, der Datenadapter fragt in bestimmten Zeitabständen die Daten vom Datenanbieter ab oder der Datenanbieter informiert den Datenadapter zu einem selbstbestimmten Zeitpunkt, dass der Datenadapter Daten abfragen soll.
Nach erfolgreicher Datenadaptierung werden diese Daten anhand des Publish-Subscribe Enterprise Integration Pattern [HW03] über die Transport- & Integrationsschicht (Data Transport and Integration) publiziert. Dabei erfolgt die Kommunikation über einen Kommunikationsbus, den Enterprise Measurement Data Bus (EMDB). Komponenten der Berechnungs- & Persistierungsschicht (Calculation and Storage), die Metrikkernel (Metric Kernel), führen auf den publizierten Daten (Metrik)-berechnungen durch, persistieren diese Resultate und stellen sie den Komponenten der obersten Schicht, der Visualisierungsschicht (Visualization) zur Verfügung, welche sie anschließend beispielsweise in Form von Diagrammen dem Anwender präsentieren.
Zusätzlich verläuft vertikal zu den vorgestellten Schichten die Betriebsschicht (Operation), welche zentrale Services einer Messinfrastruktur wie das Service Monitoring bereitstellt.

2.3.2 Bisherige Loggingsituation innerhalb der EMI-Instanzen

Um die Notwendigkeit einer Logging Infrastruktur im Rahmen einer EMI zu motivieren, stellt dieser Abschnitt die bisherige Loggingsituation in konkreten EMI-Instanzen dar.
Aktuell protokolliert jeder Mikroservice aufgetretene Ereignisse in die Log-Datei seines Anwendungsservers. Die verschiedenen Logs der Anwendungsserver werden hierbei nicht integriert, sodass für eine Analyse der Log-Einträge eine manuelle Integration der einzelnen Logs nötig wäre. Auch erhält der Anwender keine zusätzliche Unterstützung in Form eines komfortablen Verwaltungs- & Visualisierungswerkzeuges. Stattdessen ist er auf Betriebssystem-Werkzeuge angewiesen.
Auch für die Konfiguration der einzelnen Logger existiert außer dem Loggingbiblithek-spezifischen Mechanismus (häufig über Konfigurationsdateien) keine weitere Konfigurationsmöglichkeit. Da die Entwicklung der einzelnen Komponenten einer EMI von unterschiedlichen Entwicklern durchgeführt wurde, werden außerdem verschiedene Logging Bibliotheken eingesetzt, sodass eine Konfiguration sowohl lokal am Service, als auch unter Umständen mit vorheriger Recherche zur Konfiguration der entsprechenden Bibliothek durchgeführt werden muss.
Insgesamt ist eine praktikabel Verwendung der Logging Resultate in der jetzigen Form nicht möglich, sodass ein Mechanismus zum verteilten Logging, sowie der Loggerkonfiguration notwendig ist.

3 Anforderungen

> The most difficult part of requirements gathering is not the act of recording what the user wants, it is the exploratory development activity of helping users figure out what they want.
>
> Steve McConnell

Inhalt

Wie von Ludewig und Lichter definiert, beginnt der Software-Entwicklungsprozess mit der Analyse *was benötigt wird*[LL10]. Um eine zielorientierte Entwicklung sicherzustellen, müssen daher zunächst die Bedürfnisse der zukünftigen Benutzer (Stakeholder) des Systemes identifiziert werden. Ausgehend von diesen Bedürfnissen, lassen sich anschließend die Anforderungen spezifizieren, was dem zweiten Schritt des Software-Entwicklungsprozesses entspricht.

Diese Arbeit folgt dem angesprochenen Software-Entwicklungsprozess, sodass nachfolgend zunächst die Stakeholder identifiziert (siehe Abschnitt 3.1) und ihre Bedürfnisse analysiert werden (siehe Abschnitt 3.2). Insgesamt können in Abschnitt 3.3 dann die funktionalen Anforderungen und in Abschnitt 3.4 die nicht-funktionalen Anforderungen präzise definiert werden.

3.1 Stakeholder

Bereits der Titel dieser Bachelorarbeit lässt auf das Systemumfeld schließen. Die zu entwickelnde Logging Infrastruktur wird im Umfeld einer EMI eingesetzt. Aus diesem Grund beschränken sich die möglichen Stakeholder auf die Stakeholder einer EMI, die in der Arbeit von Vianden et al. [LVS14] definiert und detailliert beschrieben sind. Die für die Logging Infrastruktur relevanten EMI Stakeholder werden im Folgenden kurz beschrieben.

Messkunden stellen die Benutzer der Mess-Infrastruktur dar. Als typisches Beispiel lässt sich der Projektmanager aufführen, der sich einen Überblick über sein vermessenes Projekt verschaffen will. Messkunden interagieren lediglich mit dem Metrikvisualisierungsfrontend und sind daher zwecks Urteilsbildung auf aktuelle und korrekte Datenbestände der Metriken angewiesen. Sollten die dargestellten Informationen nicht aktuell sein (was beispielsweise nach einem fehlgeschlagenem Datenimport der Fall ist), möchten sie über diese Tatsache informiert werden und die Möglichkeit haben, die Ursache dieses Missstandes herauszufinden, um geeignete Maßnahmen treffen zu können.

Metrik Experten unterstützen und beraten Metrikkunden hinsichtlich der Metrikauswahl - & Konfiguration ihres Projekts. Des Weiteren sind sie zuständig für die Wartung vorhandener Metriken und Konzeption neuer Metriken. Tritt während der Metrikkonfiguration ein Fehler auf, möchten sie Einblick in die Fehlermeldung erhalten können.

Entwickler einer EMI sind Softwareentwickler, die für die Realisierung der gestellten Anforderungen zuständig sind. Sie möchten während der Entwicklung vor allem zu Debuggingzwecken möglichst schnell und übersichtlich das Laufzeitverhalten der einzelnen EMI Services nachvollziehen können.

Betreiber einer EMI sind zuständig für die korrekte Funktionsweise und die Verfügbarkeit der einzelnen Services innerhalb der EMI. Sollte ein Service ausfallen, möchten sie die Ursache herausfinden können. Der Mechanismus zur Befriedigung dieses Bedürfnisses soll dabei in ihrer EMI ohne großen Aufwand verwendbar und wartungsarm sein, sowie keine Komplikationen mit anderen Services erzeugen.

3.2 Bedürfnisse

Um in Abschnitt 3.3 und Abschnitt 3.4 die Anforderungen spezifizieren zu können, analysiert dieser Abschnitt die Bedürfnisse der identifizierten Stakeholder. Die Bedürfnisse wurden dabei durch Gespräche mit den entsprechenden Stakeholdern gewonnen bzw. teilweise direkt aus der Stakeholderbeschreibung abgeleitet. Insgesamt lassen sich die Bedürfnisse drei verschiedenen Bereichen zuordnen:

B1 - Ereignisprotokollierung Entwickler möchten innerhalb einer EMI einen Ereignisprotokollierungsmechanismus, der

- **B1.1 - bekannt** Entwickler möchten ihnen bekannte Mechanismen zur Erzeugung eines Ereignisses verwenden.
- **B1.2 - kompatibel** Entwickler möchten bestehende Implementierungen so wenig wie möglich anpassen, um den Mechanismus zu verwenden
- **B1.3 - dynamisch** Entwickler möchten die Protokollierung konfigurieren können (z.B. während des Debugging fein granulare Ereignisprotokollierung, zur Laufzeit nur Protokollierung von kritischen Ereignissen).

ist.

B2 - Ereignisvisualisierung Metrikkunden, Metrikexperten, Entwickler und Betreiber möchten protokollierte Ereignisse

- **B2.1 - zentral** Die aufgetretenen aller EMI Services sollen an einer Stelle präsentiert werden
- **B2.2 - kontextsensitiv** Da ihre Informationsbedürfnisse variieren (projektspezifisch bzw. fachlich vs. technisch), möchten die Stakeholder abhängig von ihrem Informationsbedürfnis unterschiedliche Sichten auf die Ereignisse haben.
- **B2.3 - schnell** Die Stakeholder möchten ihre Informationsbedürfnisse schnell befriedigen können

betrachten und verwalten können.

B3 - Betrieb Betreiber möchten einen neuen Service (z.b. eine Logging Infrastruktur)

- **B3.1 - einfach** Der neue Service soll sich einfach ineiner bestehenden EMI verwenden lassen
- **B3.2 - wartungsarm** Die benötigte Zeit zur Wartung des Service soll minimal sein
- **B3.3 - störungsfrei** Die Betreiber erwarten, dass der Betrieb des Service die bestehenden Services nicht beeinträchtigt.

innerhalb ihrer EMI betreiben können.

3.3 Funktionale Anforderungen

Ausgehend von den definierten Aufgaben einer Logging-Infrastruktur (Abschnitt 2.2) und den Informationsbedürfnissen der Stakeholder werden nachfolgend die funktionalen Anforderungen abgeleitet um zusammen mit den nicht-funktionalen Anforderungen (siehe Abschnitt 3.4) eine vollständige Anforderungsspezifikation zu erhalten, die dem Software-Entwicklungprozess folgend in Kapitel 5 zum Entwurf der Logging Infrastruktur verwendet werden können.

FA1 - verteiltes Logging Abgeleitet aus den Bedürfnissen B1 und B2.1 muss die Log Erzeugung und Sammlung aller erzeugten Logs innerhalb einer EMI ermöglicht werden. Die Logging Infrastruktur muss daher Mechanismen sowohl zur Erzeugung als auch zur Sammlung der Log-Einträge bereitstellen bzw. bestehende Erzeugungsmechanismen unterstützen. In jedem Fall muss jedoch ein Mechanismus bereitgestellt werden, der die Log-Einträge aller angeschlossenen Services integriert und persistiert.

FA2 - Integrierbarkeit Bedürfnis B1.2, sowie B3.1 und B3.2 setzen voraus, dass die Logging-Infrastruktur sich innerhalb der konkreten EMI verwenden lässt und mit bestehenden Services z.B. dem Monitoring-Service interagieren kann.

FA4 - Kompatibilität Als Konsequenz aus FA2 - Integrierbarkeit und aufgrund B1.1, soll die Logging Infrastruktur die Log-Eintrag-Erzeugung mithilfe verschiedenen Logging Bibliotheken ermöglichen bzw. erzeugte Log-Einträge verschiedener Logging Bibliotheken integrieren können.

FA5 - Konfiguration zur Laufzeit Zur gleichzeitigen Befriedigung der Bedürfnisse B1.3 und B3.2, müssen die Logger der angeschlossenen Services zur Laufzeit konfigurierbar sein.Insbesondere muss die Konfiguration persistent sein, d.h. auch bei einem Neudeployment des entsprechenden Services gelten. Da nach FA4 - Kompatibilität unterschiedliche Logging Bibliotheken unterstützt werden müssen, müssen unterschiedliche Konfigurationen abgebildet werden können.

FA6 - Kategorisierung der Log Einträge Wegen B2.2 müssen sich einzelne Log-Einträge unterschiedlich kategorisieren lassen. Eine Unterscheidung muss mindestens in **fachliche Logs** und **technische Logs** erfolgen können.

FA7 - Visualisierungs- & Konfigurationswerkzeug Zur Befriedigung von B2 und B1.3 muss ein entsprechendes Werkzeug angeboten werden, um alle Logs anzuzeigen und eine Konfiguration der Logger zu ermöglichen. Insbesondere muss dabei eine fachliche und eine technische Sicht auf die Logs angeboten werden, um B2.2 zu erfüllen.

3.4 Nicht-funktionale Anforderungen

Nachdem die funktionalen Anforderungen dargestellt wurden, werden die nicht-funktionalen Anforderungen ebenfalls aus den Bedürfnissen der Stakeholder abgeleitet. Da die Notwendigkeit eines Visualisierungs- und Konfigurationswerkzeuges (FA7 - Visualisierungs- & Konfigurationswerkzeugidentifiziert wurde, werden die nicht-funktionalen Anforderungen an die Infrastruktur und das Werkzeug getrennt beschrieben.
Insgesamt sind mit diesem Abschnitt die Anforderungen an die Logging Infrastruktur vollständig spezifiziert.

3.4.1 Infrastruktur

Es wurden folgende nicht-funktionalen Anforderungen an die Logging Infrastruktur identifiziert:

NFA-I-1 - Stabilität Wegen B3.3 darf ein Ausfall der Logging Infrastruktur die Funktionalität der restlichen EMI Services nicht beeinflussen.

NFA-I-2 - Robustheit Ein Ausfall eines Loggers eines EMI Services oder eines kompletten EMI Services darf nicht zu einem Ausfall der Logging Infrastruktur führen um B3.2 zu befriedigen.

NFA-I-4 - Erweiterbarkeit Um FA2 - Integrierbarkeit sowie FA4 - Kompatibilität zu erfüllen, muss sich die Logging-Infrastruktur sowohl funktional, als auch hinsichtlich der unterstützten Logging-Bibliotheken erweitern lassen.

3.4.2 Visualisierungs- und Konfigurationswerkzeug

Abgeleitet aus B2.3, ergeben sich folgende nicht-funktionale Anforderungen an das Visualisierungs- und Konfigurationswerkzeug.

NFA-T-1 - Usability (Bedienbarkeit) Angelehnt an die Usability Definition von Ludewig und Lichter [LL10], soll das bereitgestellte Werkzeug alle Funktionen zur effizienten Aufgabenerfüllung bieten, konsistent und zur schnellen Informationsbefriedigung übersichtlich sein. Die Beurteilung der Usability Anforderungserfüllung erfolgt mithilfe der DIN EN ISO 9241 - 110 Norm [ISO06].

NFA-T-2 - Aktualität Die dargestellten Informationen sollen immer aktuell sein. Tritt während der Betrachtung der Log Einträge ein Log Event auf, soll der entsprechende Log Eintrag direkt angezeigt werden. Ebenfalls soll eine Konfigurationsänderung direkt dargestellt werden.

4 Verwandte Arbeiten & Werkzeuge

> To design the future effectively, you must first let go of your past.
>
> Charles J. Givens

Inhalt

Dem Software-Entwicklungsprozess folgend, wurde im vorangegangenem Kapitel eine Anforderungsanalyse durchgeführt, deren Resultat eine Spezifikation der Anforderungen war. Der nächste Schritt im Software-Entwicklungsprozess ist der Grobentwurf, d.h. die Strukturen der Lösung werden festgelegt. Da, um Anthony J. D'Angelo zitieren, das Rad nicht neu erfunden, sondern lediglich neu ausgerichtet werden soll, untersucht dieses Kapitel die Notwendigkeit einer Neuentwicklung. Sollte bereits ähnliche Softwarelösungen existieren, die die spezifizierten Anforderungen (teilweise) erfüllt, so ist unter Umständen keine Neuentwicklung notwendig oder sinnvolle Konzepte könnten übernommen werden. Aus diesem Grund werden zunächst einige verwandte Arbeiten bzw. Softwarelösungen vorgestellt, die anschließend den spezifizierten Anforderungen gegenüber gestellt werden.

4.1 ElasticSearch ELK Stack

Der ElasticSearch ELK Stack lässt sich als ein Echzeit-Analyse Werkzeug von Zeitstempel-basierten Daten darstellen. Er kombiniert **E**lasticSearch [1] zur Datenanalyse, **L**ogStash [2] zur Integration von Logdaten und **K**ibana [3] zur Visualisierung. ElasticSearch, LogStash und Kibana als OpenSource Projekte von Elasticsearch Inc entwickelt, werden im Folgenden vorgestellt.

[1] http://www.elasticsearch.org/overview/elasticsearch
[2] http://www.elasticsearch.org/overview/logstash/
[3] http://www.elasticsearch.org/overview/kibana/

4.1.1 ElasticSearch

ElasticSearch[1] ist eine auf Apache Lucene [4] basierende OpenSource Such Engine. Um Lucenes Komplexität und Bibliothekscharakter transparent zu halten, stellt ElasticSearch im Kern Lucenes Funktionalität, grob dargestellt eine mächtige Volltext-Suche, über eine RESTful API zur Verfügung. Auf diesem Kern aufbauend, bietet ElasticSearch eine verteilte dokumentenoriente Echzeitdatenbank, die jedes Feld indiziert und durchsuchbar macht, sowie eine verteilte Suchengine mit Echtzeitanalyse-Fähigkeiten an.
Der Verteilungaspekt wird transparent behandelt, sodass der Anwender nicht erfährt, ob er mit einem, oder mit mehren Knoten interagiert. Insgesamt lässt sich ElasticSearch also als skalierbare OpenSource Such- und Analyseengine beschreiben.

4.1.2 LogStash

LogStash[2] ist ein OpenSource Werkzeug zum Empfang, sowie zur Verarbeitung und Ausgabe von Logs bzw. Log-Einträgen. Dabei ist LogStash sehr modular aufgebaut und folgt dem von Ludewig und Lichter beschriebenen Pipe-Filter Architekturmuster [LL10], wobei jeder Filter als ein Modul bezeichnet wird. Die Module teilen sich in vier Bereiche auf.

Empfangsmodul Das Empfangsmodul ist für den Empfang der Log-Einträge aus verschiedenen Quellen zuständig. Die Bandbreite der Empfangsmodule reicht dabei von betriebsystemspezifischen Quellen (Dateien, Pipe, Syslog) über netzwerkspezische Quellen (Websockets, Amazon S3, TCP, UDP, XMPP) zu anwendungsspezifischen Quellen (Elasticsearch, Sqlite, Log4J). Aktuell gibt es 41 verschiedene Empfangsmodule. [Ela14]

Kodierungsmodul : Das Kodierungsmodul ist sowohl für die Dekodierung der empfangenen Daten, sowie für die Kodierung der auszugebenen Daten zuständig. Momentan existieren 20 verschiedene Kodierungsmodule. [Ela14]

Filtermodul Das Filtermodul ist verantwortlich für das Normalisieren der dekodierten Daten. Mithilfe geeigneter Filterdefinition können die Daten in eine eigene Datenstruktur transformiert werden. Ein typisches Anwendungsszenario eines Filtermoduls ist die Normalisierung des Zeitstempels, da in Daten aus heterogenen Quellen die Zeitrepräsentation unterschiedlich ist. Aktuell exisitieren 50 verschiedene Filtermodule. [Ela14]

Ausgabemodul : Nach der Normalisierung der Daten, werden diese über das Ausgabemodul entweder zu Persistierungzwecken oder zur Weiterverarbeitung wie beispielsweise der Analyse ausgegeben.Da es momentan 55 verschiedene Ausgabemodule gibt, ist die Auswahl an Verarbeitungsmöglichkeiten entsprechend groß.

[4] https://lucene.apache.org/core/

Zusammenfassend liegt der Fokus von LogStash auf der Integration & Aggregation heterogener Logeinträge.

4.1.3 Kibana

Kibana[3] ist ein webbasiertes Analyse und Such Werkzeug für verschiedene Log Aggregatoren. Hauptsächlich wird Kibana in Kombination mit ElasticSearch aufgrund dessen umfangreicher Such- & Analysemöglichkeiten eingesetzt. Kibana visualisiert für unerfahrene Awender die Daten in Form eines Dashboards und bietet gleichzeitig mächtige Analyse und Suchmöglichkeiten für erfahrene Anwender.

4.2 Blitz4J

Blitz4J [5] ist ein auf Log4J basierendes Logging Framework, welches mit dem Fokus auf Performance und Skalierbarkeit bei Netflix entwickelt wurde. Bestimmte Aspekte der Log4J Architektur wurden überarbeitet und durch nebenläufige Datenstrukturen ersetzt um die Deadlockgefährdung hinsichtlich verteiltem Logging zu vermeiden, sowie die allgemeine Performance zu verbessern. [kvG14] Konzeptionell dargestellt werden die Log4J Logger mit einem asynchronen Appender konfiguriert, der die Aufgabe des Versandes der Log Nachrichten übernimmt um verteiltes Logging zu ermöglichen. Über den mitgelieferten Appender bietet Blitz4J eine auf das LogLevel beschränkte Konfigurationsmöglichkeit zur Laufzeit.

4.3 Splunk

Splunk [6] ist eine umfangreiche kommerzielle Softwarelösung zu Log-, Monitoring und Reportingzwecken. Da auch eine kostenfreie Lizenz verfügbar ist, wird Splunk im Folgenden näher beschrieben. Den Kern von Splunk bildet ein performanter, skalierbarer Server, der Log Einträge aus verschieden Quellen sammelt, indiziert und durchsucht. Dabei erfolgt keine Normalisierung der Daten, stattdessen ist Splunk für einen Betrieb mit heterogenen Daten ausgelegt. Nach der Integration der Daten bietet Splunk dem Anwender umfangreiche Analysemöglichkeiten sowohl historischer Natur, als auch zur Optimierungszwecken (Geschäftsprozesse, Infrastruktur). Jede Splunk Instanz übernimmt dabei eine oder mehrere Rolle(n).

1. Deployment Server: Der Deployment Server ist zuständig für das lokale oder verteilte Management der einzelnen Splunk Instanzen.

2. Vermittler: Der Vermittler sammelt die Daten aus verschiedenen Quellen und leitet sie an einen Indizierer weiter.

[5] https://github.com/Netflix/blitz4j
[6] http://www.splunk.com/

3. Indizierer: Der Indizierer speichert die empfangenen, heterogenen Daten, indiziert diese und ist stellt einen Suchservice zur Verfügung.

4. Suchknoten: Eine konkrete Suche wird vom Suchknoten an die einzelnen Indizierer verteilt, sodass sogar eine Datencenter übergreifende Suche ermöglicht wird. Des Weiteren ist der Suchknoten zuständig für das Reporting.

Insgesamt lässt sich Splunk damit als Enterprise Alternative zum ELK Stack darstellen.

4.4 Anforderungsüberdeckung

In diesem Abschnitt wird die funktionale Anforderungsüberdeckung der vorgestellten Werkzeuge analysiert um zu evaluieren, ob es bereits eine Lösung gibt, die verwendet werden könnte.
Insgesamt wurde die in Tabelle 4.1 dargestellte Anforderungsüberdeckung ermittelt, die nachfolgend begründet wird. Wie zu erkennen ist, erfüllt keine der untersuchten Softwarelösungen die funktionalen Anforderungen vollständig, sodass eine Analyse der nicht-funktionalen Anforderungsüberdeckung nicht notwendig ist.

	Existierende Lösung		
Anforderung	ELK Stack	Blitz4j	Splunk
FA1 - verteiltes Logging	✓	(✓)	✓
FA2 - Integrierbarkeit	(✓)	X	X
FA4 - Kompatibilität	✓	X	✓
FA5 - Konfiguration zur Laufzeit	X	(✓)	X
FA6 - Kategorisierung der Log Einträge	✓	X	✓
FA7 - Visualisierungs- & Konfigurationswerkzeug	(✓)	X	(✓)

Legende:

- ✓ Anforderung erfüllt oder nur minimale Anpassungen des Werkzeugs oder der Infrastruktur notwendig
- (✓) Anforderung teilweise erfüllt oder Erfüllung bedingt umfangreichere Anpassungen des Werkzeugs
- X Anforderung nicht bzw. sehr eingeschränkt erfüllt

Tabelle 4.1: Anforderungsüberdeckung existierender Lösungen

4.4.1 ELK Stack

Durch seine Flexibilität erfüllt der ELK Stack alle Anforderungen bis auf den Konfigurationsaspekt zumindestens teilweise bzw. lässt sich dahingehend erweitern.
LogStash als Log-Integrationskomponente des Stacks erfüllt aufgrund der verschiedenen Empfangsmodule sowohl *FA4 - Kompatibilität* als auch die Sammlung der Log-Einträge.

Zusammen mit der Persistierungsmöglichkeit von ElasticSearch wird damit *FA1 - verteiltes Logging* erfüllt. ElasticSearch indiziert und analysiert jedes Feld eines Log-Eintrages, sodass *FA6 - Kategorisierung der Log Einträge* gewährleistet werden kann.
Da durch Kibana die Visualisierung der gesammelten Daten sichergestellt ist, ist *FA7 - Visualisierungs- & Konfigurationswerkzeug* zumindestens teilweise erfüllt. Ein Konfigurationswerkzeug wird nicht angeboten, weil der ELK Stack nur bereits erzeugte Log-Einträge betrachtet und der Fokus daher außerhalb der (Erzeugungs-)Konfiguration liegt. Daher erfüllt der ELK Stack *FA5 - Konfiguration zur Laufzeit* nicht. Die letzte Anforderung, *FA2 - Integrierbarkeit*, erfüllt der ELK Stack nicht direkt. ElasticSearch, LogStash und Kibana ließen sich als dedizierter Service in einer EMI verwenden, allerdings ohne mit anderen EMI Services (Monitoring!) interagieren zu können. Aufgrund der Open Source Natur dieser Werkzeuge, könnte aber ähnlich dem Legacy Wrapper Pattern [Erl09] ein EMI-Datenadapter-Wrapper für LogStash und ein Metrikkernel-Wrapper für ElasticSearch entwickelt werden, um die Anforderung zu erfüllen.

4.4.2 Blitz4j

Blitz4j, als Erweiterung von Log4j, ist eine Logging Bibliothek. Dadurch kann Blitz4j die die infrastrukturellen Anforderungen *FA2 - Integrierbarkeit*, *FA4 - Kompatibilität*, *FA6 - Kategorisierung der Log Einträge* und *FA7 - Visualisierungs- & Konfigurationswerkzeug* nicht erfüllen. Die übrigen Anforderungen, *FA1 - verteiltes Logging* und *FA5 - Konfiguration zur Laufzeit* erfüllt Blitz4j zumindestens eingeschränkt bzw. durch entsprechende Erweiterung. Die Verwendung eines Log4j-kompatiblen Appenders ermöglicht den Versand der erzeugten Log-Einträge, sodass für *FA1 - verteiltes Logging* zusätzlich ein Service zur Integration und Persistierung der empfangenen Log-Einträge konzeptioniert werden müsste. Für *FA5 - Konfiguration zur Laufzeit* wäre diese Persistierungsmöglichkeit ebenfalls notwendig, denn der Appender lässt sich zwar zur Laufzeit konfigurieren, eine Konfiguration ist aber auf das LogLevel beschränkt und lässt sich nicht persistieren. Insgesamt erfüllt Blitz4j diese beiden Anforderung also nur, wenn ein entsprechender Backend Service angeboten wird.

4.4.3 Splunk

Als kommerzielle Alternative zum ELK Stack ist die Anforderungsüberdeckung von Splunk fast identisch. Durch die Splunk Vermittler ist die Sammlung von Log-Einträgen aus heterogenen Quellen möglich, was sowohl *FA4 - Kompatibilität* als auch in Kombination mit der Persistierungsfähigkeit der Splunk Indizierer *FA1 - verteiltes Logging* erfüllt. Deren umfangreichen Indizierungsmöglichkeiten ermöglicht außerdem *FA6 - Kategorisierung der Log Einträge* durch entsprechende Flags.
Zur Visualisierung der Daten bietet Splunk ein umfangreiches Analyse- & Reportingtool. Wie der ELK Stack startet der Lebenzyklus eines Log-Eintrages für Splunk mit dessen Erhalt, sodass eine Konfiguration der Log-Erzeugung nicht teil des Funktionsumfanges

ist. Daher wird *FA5 - Konfiguration zur Laufzeit* überhaupt nicht und *FA7 - Visualisierungs- & Konfigurationswerkzeug* durch das dementsprechend fehlende Konfigurationswerkzeug nur teilweise erfüllt.
Aufgrund seiner kommerziellen Natur und da nur bestimmte Schnittstellen verfügbar sind, ließe sich Splunk sehr schwierig kollaborativ, d.h. fähig mit anderen EMI Services (z.B. Monitoring) zu interagieren, in einer EMI verwenden, sodass Splunk *FA2 - Integrierbarkeit*nicht erfüllt.

4.5 Fazit

Wie Abschnitt 4.4 gezeigt hat, erfüllt keine vorgestellte Lösung alle Anforderungen, sodass zumindestens eine teilweise Eigenentwicklung notwendig ist. Da sich alle Lösungen nur sehr schwierig bzw. gar nicht in EMI integrieren lassen und speziell der Konfigurationsaspekt nicht unterstützt wird, wird aus Flexibilitätsgründen eine vollständige Eigenentwicklung präferiert. Der Fokus des ELK Stack bzw. Splunk liegt außerdem auf dem Log Management, also der Analyse und dem Reporting basierend auf Log-Einträgen. Der ELK Stack und Splunk setzen voraus, dass die Anwendungen bereits verteiltes Logging unterstützen, d.h. alle Logger entsprechend konfiguriert sind bzw. ein Zugriff auf die Log-Dateien möglich ist. In der EMI-Referenzarchitektur ist diese Konfiguration nicht gegeben, sodass in dieser Arbeit durch eine Eigenentwicklung in Form einer Logging Infrastruktur zunächst das verteilte Logging inklusive der Konfiguration zur Laufzeit sichergestellt werden muss.
Diese Logging Infrastruktur lässt sich anschließend als Grundlage für weiterführende Aufgaben, wie z.B. die Analyse mit dem ELK Stack verwenden. In den folgenden Kapiteln wird diese Logging Infrastruktur konzeptioniert.

5 Architekturentwurf

> There are no rules of architecture
> for a castle in the clouds.
>
> G. K. Chesterton

Inhalt

Logging spielt beim Betrieb eines Softwaresystemes eine zentrale Rolle. Voraussetzung für die Verwendbarkeit des Loggings ist die Zugriffsmöglichkeit auf die protokollierten Ereignisse. Während in monolithischen Systemen der Zugriff auf diese meist implizit möglich ist, ist der Zugriff in verteilten Systemen eine Herausforderung, da insbesondere keine Integration der protokollierten Ereignisse stattfindet. Für diese Herausforderung des verteilten Loggings soll im Folgenden ein Lösungskonzept vorgestellt werden, welches neben der Log Integration, d.h. dem Zusammenführen der Logeinträge der verteilten Systeme, auch die zentral gesteuerte Konfiguration der Ereignisprotokollierung ermöglicht. Das zu betrachtende Systemumfeld stellt dabei eine EMI Messinfrastruktur dar.

Da keine der analysierten verwandten Arbeiten (Kapitel 4) die von den Logging Bedürfnissen der EMI Stakeholder abgeleiteten Anforderungen (vgl. Kapitel 3) vollständig erfüllt, wird eine EMI spezifische Lösung entwickelt, die in diesem Kapitel entworfen wird.

Aufgrund des Systemumfeldes in Form einer EMI, ist die Logging Infrastruktur konzeptionell im Bereich der Enterprise Application Integration (EAI) anzusiedeln. Da die verwendete Integrationstopologie der EMI-Referenzarchitektur der Bus [Hor14] ist, wird die Logging Infrastruktur aus Homogenitätsgründen ebenfalls anhand des Enterprise Service Bus (ESB) Architekturmusters [SHLP05] entworfen.

Die Darstellung der Architektur in den folgenden Abschnitten folgt dabei dem von Ludewig und Lichter [LL10] beschriebenen Top-Down-Ansatz, d.h. ausgehend von einer groben Beschreibung der Schichten, erfolgt eine Dekomposition in Komponenten und deren Schnittstellen. Anschließend kann dann das Laufzeitverhalten in Form der

Zusammenarbeit der Komponenten modelliert wird, sodass sowohl die statische als auch die dynamische Struktur der Logging Infrastruktur dargestellt ist. Da die Logging Infrastruktur im Rahmen einer EMI verwendet werden soll, modelliert anschließend Abschnitt 5.4 dieses Integrationsszenario.

5.1 Schichten

Zunächst soll in Abbildung 5.1 ein Überblick über die entworfene Logging Infrastruktur gegeben werden. Wie zu erkennen, ist die Architektur in die vier Schichten Remote, Datentransports & Integration, Verwaltung & Persistierung, sowie Visualisierung aufgeteilt und damit angelehnt an die EMI Referenzarchitektur (Unterabschnitt 2.3.1) konzeptioniert.

Die einzelnen Schichten werden nachfolgend beginnend mit der untersten Schicht, d.h. der Remote Schicht, beschrieben. Anschließend wird die Architekturentscheidung begründet.

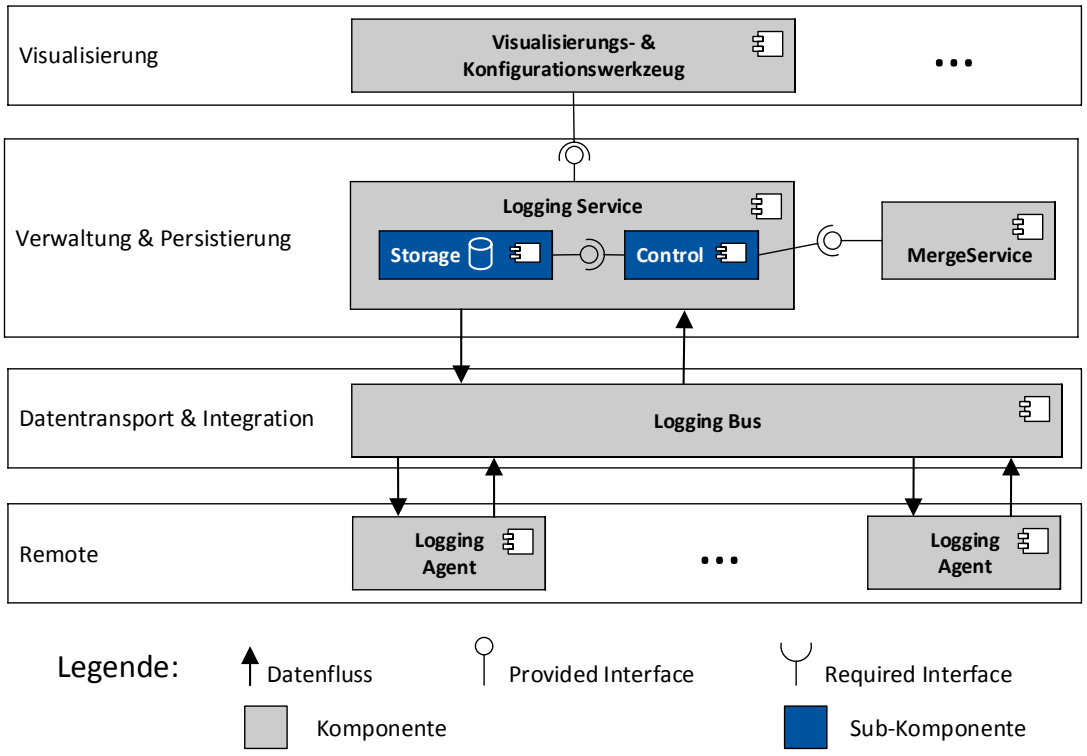

Abbildung 5.1: Logging Infrastruktur Schichtendarstellung

Remote Die Remote Schicht ist für die Anbindung externer Systeme an die Logging Infrastruktur zuständig. Bezogen auf die EMI Referenzarchitektur übernimmt

diese Schicht gleichzeitig die Rolle der Datenanbieter und der Datenadapter, denn die Teilnehmer bieten über diese Schicht ihre Daten abstrahiert von der durch verschiedene Logging Bibliotheken eingeführten Heterogenität an. Neben dem abstrahierten Zugriff auf die Logging Infrastruktur spezifischen Daten, d.h. die protokollierten Ereignisse und die Konfigurationen der Logger, stellt diese Schicht Schnittstellen zur lokalen Konfigurationsänderung der Logger des adaptierten Service zur Verfügung.

Datentransport & Integration Die Transportschicht übernimmt die Vermittlung der versendeten Daten und stellt den korrekten Empfang sicher. Die Daten werden dabei von Komponenten der Remote Schicht oder von Komponenten der Verwaltungs- & Persistierungsschicht versendet bzw. empfangen. Da die Nachrichten aufgrund der Bustopologie in Form eines Broadcasts veröffentlicht werden und der Emfpang nicht bestätigt wird, handelt es sich um eine unidirektionale Kommunikation.

Verwaltung & Persistierung In der dritten Schicht werden die empfangenen Daten verarbeitet und gespeichert. Die Verarbeitung umfasst dabei insbesondere Mechanismen zur Auflösung von Konflikten zwischen gespeicherter und empfangener Logger Konfiguration. Des Weiteren dient diese Schicht als Datenschnittstelle für die nachfolgende Visualisierungsschicht.

Visualisierung In der obersten Schicht werden die Daten der Verwaltungs-& Persistierungschicht durch entsprechende Komponenten dargestellt und dem Anwender die Anpassung dieser Daten, d.h. insbesondere der Konfiguration, ermöglicht.

Durch die Trennung in Schichten und der damit verbundenen Funktionskapselung wird sowohl eine geringe Kopplung als auch eine hohe Kohäsion der einzelnen Schichten ermöglicht. Die Aufteilung orientiert sich dabei an der EMI Referenzarchitektur (vgl. Unterabschnitt 2.3.1) um die geforderte Integrierbarkeit (*FA2 - Integrierbarkeit*) in eine EMI zu erleichtern und folgt den von Ludewig und Lichter empfohlenen Architekturentwurfsprinzipien [LL10] *Trennung von Zuständigkeiten* und *loose Kopplung*.

5.2 Komponenten & Schnittstellen

Nachdem im vorherigen Abschnitt die Schichtenarchitektur der Logging Infrastruktur vorgestellt wurde, beschreibt dieser Abschnitt die wichtigen Komponenten der Logging Infrastruktur. Die in Abbildung 5.1 visualisierten Komponenten werden von unten nach oben bezüglich der Schichtenarchitektur beschrieben.

5.2.1 LoggingAgent

Der Definition eines reaktiven Agenten [WJ95] aus dem Bereich der künstlichen Intelligenz entsprechend, arbeitet der LoggingAgent autonom und kommunikativ und reagiert auf Veränderungen seiner Umwelt. Konkret fungiert der LoggingAgent als Komponente der Remoteschicht als Schnittstelle zum Ansteuern externer Systeme und kapselt die Logging Infrastruktur spezifischen Funktionalitäten für ein externes System. Aus diesem Grund übersteigt seine Funktion die Rolle eines ServiceAdapters [Cha04], denn neben der Anbindung des adaptierten Service an den Logging Bus und der Publizierung der Log Events, verwaltet und konfiguriert der LoggingAgent die Logger des adaptierten Service.
Dazu entdeckt er in einem ersten Schritt selbstständig (Autonomitätseigenschaft) die Logger des adaptierten Service über von ihm verwaltete Loggingbiblitothek spezifische Adapter (*FA4 - Kompatibilität*) und konfiguriert diese zur Veröffentlichung ihrer Log Events auf dem Logging Bus (*FA1 - verteiltes Logging*). Diese Konfiguration publiziert der Logging Agent auf dem Logging Bus um eine Remote Konfiguration über den Logging Service zu ermöglichen (Kommunikationseigenschaft). Zur Laufzeit überwacht er außerdem die lokale Konfiguration der Logger, um Änderungen der Konfiguration dem Logging Service mitteilen zu können (Reaktivitätseigenschaft).
Neben dem Versand der Log Events und Konfigurationsnachrichten konsumiert der LoggingAgent Nachrichten. Im Falle des Empfanges einer Konfigurationsänderung, beispielsweise durch eine vom LoggingSerivce ausgelöste Konfigurationsänderung, wendet der LoggingAgent diese Konfiguration lokal an (*FA5 - Konfiguration zur Laufzeit*). In Kapitel 7 wird der LoggingAgent ausführlich vorgestellt.

5.2.2 LoggingBus

Als Komponente der Transportsschicht ist der LoggingBus für die Vermittlung der Nachrichten innerhalb der Logging Infrastruktur zuständig. Da die Logging Infrastruktur in eine EMI integrierbar sein soll (*FA2 - Integrierbarkeit*), stellt der Logging Bus wie in Abschnitt 5.4 dargestellt, einen dedizierten Kommunikationskanal des Enterprise Measurement Data Bus (EMDB) [Ste13] einer EMI dar. Insgesamt erfolgt die Kommunikation über den LoggingBus also asynchron über Nachrichten und anhand des Publish- & Subscribe ([GHJV95]) Musters. Abschnitt 5.3 beschreibt die Kommunikation innerhalb der Logging Infrastruktur.

5.2.3 LoggingService

Der LoggingService stellt die zentrale Persistierungs- & Verwaltungskomponente der Logging Infrastruktur dar. Er konsumiert die publizierten Nachrichten der LoggingAgents, d.h. die Nachrichten bezüglich Log Events und Logger Konfiguration, und speichert diese um Komponenten der Visualisierungsschicht als Datenschnittstelle zu dienen.
Außerdem bietet der LoggingService Schnittstellen zur Remotekonfiguration der Logging

Agenten, d.h. der Logger der adaptierten externen Systeme. Da die Konfiguration in einem verteilten System zu Konflikten durch Synchronisationsprobleme führen kann, verwendet der LoggingService den Merge Service, der im nachfolgenden Abschnitt vorgestellt wird. Insgesamt kann der LoggingService damit sicherstellen, dass lokal, d.h. LoggingAgent-seitig, angewandte und am LoggingService gespeicherte Konfiguration auch nach einem Neustart des adaptierten Service des LoggingAgents synchron sind und Konfigurationsänderungen nicht verloren gehen. In Kapitel 6 wird der LoggingService ausführlich beschrieben.

5.2.4 MergeService

Der MergeService dient der Konfigurationskonfliktlösung. Da zum einen mithilfe des LoggingService eine Remotekonfiguration, also die Konfiguration von entfernten Loggern durchgeführt werden kann und zum anderen eine lokale Konfigurationsänderung möglich ist (*FA5 - Konfiguration zur Laufzeit*), führt der Verteilungsaspekt zu Synchronisationsproblemen und damit zu Konfigurationskonflikten. Eine Konfliktlösung mithilfe einer Heuristik ist hierbei nicht möglich, da die lokale Konfiguration keinen Zeitstempel besitzt und jede Zeitstempel unabhängige Heuristik den Verlust der Konfigurationsflexibilität (sowohl lokal und remote kann konfiguriert werden) zur Folge hätte. Aus diesem Grund löst der MergeService Konfigurationkonflikte und entscheidet, welche (gemeinsame) Konfiguration verwendet werden soll. Da die Konfigurationen vom LoggingService verwaltet werden, bietet er dem LoggingService eine entsprechende synchrone Schnittstelle, d.h. nicht über den LoggingBus, um eine sinnvolle Fehlerbehandlung zu ermöglichen. Zusammen mit dem LoggingService wird der MergeService ausführlich in Kapitel 6 beschrieben.

5.2.5 Visualisierungs- & Konfigurationswerkzeug

Direkt aus *FA7 - Visualisierungs- & Konfigurationswerkzeug* abgeleitet, präsentiert das Visualisierungs- & Konfigurationswerkzeug die Daten des LoggingService, d.h. es visualisiert die Logdaten und stellt die Logger Konfigurationen der durch LoggerAgents adaptierten Services dar. Neben der Visualisierung ermöglicht das Werkzeug über die entsprechenden Schnittstellen des LoggingService diese Konfigurationen anzupassen.
In Kapitel 8 wird das Werkzeug ausführlich beschrieben.

5.3 Kommunikation

Die bisherigen Abschnitte haben die statische Architektur der Logging Infrastruktur vorgestellt. In diesem Abschnitt wird der dynamische Aspekt der Architektur berücksichtigt. Dabei beschränkt sich dieser Abschnitt auf die Zusammenarbeit der einzelnen Komponenten zur Laufzeit. Das Laufzeitverhalten der einzelnen Komponenten wird in ihren jeweiligen Kapiteln (6, 7) vorgestellt.
Um die Zusammenarbeit der einzelnen Komponenten beschreiben zu können, muss

zunächst ein weiterer statischer Aspekt der Logging Infrastruktur, nämlich die Kommunikationsbasis, vorgestellt werden.

5.3.1 Kommunikationsbasis

Wie der Architekturentwurf in Abschnitt 5.1 darstellt, erfolgt die Kommunikation anhand des Message Bus Enterprise Pattern [HW03], d.h. über eine gemeinsames, austauschbares Kommunikationsmedium. Über diesen Datenbus kommunizieren LoggingAgent und LoggingService anhand der Publish & Subscribe Muster ([GHJV95]) asynchron über Nachrichten miteinander.
Zur Entkoppelung vom konkreten Kommunikationsmedium soll die Kommunikation mithilfe der abstrakten LoggingSender bzw. LoggingReceiver (vgl. Abbildung 5.2) realisiert werden, die erst in einer konkreten Ausprägung das Kommunikationsmedium festlegen, sodass das Kommunikationsmedium der Logging Infrastruktur beliebig gewählt werden kann. Die Kommunikation erfolgt dabei über LoggingMessages (vgl. Unterabschnitt 5.3.3), sodass eine Nachrichtenhierachie aufgebaut werden kann.

Abbildung 5.2: Kommunikationsbasis - Logging Sender & Empfänger

Insgesamt lässt sich somit eine flexible und erweiterbare Handhabung durch Ausnutzung des Polymorphie-Konzeptes im Rahmen der Objekt Orientierung realisieren. Zur Modellierung der Nachrichtenhierachie wird im nächsten Unterabschnitt zunächst das dynamische Verhalten in Form der Zusammenarbeit der Komponenten modelliert.

5.3.2 Kollaboration

In diesem Unterabschnitt wird die Zusammenarbeit der Komponenten zur Laufzeit beschrieben. Das Laufzeitverhalten der einzelnen Komponenten wird in den entsprechenden Komponentenkapiteln (6, 7) beschrieben und an dieser Stelle daher größtenteils [1] nicht betrachtet. Auch wird das zur Veranschaulichung ausreichende Standartverhalten dargestellt und Sonderfälle ignoriert. Diese werden ebenfalls in den entsprechenden Komponentenkapiteln betrachtet. Insgesamt wird in Kombination

[1] Außer wenn es aus Verständlichkeitsgründen notwendig ist, das Verhalten der Komponente kurz zu erläutern

mit den Komponentenkapiteln der dynamische Aspekt der Architektur der Logging Infrastruktur vollständig beschrieben.
Zur Modellierung der Zusammenarbeit verwendet dieser Abschnitt UML-Sequenzdiagramme. Wie von B. Rumpe beschrieben [Rum11] werden Sequenzdiagramm zur Modellierung der Objekteinteraktion verwendet. Da in diesem Abschnitt die Interaktion der Komponenten modelliert werden soll, wird die Objektdefinition auf Systeme erweitert. Auch werden nach der Definition von Rumpe [Rum11] die Transitionen zwischen den Objekte mit den entsprechenden Methodenaufrufen annotiert. In diesem Kapitel liegt der Fokus allerdings auf dem Entwurf, sodass die Annotationen in den in diesem Abschnitt verwendeten UML-Sequenzdiagrammen eine erklärende Funktion übernehmen und keinen Methodenaufrufen entsprechen.

Grundsätzlich lassen sich zwei verschiedene Szenarien identifizieren, die zur Laufzeit der Logging Infrastruktur eintreten können und eine Reaktion auslösen. Entweder tritt ein Log Event auf (wegen *FA1 - verteiltes Logging*) oder eine Konfigurationsänderung wurde durchgeführt (wegen *FA5 - Konfiguration zur Laufzeit*). Beide Szenarien werden im Folgenden betrachtet.

Log Event

Das abgewandelte UML-Sequenzdiagramm in Abbildung 5.3 stellt die Zusammenarbeit der Komponenten innerhalb der Logging Infrastruktur dar. Tritt ein Log Event auf, kategorisiert der Logging Agent es entsprechend (*FA6 - Kategorisierung der Log Einträge*) und publiziert es gemäß des **Push-Forward** Datenadaptierungsmuster [Ste13] auf dem LoggingBus. Der LoggingService konsumiert diese Nachricht, speichert das empfangene Log Event und benachrichtigt das Visualisierungs-& Konfigurationswerkzeug, dass neue Logdaten vorhanden sind. Das Visualisierungs- & Konfigurationswerkzeug aktualisiert daraufhin seinen Datenbestand. Die Kommunikation zwischen LoggingService und Visualisierungs- & Konfigurationswerkzeug orientiert sich damit am **Invoke-Push** Kommunikationsmuster [Ste13].

Konfigurationsänderung

Da nach *FA5 - Konfiguration zur Laufzeit* die Konfigurationsänderung sowohl lokal am LoggingAgent als auch remote über die Schnittstellen des LoggingService ausgelöst werden kann, sind zwei verschiedene Szenarien zu betrachten, nämlich die lokale und die entfernte Konfigurationänderung (Remotekonfigurationsänderung).

Lokale Konfigurationsänderung Der Ablauf einer lokalen Konfigurationsänderung ist in Abbildung 5.4 dargestellt. Detektiert der Logging Agent eine lokale Konfigurationsänderung, publiziert er eine entsprechende Nachricht auf dem

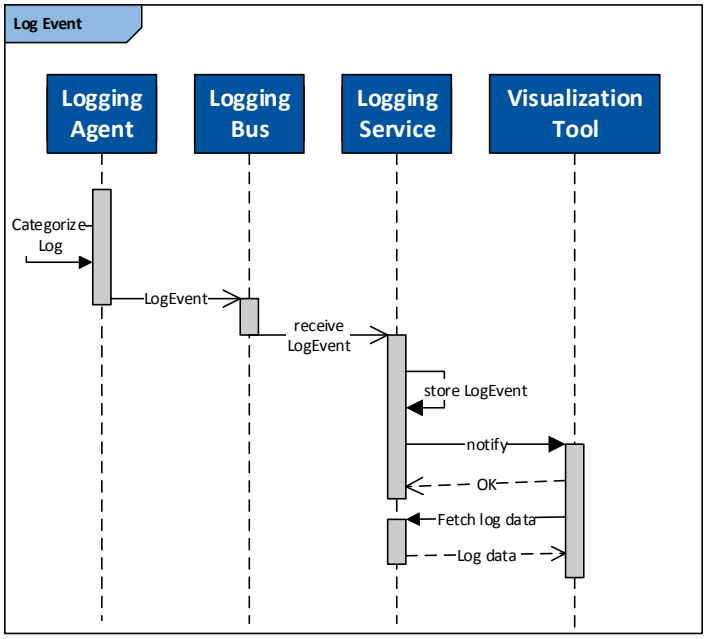

Abbildung 5.3: Log Event - Kommunikation

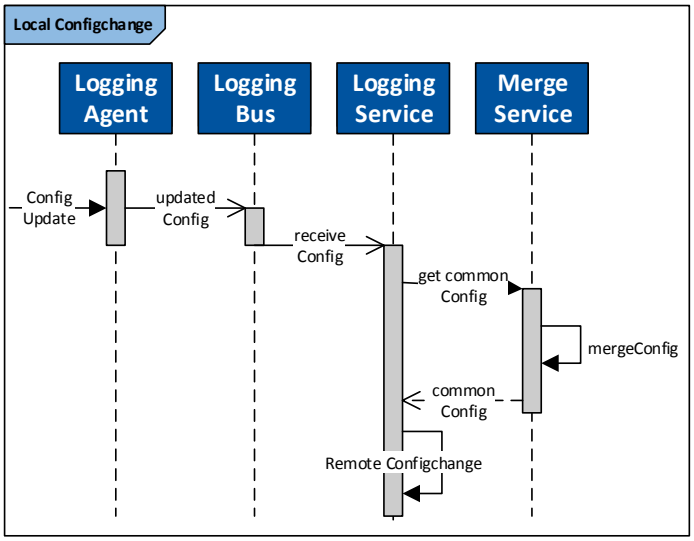

Abbildung 5.4: Lokale Konfigurationsänderung - Kommunikation

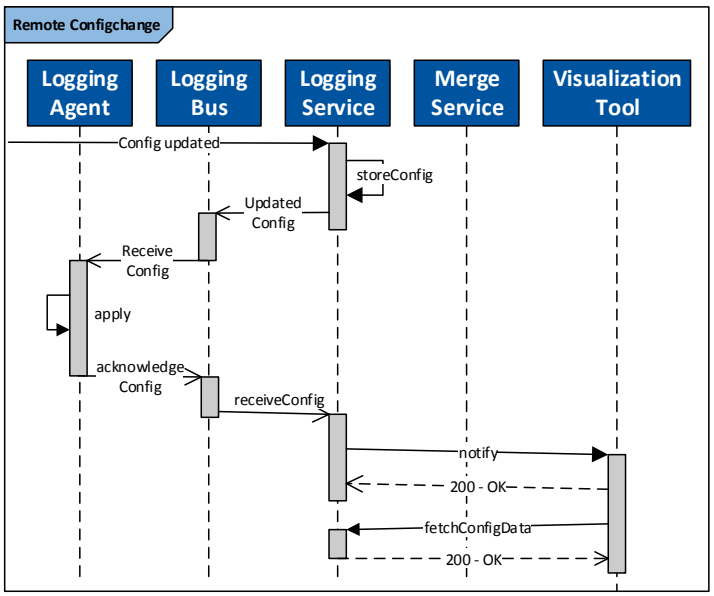

Abbildung 5.5: Remote Konfigurationsänderung - Kommunikation

Logging Bus, welche vom Logging Service konsumiert wird. Aufgrund des beschriebenen Konfliktpotenzials (es ist nicht deterministisch, ob die empfangene oder gespeicherte Konfiguration verwendet werden soll) , löst der Logging Service einen eventuellen Konflikt mithilfe des Merge Service. Das Resultat eines Merge Prozesses ist eine (unter Umständen) neue, gemeinsame Konfiguration, sodass sich einer lokalen Konfigurationsänderung, die nachfolgend beschriebende Remotekonfigurationsänderung anschließt.

Remote Konfigurationsänderung Wird über die Schnittstellen des LoggingService eine Konfiguration geändert, wird ein Remote Konfigurationsvorgang eingeleitet, welcher sich am **Request/Acknowledge** Service Muster [Dai11] orientiert und durch eine Angewandt-Markierung jeder Loggerkonfiguration realisiert wird. Wie in Abbildung 5.5 modelliert, speichert der Logging Service also die Konfiguration zunächst mit einer *Nicht-angewandt*-Markierung und publiziert diese auf dem Logging Bus (**Request**). Der entsprechende LoggingAgent konsumiert diese Nachricht, wendet die Konfiguration an und veröffentlicht die nun als angewandt markierte Konfiguration wieder auf dem Logging Bus. Der LoggingService erhält diese Konfiguration (**Acknowledgement**), aktualisiert die gespeicherte Konfiguration und benachrichtigt das Visualisierungs- & Konfigurationswerkzeug über diese Aktualisierung, welches die aktualisierten Daten abfragt (**Invoke-Push**).

5.3.3 Nachrichten

Wie in Unterabschnitt 5.3.1 modelliert, kommunizieren LoggingSender und LoggingReceiver mithilfe der LoggingMessage miteinander. Dieser Basisnachrichtentyp soll durch Vererbung spezialisiert und somit eine Nachrichtenhierachie aufgebaut werden, die Flexibilität und Erweiterbarkeit durch Ausnutzung des Polymorphie-Konzeptes ermöglicht.
Nachdem im vorigen Unterabschnitt die Kollaboration der Komponenten der Logging Infrastruktur beschrieben wurde, lassen sich zwei Nachrichtentypen identifizieren: Zur Kapselung der Kommunikationsdaten einer LogEvent Kommunikation (5.3.2) wird die LogEventMessage eingeführt und die Repräsentation der relevanten Daten zur Kommunikation einer Konfigurationsänderung (5.3.2) erfolgt mithilfe der LoggerConfigSynchronisationMessage.
Um die nachträgliche Erweiterbarkeit um neue Nachrichtentypen sicherzustellen, werden diese beide Nachrichten durch die LoggingNotificationMessage bzw. die LoggingCommandMessage generalisiert, welche semantisch einer Benachrichtigung bzw. einem Befehl entsprechen, sodass bezogen auf die Kommunikationsmuster (push, pull) alle Kommunikationsarten abgebildet werden können (push = Benachrichtigung, pull = Befehl).
Insgesamt stellt sich die Nachrichtenhierachie damit wie in Abbildung 5.6 dar. Die konkreten Ausprägungen der LogEventMessage und der LoggerConfigSynchronisationMessage werden im Folgenden detaillierter beschrieben. Grundsätzlich orientiert sich der Payload einer LoggingMessages dabei an EMI-Nachrichten und wird daher an dieser Stelle vernachlässigt.

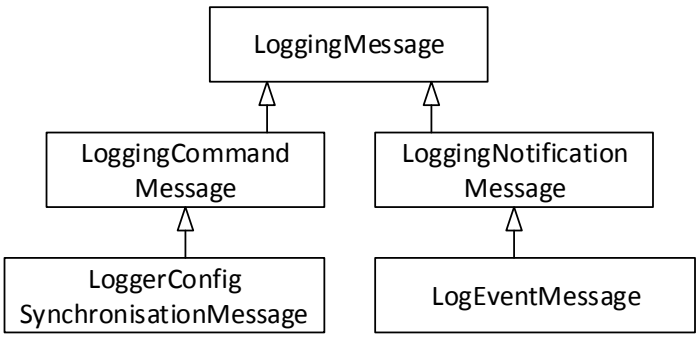

Abbildung 5.6: Nachrichtentypen

LoggerConfigSynchronisationMessage

Die LoggerConfigSynchronisationMessage muss konfigurationspezifische Daten kapseln, um eine Konfigurationsänderung kommunizieren zu können. Um mehrere Logger auf

einmal konfigurieren bzw. mehrere lokale und entfernte Konfiguration gleichzeitig synchronisieren zu können, verwendet eine LoggerConfigSynchronisationMessage den sogenannten **LoggerConfigCache**, der in Kapitel 7 ausführlich vorgestellt wird. Grob dargestellt hält der LoggerConfigCache alle Konfigurationen der Logger des vom LoggingAgent adaptierten Services in Form einer Hierachie (Composite Pattern [GHJV95]) vor. Da aufgrund *FA4 - Kompatibilität* heterogene Konfigurationen verwaltet werden müssen, werden die Logger Konfigurationen basierend auf dem **Variabilitätsmodell** von Ricaro Tavizon [Tav11], Thomas Rölling [Röl14] und Martin Mädler [Mäd13] abgebildet, was ebenfalls in Kapitel 7 ausführlich modelliert wird.

LogEventMessage

Die LogEventMessage kapselt die Daten eines protokollierten Ereignisses, beinhaltet als Payload also einen Log-Eintrag mit den typischen Feldern wie Timestamp, Message, Erzeuger und LogLevel. Da nach *FA6 - Kategorisierung der Log Einträge* die Log-Einträge kategorisiert werden müssen, entspricht der Payload einer LogEventMessage entweder einem **FunctionalLog** zur Abbildung eines fachlichen Log-Eintrages welcher die zu vermessende Einheit (EOM - Entity of Measurement) beinhaltet, oder einem **TechnicalLog** zur Darstellungen eines technischen Log-Eintrages. Die genauen Felder sind an dieser Stelle zu vernachlässigen und sind beispielsweise der Realisierung der Visualisierungswerkzeuges zu entnehmen (siehe Abschnitt 8.2).

5.4 EMI Integration

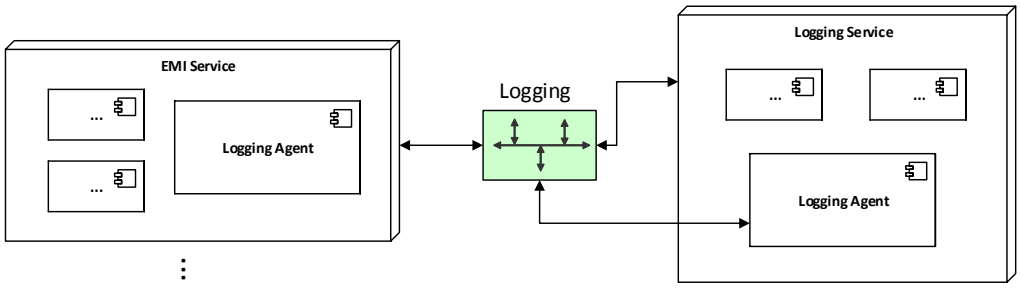

Abbildung 5.7: Integration eines EMI Service in die Logging Infrastruktur

Nachdem in den vorherigen Abschnitten anhand des Top-Down Ansatzes [LL10] die Architektur der Logging Infrastruktur entworfen wurde, widmet sich dieser Abschnitt der Anbindung externer Systeme, präziser formuliert der Integration in eine EMI, was nach *FA2 - Integrierbarkeit* gefordert ist.
In Abbildung 5.7 ist die Anbindung eines EMI Services modelliert. Da der LoggingAgent

alle benötigten Funktionalitäten eines Teilnehmers der Logging Infrastruktur kapselt, ist zur Integration des EMI Service in die Logging Infrastruktur das gemeinsame Deployment des EMI Service und des Logging Agenten im gleichen Kontext ausreichend, wie es in Abschnitt 7.3 veranschaulicht wird.
Da innerhalb einer EMI bereits ein Kommunikationsmedium in Form des EMDB existiert, wird der LoggingBus in Form eines dedizierten Logging Kanals des EMDB realisiert. Insgesamt integriert sich die Logging Infrastruktur damit durch Anbindung aller EMI Service in diese EMI. Da der LoggingService in diesem Szenario ebenfalls einen Service innerhalb der EMI darstellt, bündelt er ebenfalls einen LoggingAgenten.

5.5 Zusammenfassung

In diesem Kapitel wurde angelehnt an die EMI Referenzarchitektur (Unterabschnitt 2.3.1) eine Logging Infrastruktur zur zentralen Integration der Logdaten von verteilten Services, sowie zur erzeugungsspezifischen Konfiguration der Logdaten dieser Datenanbieter konzeptioniert.
Basierend auf einer 4-schichtigen Architektur erfolgt die Datenadaptierung innerhalb der Logging Infrastruktur auf der untersten Schicht durch die Logging Agenten. Über logginbibliothekspezifische LoggingAdapter sind die LoggingAgenten in der Lage heterogene Logger zu konfigurieren und ihre Log-Events zu kommunizieren. Die adaptierten Daten werden asynchron anhand des Publish & Subscribe Muster [GHJV95] auf einem Nachrichtenbus in LoggingMessages veröffentlicht und vom Logging Service, eine Komponente der Verwaltungs- & Persistierungschicht, persistiert und verwaltet, sowie über entsprechende Schnittstellen vom Visualisierungs- & Konfigurationswerkzeug dargestellt. Neben der Darstellung der Daten ermöglicht das Werkzeug eine Remotekonfiguration der Logger der Logging Agenten. Da die Konfiguration auch lokal am Logging Agenten durchgeführt werden kann, verwendet der Logging Service zur Konfliktlösung einen MergeService. Die Architektur der Logging Infrastruktur ermöglicht eine einfach Integration in eine EMI, wobei die Kommunikation über den EMDB erfolgt. Dabei ermöglicht die flexible Kommunikationsschnittstelle der Logging Infrastruktur einen variablen Austausch des Kommunikationsmediums.

Basierend auf diesem Architekturentwurf werden in den nächsten Kapiteln die vorgestellten Komponenten und ihre Arbeitsweise detaillierter beschrieben. Zunächst wird in Kapitel 6 der Logging Service inkl. des benötigten MergeService vorgestellt, Kapitel 7 stellt den Logging Agenten detailliert vor und Kapitel 8 behandelt das Visualisierungs- & Konfigurationswerkzeug. Der UML Spezifikation folgend (siehe [HoEE04]) erfolgt die Detaildarstellung der Komponenten dabei getrennt nach statischer und dynamischer Sicht.

6 Logging Service

Start where you are.
Use what you have.
Do what you can

Arthur Ashe

Inhalt

Im vorherigem Kapitel wurde die Architektur der Logging Infrastruktur modelliert und die Komponenten der Logging Infrastruktur vorgestellt (siehe Kapitel 5). Den Entwurf verfeinernd, werden in den nächsten Kapiteln die eingeführten Komponenten detaillierter vorgestellt. Dieses Kapitel widmet sich den Komponenten der Verwaltungs- & Persistierungsschicht, nämlich dem Logging Service, sowie dem Merge Service.

Der Logging Service ermöglicht die Remotekonfiguration der Logging Agenten, welche externe Systeme adaptieren, um Zugriff auf deren Logger zu erhalten, sodass die Logging Agenten die Log Events der Logger zur zentralen Integration auf dem Logging Bus veröffentlichen. Der Logging Service konsumiert und persistiert diese Log Events und bietet somit eine zentrale Zugriffsschnittstelle auf die protokollierten Ereignisse der externen Systeme.
Da die Konfiguration der Logging Agenten, d.h. die Konfiguration der Logger des vom Logging Agenten adaptierten Systemes, sowohl lokal am Logging Agenten als auch entfernt über den Logging Service erfolgen kann, können Konfigurationskonflikte auftreten, wie es in 5.3.2 an einem Beispiel ausgeführt wird. Aus diesem Grund wird der MergeService zur Konfliktlösung von verschiedenen Konfigurationen benötigt.

Die Darstellung des Logging Service orientiert sich am Top Down Prinzip [LL10]. Ausgehend von einer Einteilung in Schichten, werden im nächsten Abschnitt die zentralen Konzepte und Komponenten des Logging Service beschrieben. Da der Logging Service den MergeService bedingt, wird der Merge Service als Komponente des Logging Service vorgestellt. Nachdem die Komponenten vorgestellt wurden, kann anschließend das Laufzeitverhalten und damit die dynamische Sicht auf den Logging Service dargestellt werden.

6.1 Statische Sicht

Dieser Abschnitt beschreibt die statische Architekturkonstruktion des Logging Service. Neben der Einteilung in Schichten und der Beschreibung der Komponenten, wird das zentrale Modell zur Verwaltung der Logger und ihrer Konfigurationen vorgestellt, der LoggerContext. Außerdem erfolgt eine Darstellung des MergeService, den der Logging Service zur Lösung von Konfigurationskonflikten verwendet.

6.1.1 Schichten

Dieser Abschnitt stellt die Architektur des Logging Service dar. Aufgrund der Zielumgebung einer EMI und der konzeptionellen Vergleichbarkeit mit einem Metrikkernel, orientiert sich die Architektur des Logging Service an der Referenzarchitektur eines Metrikkernels (siehe [Via15]).

Abbildung 6.1 stellt den Aufbau des Logging Service anhand seiner Komponenten dar. Dabei ist der Logging Service in einer dreischichtigen Architektur bestehend aus der Kommunikations-, der Verwaltungs und der Zugriffschicht modelliert. Die einzelnen Schichten werden nachfolgend von unten nach oben bezüglich des Architekturentwurfs vorgestellt.

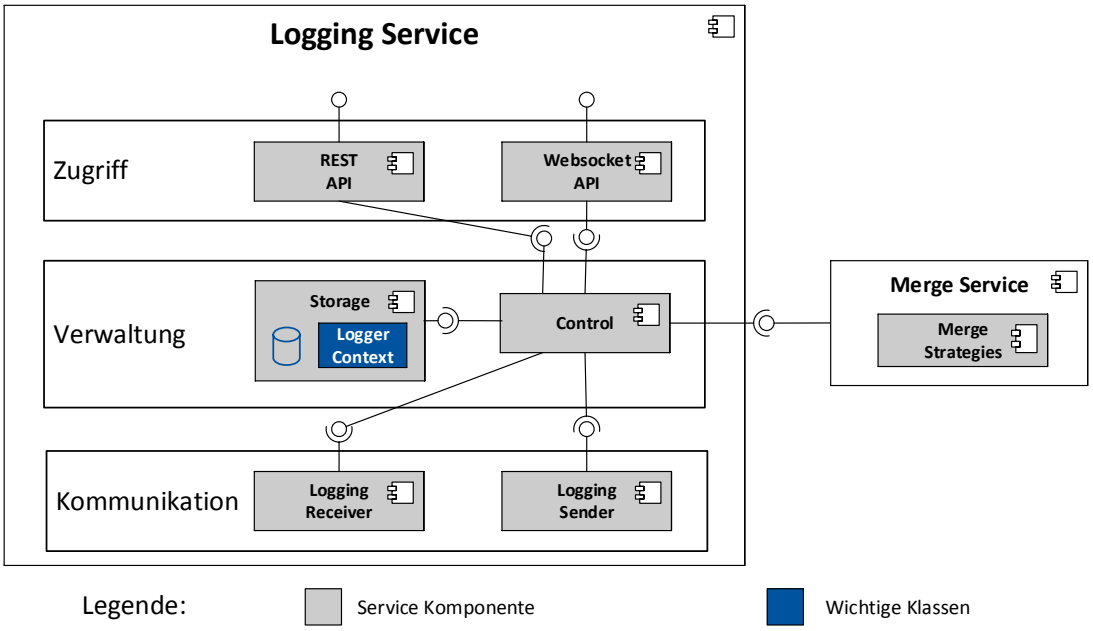

Abbildung 6.1: Logging Service - Komponentendarstellung

Kommunikationsschicht Die Kommunikationsschicht fasst die Komponenten zusammen, die die Anbindung des Logging Service an den Logging Bus ermöglichen

(vgl. EMDB Verbindungsschicht ([Via15]). Mithilfe dieser Komponenten kann der Logging Service Nachrichten auf dem Logging Bus publizieren und konsumieren. Konkret handelt es sich bei diesen Komponenten um den LoggingSender bzw. LoggingReceiver, die in Unterabschnitt 5.3.1 vorgestellt wurden.

Verwaltungsschicht Die Verwaltungsschicht fasst die Anwendungs- und Datenhaltungschicht des von Ludewig und Lichter beschriebenen Drei-Schichten Architekturmusters ([LL10]) durch eine Verwaltungs- und eine Speicherkomponente zusammen. Die Verwaltungskomponente kapselt die fachliche Funktionalität des Logging Service, reagiert also auf empfangene Nachrichten, löst Konfigurationskonflikte mithilfe des Merge Service und verwendet die Speicherkomponente zur Persistierung und zum Zugriff auf die Daten. Die Speicherkomponente realisiert dabei den konkreten Datenzugriff, sowie die konkrete Persistierung. Bezogen auf die Referenzarchitektur eines Metrikkernels entspricht diese Schicht der Kern Schicht (Berechnung und Speicher) (siehe [Via15]). 1

Zugriffschicht Die Zugriffsschicht bietet über verschiedene Schnittstellen den Zugriff auf die Dienstleistungen des Logging Service an, d.h. mithilfe der Komponenten der Zugriffschicht kann das Visualisierungs- & Konfigurationswerkzeug beispielsweise die Logdaten anzeigen, sowie eine Konfigurationsänderung der Logger auslösen. Da das Visualisierungs- & Konfigurationswerkzeug die Daten angelehnt an das Invoke-Push Muster [LVS14] adaptiert, beinhaltet die Zugriffsschicht neben einer RESTful Schnittstelle (pull-Schnittstelle) eine Websocket-Komponente zur bidirektionalen Kommunikation (invoke-Schnittstelle) des Logging Service mit dem Visualisierungs- & Konfigurationswerkzeug.

6.1.2 LoggerContext

Nachdem die Architektur und die Komponenten des Logging Service im vorherigem Abschnitt kurz vorgestellt wurden, widmet sich dieser Abschnitt dem Konzept zur Verwaltung der Loggerkonfigurationen.
Nach *FA4 - Kompatibilität* muss die Logging Infrastruktur mit verschiedenen Loggingbibliotheken kompatibel sein, sodass heterogene Konfigurationen abgebildet werden müssen. Da die Verwaltung der Konfigurationen durch den Logging Service erfolgt, benötigt er ein Modell zur Verwaltung der Logger Konfigurationen, den LoggerContext.
Das Klassendiagramm bezüglich des LoggerContext ist in Abbildung 6.2 dargestellt. Aus Flexibilitäts- & Erweiterbarkeitsgründen wird die Loggerkonfiguration nicht singulär modelliert, sondern in Kombination mit einem Logger, welche der LoggerContext verwaltet.
Konzeptionell lässt sich ein LoggerContext mit der System- bzw. Laufzeitumgebung seiner Logger, d.h. typischerweise einem Logging Agenten vergleichen. Durch die Verwendung eines Composite-Patterns [GHJV95] lassen sich die Logger eines Contextes

hierachisch anordnen, um beispielsweise mehrere Logger auf einmal zu konfigurieren. Die Logger werden dabei schwach referenziert (über ihren Namen und Typ) um auch Logger externer, nicht-objektorienter Systeme verwalten verwalten zu können.
Pro Logger existiert eine LoggerConfig, die beliebige Konfigurationen abbilden kann und im Folgenden vorgestellt wird. Aufgrund der Loggerhierachie entspricht die Konfiguration eines CompositeLoggers dabei den zusammengeführten Konfigurationen seiner Kindlogger, wobei die Zusammenführung mathematisch dem Schnitt der Konfigurationen entspricht, da die Kindkonfigurationen heterogen sein können.

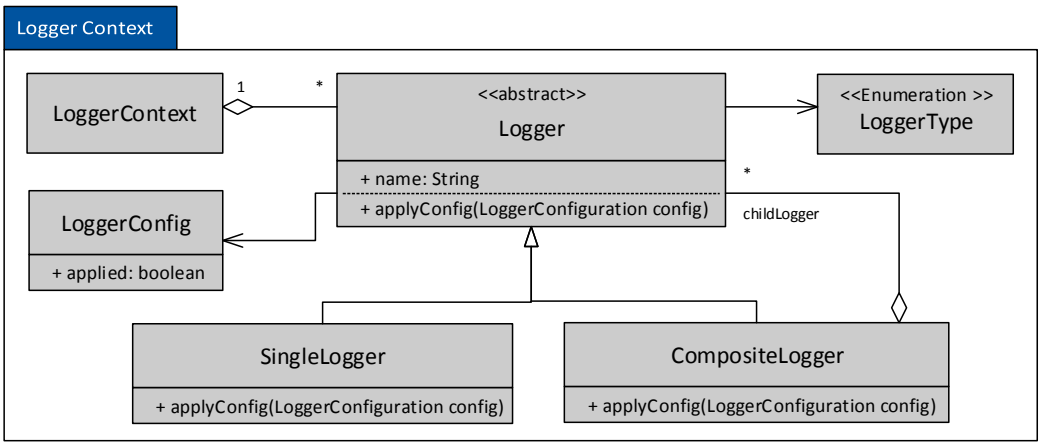

Abbildung 6.2: Logging Service - Loggerverwaltung

LoggerConfig

Bedingt durch die Unterstützung heterogener Logger muss die Konfigurationsabbildung generisch sein. In seiner Masterarbeit „Systematisches, werkzeugunterstütztes Anpassen von Metriken"hat Ricardo Tavizon einen Variabilitätsansatz entwickelt, welcher anschließend von Martin Mädler [Mäd13] und Thomas Rölling [Röl14] überarbeitet wurde. Dieser Variabilitätsansatz soll zur generischen Konfigurationsabbildung verwendet werden.

Die grundsätzliche Idee dieses Variabilitätsansatzes ist die Definition eines Variabilitätsmodelles (VariabilityModel) und der darauf basierenden Variabilitätskonfiguration (VariabilityConfig).
Ein Variabilitätsmodell (siehe Abbildung A.1) beschreibt die Konfigurationsmöglichkeiten. Es besteht aus mindestens einem Variationspunkt, der entweder ein offener Variationspunkt oder ein geschlossener Variationspunkt ist. Ein offener Variationspunkt kann in der Variabilitätskonfiguration beliebige Ausprägungen

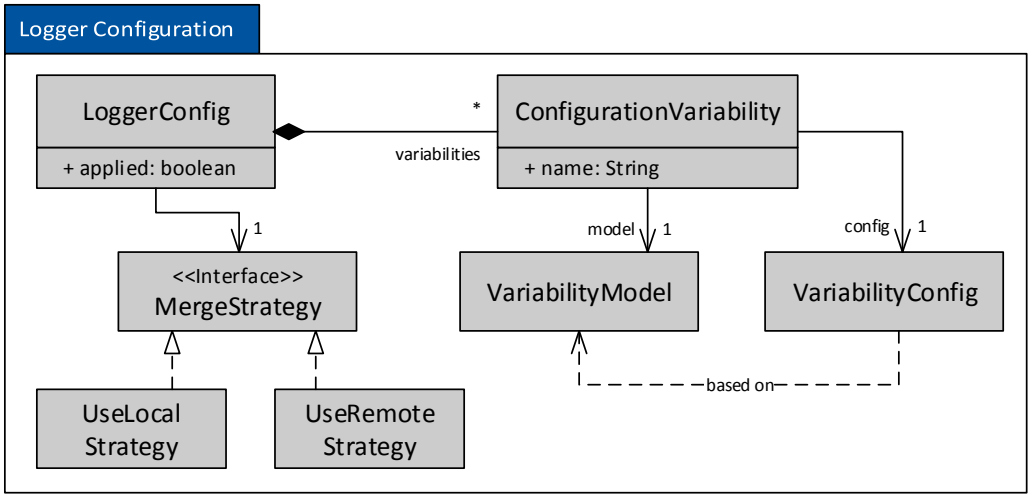

Abbildung 6.3: Loggerkonfiguration Klassendiagramm

seines definierten Typbereichs annehmen. Ein geschlossener Variationspunkt hingegen definiert mehrere Ausprägungsvarianten, wobei eine Variante eine Kombination aus Name und Wert ist.
Basierend auf diesem Modell wird die entsprechende Variabilitätskonfiguration (siehe Abbildung A.2) definiert, d.h. pro Variationspunkt lässt sich eine Variationspunkt-Konfiguration definieren, die entweder einer offene Variationspunkt-Konfiguration oder eine geschlossen Variationpunkt-Konfiguration mit entsprechenden Variantenkonfiguration entspricht.
Insgesamt lässt sich somit eine variable Konfiguration darstellen und die Trennung von Modell und Konfiguration ermöglicht zusätzlich eine einfache Validierung der Konfiguration.

Da Loggerkonfiguration komplex sein können, soll jede Konfiguration beliebig viele Variabilitätsmodelle unterstützen, um beispielsweise eine Kategorisierung mit überlappenden Variabilitätspunkten zu ermöglichen. Veranschaulichen lässt sich eine Überlappung von Variabilitätspunkten anhand mehrerer Appenderkonfiguration. Angenommen einem Logger lassen sich mehrere Appender hinzufügen, dessen Log Level konfigurierbar sind. Würde dieses Szenario in einem einzigen Varibilitätsmodell abgebildet werden, müsste der Log Level Variationspunkt beispielsweise anhand seines Namens einem Appender zugeordnet werden, was fehleranfällig ist. Aus diesem Grund würde es sich anbieten pro Appender (d.h. allgemein pro Konfigurationskategorie) ein eigenes Variabilitätsmodell zu verwenden.

Wie in Abbildung 6.3 modelliert, besteht jede LoggerKonfiguration (LoggerConfig)

daher aus Konfigurationsvariabilitäten (ConfigurationVariability), die einem Variabilitätsmodell (VariabilityModel) und einer Variabilitätskonfiguration (VariabilityConfig) entsprechen und diese gegeneinander validieren können. Das Variabilitätsmodell bzw. die Variabilitätskonfiguration entspricht dabei dem VariabilityModel bzw. der VariabilityConfig von Thomas Rölling [Röl14].

Jeder Loggerkonfiguration besitzt zusätzlich ein „Angewandt"(applied) -Flag. Das Flag realisiert die Bestätigung des Request/Acknowledge Remotekonfigurationsprotokoll (vgl. 5.3.2).

Wie im nachfolgenden Abschnitt beschrieben, löst der Merge Service Konfigurationskonflikte mithilfe verschiedener Merge Strategien. Der Merge Service instanziert die entsprechende Merge Strategie anhand der erhaltenen Merge Informationen. Da das Mergeverhalten für jede Konfiguration einstellbar sein soll, wird die Merge Information in Form der entsprechenden Mergestrategie pro Loggerkonfiguration verwaltet. Die Modellierung der LoggerConfig in Abbildung 6.3, stellt diese Merge Informationen veranschaulicht als Strategy Pattern [GHJV95] dar. Da die MergeStrategy aber wie alle anderen Variabilitäten konfiguriert werden soll, wurde die MergeStrategy in der Implementierung als Konfigurationsvariabilität abgebildet.

Insgesamt ermöglicht die LoggerConfig damit eine generische Konfigurationsabbildung, die nicht nur auf die Domäne der Logger beschränkt ist.

6.1.3 MergeService

Nachdem in den vorangegangenen Abschnitten die Komponenten des Logging Service und das Modell des LoggingContext zur serviceseitigen Verwaltung der Logger vorgestellt wurde, beschreibt dieser Abschnitt den Merge Service, also die zur Konfliktlösungen von Loggerkonfigurationen zuständige Komponente. Wie in 5.3.2 beschrieben, wird der Merge Service zur deterministischen Konfiguration benötigt, die sowohl lokal als auch remote durchgeführt werden kann.
Basierend auf dem Plugin-Architekturmuster [LL10] ist der MergeService in Abbildung 6.4 modelliert. Den *Erweiterungspunkt* der Architektur stell dabei die MergeStrategy (vgl. SeparatedInterface Muster [Fow02]) dar, sodass zur Laufzeit beliebige Strategien zur Konfliktlösung mithilfe des Plugin-Konzeptes hinzugefügt werden können. Die Strategien werden in der MergeStrategyRegistry (vgl. Registry Entwicklungsmuster [Fow02]) verwaltet und dem Host zur Verfügung gestellt, sodass über den MergeController der Konflikt abhängig von der Strategy gelöst werden kann. Die Auswahl der Strategie erfolgt dabei anhand der Merge Informationen, die jede Loggerkonfiguration besitzt (siehe 6.1.2).
Dem Plugin Entwicklungsmuster folgend [Fow02], erfolgt das Linken der Strategien erst zur Laufzeit, sodass jede Strategy über ihren StrategyRegistrator anhand ihres Klassennamenes registriert wird und auf Anfrage initialisiert wird (lazy instantiation).

Die Funktionalitäten des Merge Service werden dabei über die MergeServiceFacade angeboten (vgl. FacadePattern [GHJV95]). Da der Merge Service als eigenständiger Service innerhalb der Logging Infrastruktur realisiert ist, besitzt er wie der Logging Service auch, einen Logging Agent.

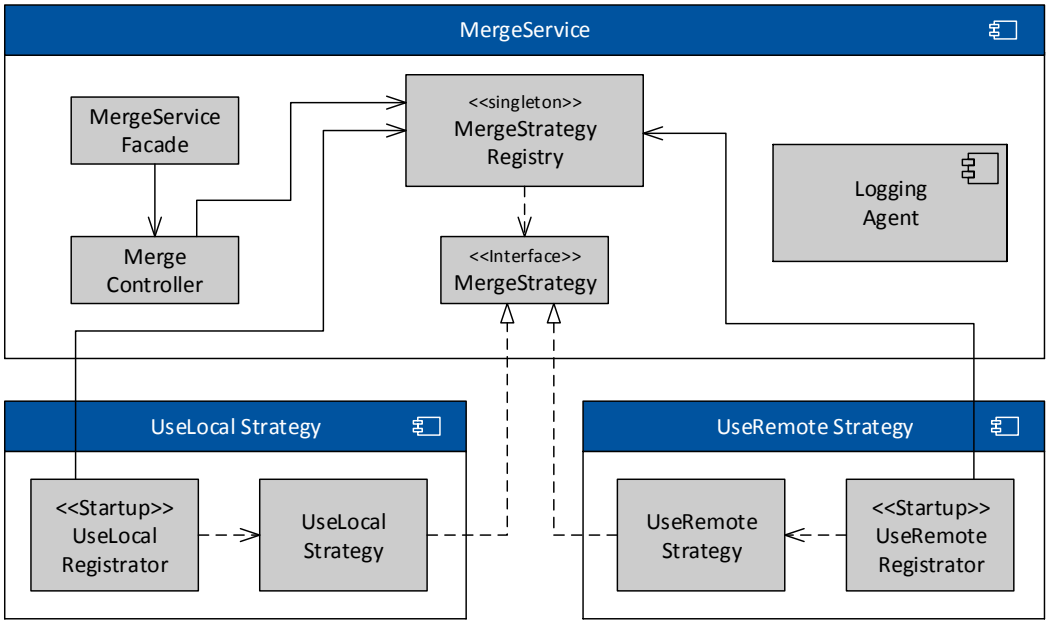

Abbildung 6.4: Komponenten und zentrale Klassen des Merge Service

Aufgrund der Plugin-Architektur ist eine einfache Erweiterbarkeit des MergeService um neue Strategien sichergestellt und das von Ludewig und Lichter beschriebene *Offen-geschlossen-Prinzip* [LL10] realisiert.

6.2 Dynamische Sicht

Nachdem im vorherigem Abschnitt der Logging Service unter statischen Gesichtspunkten vorgestellt wurde, beschreibt dieser Abschnitt das Laufzeitverhalten des Logging Service. Im Architekturentwurf (Kapitel 5) hat Unterabschnitt 5.3.2 die Zusammenarbeit der einzelnen Komponenten der Logging Infrastruktur erläutert und aus Verständlichkeitsgründen das Laufzeitverhalten des Logging Service bereits grob beschrieben. In diesem Abschnitt wird dieses Verhalten detaillierter beschrieben und insbesondere Ausnahme und Fehlerfälle beachtet.
Grundsätzlich lässt sich das Laufzeitverhalten des Logging Service in drei Szenarien aufteilen, die im Folgenden beschrieben werden: Neustart / Deployment des Service,

eine Remotekonfigurationsänderung und der Empfang einer LoggingMessage, also einer Logging Infrastruktur spezifischen Nachricht (siehe Unterabschnitt 5.3.1).

6.2.1 Neustart / Deployment

Wird der Logging Service deployed oder startet neu, muss er die Konfigurationen der Logger Agenten abfragen bzw. geänderte Konfigurationen synchronisieren. Dazu markiert der Logging Service als Heuristik alle gespeicherten Loggerkonfigurationen als nicht angewandt und broadcasted eine speziell markierte LoggerConfigSynchronisation (Wildcard) Nachricht, die die Logging Agenten auffordert ihre aktuellen Loggerkonfigurationen zu verschicken. Durch die in Unterabschnitt 6.2.3 beschriebene Behandlung der Antwortnachrichten sind die lokalen und entfernten Konfigurationen anschließend synchron bzw. im Fehlerfall die Remotekonfigurationen entsprechend als nicht-angewandt markiert, sodass *FA5 - Konfiguration zur Laufzeit* erfüllt ist.

6.2.2 Remotekonfigurationsänderung

Als Remotekonfigurationsänderung wird eine über die Schnittstellen des Logging Service ausgelöste Konfigurationsänderung bezeichnet. Die Konfigurationsänderung muss dabei mit dem entsprechenden Logging Agenten synchronisiert werden. Aufgrund des Anfrage/Bestätigungs-basierten Konfigurationsprotokolls (vgl. 5.3.2) markiert der Logging Service in einem ersten Schritt die geänderte Konfiguration als nicht angewandt und persistiert die Änderungen. Anschließend publiziert er die geänderte Loggerkonfiguration auf dem Logging Bus, was die Anfragephase des Konfigurationsprotokolls beendet. Der Erhalt einer Bestätigungsnachricht ist in Unterabschnitt 6.2.3 beschrieben. Erhält der Logging Service aufgrund eines Fehlers keine Nachricht, ist über die Angewandt-Markierung trotzdem die Korrektheitsanforderung von *FA5 - Konfiguration zur Laufzeit* sichergestellt.

6.2.3 LoggingMessage Empfang

Dieser Abschnitt beschreibt die etwas komplexeren Szenarien des Empfang einer LoggingMessage. Wie im Architekturentwurf in Unterabschnitt 5.3.3 modelliert wurde, existieren zwei konkrete Ausprägungen der LoggingMessage: die LoggerConfigSynchronisationMessage und die LogEventMessage. Aufgrund der unterschiedlichen Semantik beider Nachrichtentypen wird das Verhalten des Logging Service im Folgenden getrennt nach dem erhaltenen Nachrichtentyp beschrieben.

LoggerConfigSynchronisationMessage

Zur Modellierung des Laufzeitverhalten des Logging Service in Reaktion auf dem Empfang einer LoggerConfigSynchronisationMessage, wird vereinfachend vorausgesetzt, dass die empfangene Nachricht nur eine Loggerkonfiguration enthält, obwohl wie in 5.3.3 beschrieben, die LoggerConfigSynchronisationMessage mehrere Loggerkonfigurationen enthalten kann. Diese Einschränkung gilt ohne Beschränkung

der Allgemeinheit und dient nur der Übersichtlichkeit der Modellierung. Sollte eine LoggerConfigSynchronisationMessage mehrere Konfigurationen beinhalten, wird der in Business Process Model Notation [1] modellierte Prozess in Abbildung 6.5 mehrfach durchgeführt.

Erhält der Logging Service eine LoggerConfigSynchronisationMessage, extrahiert

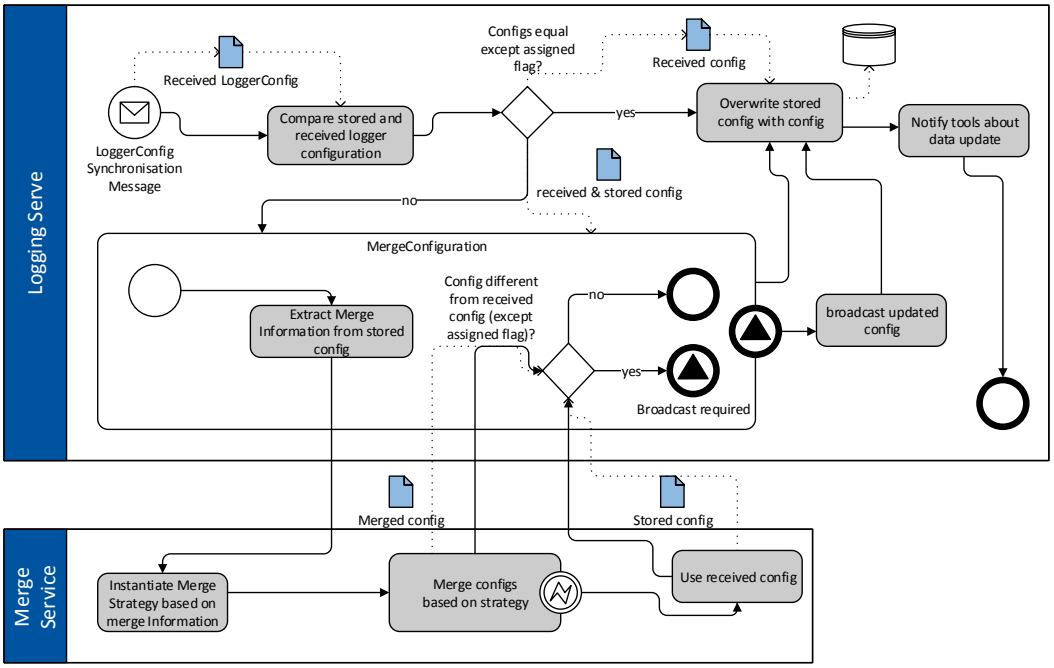

Abbildung 6.5: Logging Service - LoggerConfigSynchronisationMessage Empfang

er zunächst die (nach Annahme einzige) Loggerkonfiguration und vergleicht diese mit der korrespondierenden gespeicherten Loggerkonfiguration. Dieser Vergleich ist notwendig, um die Szenarien einer lokalen Konfigurationsänderung und Rückmeldung auf eine Remotekonfigurationsänderung unterscheiden zu können. Sind beide Konfigurationen bis auf das Angewandt-Flag (siehe 6.1.2) identisch, handelt es sich also um eine Rückmeldung (positiv oder negativ bzw. acknowledgement oder negative acknowledgment), aktualisiert der Logging Service das Angewandt-Flag der gespeicherte Konfiguration anhand der empfangenen Wertes. Auf diese Weise kann das Visualisierungs- & Konfigurationswerkzeug den Status jeder Loggerkonfiguration anzeigen, was dem Anwender Rückschlüsse auf die aktuellen Konfiguration ermöglicht.

Unterscheiden sich empfangene und gespeicherte Konfiguration, führt der Logging Service einen Merge Prozess durch, dessen Resultat eine gemeinsame Konfiguration ist. Der eigentliche Merge Prozess wird dabei vom bereits beschriebenen Merge Service (vgl.

[1] http://www.bpmn.org/

Unterabschnitt 6.1.3) durchgeführt. Anhand der Merge Informationen der gespeicherten Loggerkonfiguration löst der Logging Service einen Merge Prozess aus. Der Merge Service instanziert abhängig von den Merge Informationen die entsprechende Merge Strategie und löst den Konfigurationskonflikt mit dieser Strategie. Diese konfliktfreie Konfiguration speichert der Logging Service anschließend als neue Konfiguration für den entsprechenden Logger und benachrichtigt die angeschlossenen Werkzeuge über eine Aktualisierung des Datenbestandes.
Sollte während des Merge Prozesses ein Fehler auftreten, beispielsweise dass die Merge Strategie nicht instanziert werden kann, wird als Heuristik die empfangene Konfiguration als neue Konfiguration verwendet. War der Merge Prozess erfolgreich, kann die vom Merge Prozess erzeugte Konfiguration unter Umständen verschiedenen von der empfangenen Konfiguration sein, sodass die neue Konfiguration dem entsprechenden Logging Agenten mitgeteilt werden muss. An den Empfang einer LoggerConfigSynchronisationMessage schließt sich also das Szenario einer Remotekonfigurationsänderung an (vgl. Unterabschnitt 6.2.2).

LogEventMessage

Empfängt der LoggingService eine LogEventMessage extrahiert der Logging Service das entsprechende Log Event aus der empfangenen Nachricht und persistiert das Event. Nach einer erfolgreichen Persistierung werden über die Websocket Schnittstelle des Logging Service angeschlossene Werkzeuge über eine Datenbestandsveränderung informiert.

6.3 Zusammenfassung

Dieses Kapitel hat zwei Komponenten der Logging Infrastruktur vorgestellt: Den Logging Service und den Merge Service. Der Logging Service stellt die Datenschnittstelle der Visualisierungs- & Konfigurationswerkzeug dar. Aus diesem Grund ist er für die zentrale Persistierung der protokollierten Ereignisse und Loggerkonfiguration der Logging Agenten zuständig und bietet Schnittstellen um eine Remotekonfiguration durchführen zu können. Da die Konfiguration auch lokal am Logging Agenten durchgeführt werden kann, verwendet der Logging Service den Merge Service zur Konfigurationskonfliktlösung. Basierend auf einer Plugin-Architektur verwaltet der Merge Service unterschiedliche Strategien zur Lösung eines Konfigurationskonfliktes, die abhängig von den Merge Informationen einer Loggerkonfiguration bei Bedarf instanziert werden. Außer den Merge Informationen können die Inhalte der Loggerkonfigurationen beliebig sein, sodass der Logging Service ein Variabilitätsmodell zur Abbildung der Konfiguration verwendet. Des Weiteren ermöglicht die hierarische Abbildung der Loggerkonfigurationen die Konfiguration ganzer Loggergruppen.
Insgesamt ermöglicht die Kombination aus Logging und Merge Service eine generische und flexible Konfiguration, die nicht auf Logger beschränkt ist. Speziell das Potential des Merge Service ließe sich bei entsprechender Erweiterung um beispielsweise komplexere Strategien oder die werkzeugbasierte Erstellung von Strategien nutzen.

Nachdem in diesem Kapitel die Verwaltung der einzelnen Loggerkonfiguration vorgestellt wurde, kann im nächsten Kapitel die konkrete Konfiguration beschrieben werden. Dazu widmet sich das nächste Kapitel den Logging Agenten, die als Schnittstelle zur Integration externen Systeme in die Logging Infrastruktur dienen und daher unter anderem den konkreten Konfigurationsvorgang durchführen.

7 Logging Agent

> Everything has beauty, but not everyone can see.
>
> Confucius

Inhalt

Nachdem in Kapitel 5 die Architektur der Logging Infrastruktur modelliert wurde und Kapitel 6 den Logging Service, also die Komponente zur Verwaltung der protokollierten Ereignisse und der Loggerkonfigurationen, beschrieben hat, widmet sich dieses Kapitel der Adaptionsschnittstelle externer Systeme, die alle Logging Infrastruktur spezifischen Funktionalitäten für ein externes System kapselt: dem Logging Agent.

In Unterabschnitt 5.2.1 wurde der Logging Agent bereits kurz beschrieben und die Begrifflichkeit des Agenten erläutert. Zusammenfassend ist der Logging Agent für die Veröffentlichung der Log Events des adaptierten Systemes und für die Konfiguration der Logger zuständig. Angelehnt an die Vorgehensweise von Blitz4j (vgl. Abschnitt 4.2), erfolgt die Publizierung des Log Events auf dem Logging Bus während der Erzeugung des Log Events (push-basierter Ansatz). Somit benötigt der Logging Agent keinen Zugriff auf die Logdatei, was *FA2 - Integrierbarkeit* erschweren würde. Ebenso konfiguriert der Logging Agent aus diesem Grund einen Logger nicht indirekt über eine Konfigurationsdatei (vgl. Unterabschnitt 2.3.2), sondern direkt zur Laufzeit, wobei eine Konfigurationsänderung über die Konfigurationsdatei weiterhin möglich ist, wie es Unterabschnitt 7.2.2 ausführt.

Insgesamt erleichtert die direkte Konfiguration ohne Dateisystem - Zugriffe *FA2 - Integrierbarkeit*, setzt allerdings voraus, dass der Logging Agent zu unterschiedlichen Logging Bibliotheken kompatibel ist. Wie diese Kompatibilität vom Logging Agenten realisiert wird, beschreiben die folgenden Abschnitte. Zunächst erfolgt die statische Architekturkonstruktion des Logging Agenten (Abschnitt 7.1) und anschließend wird das Laufzeitverhalten (Abschnitt 7.2) modelliert. Der letzte Abschnitt verdeutlicht anhand eines Beispiels, wie die Integration des Logging Agenten in ein externes System funktioniert bzw. welche Schritte notwendig sind.

7.1 Statische Sicht

In diesem Abschnitt wird der Logging Agent unter statischen Gesichtspunkten beschrieben. Dabei werden die zentralen Komponenten und Klassen bzw. Konzepte des Logging Agenten dargestellt.

Ausgehend vom Problem der Loggerheterogenität, also der Existenz beliebiger Logging Bibliotheken im vom Logging Agent adaptierten System, , ist der Logging Agent anhand des Plugin-Architekturmusters [LL10] entworfen. Die Kompatibilität des Logging Agenten zu einer Logging Biblitohek lässt sich mithilfe eines entsprechenden Plugins, dem Logging Adapter, sicherstellen. Durch die zugrunde liegende Plugin-Architektur ist das Offen-geschlossen Prinzip [LL10] umgesetzt und die Kompatibilität zu verschiedenen Logging Bibliotheken lässt sich leicht erweitern. Neben der Lösung der Loggerheterogenität muss der Logger Agent auch lokale Konfigurationsänderungen detektieren, die beispielsweise über eine Konfigurationsdatei (vgl. Unterabschnitt 2.3.2) durchgeführt wurden. Da der Logging Agent keinen Dateisystem Zugriff (insbesondere auf die Konfigurationsdatei) voraussetzt, müssen Konfigurationsänderung auf andere Weise detektiert werden. Da eine Erweiterung der entsprechenden Logging Bibliothek um eine Benachrichtungskomponente bei einer Konfigurationsänderung nicht immer möglich ist, sollen Konfigurationsänderungen durch Vergleich der alten und der aktuell angewandten Konfiguration detektiert werden. Dazu werden alle Konfigurationen in dem LoggerConfigCache vorgehalten und in regelmäßigen Abständen durch den Config Observer auf Gültigkeit geprüft. Insgesamt stellen sich die Komponenten und wichtige Klassen des Logging Agenten damit wie in Abbildung 7.1 dar.

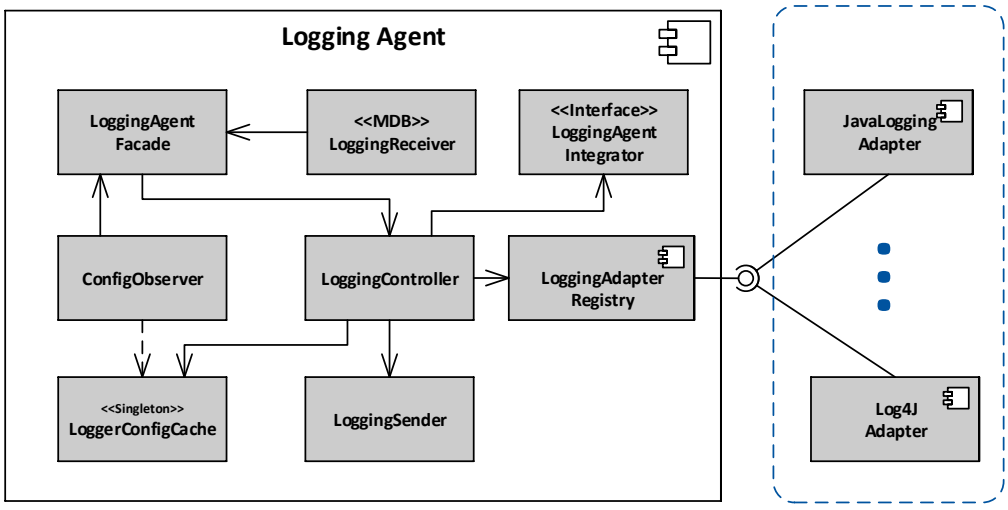

Abbildung 7.1: Komponenten und wichtige Klassen eines Logging Agent

Neben der durch die Plugin-Architektur eingeführten LoggingAdapterRegistry, die die einzelnen Logging Bibliothek spezifischen Adapter verwaltet, orientiert sich der Logging Agent am Facade Muster [GHJV95]. Die LoggingAgentFacade abstrahiert die Implementierungsdetails des Logging Agent und stellt eine definierte Service Schnittstelle zur Verfügung. Für die Geschäftslogik des Logging Agenten ist der Logging Controller zuständig. Dabei wird, obwohl nur ein Controller existiert, das Facade Muster verwendet, um auf einfache Weise *NFA-I-4 - Erweiterbarkeit* sicherzustellen und die Implementierung leicht verändern zu können. Das Verhalten des Logging Controller lässt sich über den LoggingAgentIntegrator steuern, um eine konfigurierbare Adaptierung eines externen Systemes zu ermöglichen. Zur Kommunikation über den Logging Bus verwendet der Logging Agent den LoggingSender zum Versand und den LoggingReceiver zum Empfang von Logging Nachrichten (vgl. Abschnitt 5.3.1). Die restlichen Komponenten des Logging Agenten, der ConfigObserver und der LoggerConfigCache werden in den folgenden Unterabschnitten beschrieben und der LoggingAdapter dekompositiert.

7.1.1 Logging Adapter

Dieser Abschnitt beschreibt die Komponente des Logging Agenten, die die Kompabilität des Logging Agent mit verschiedenen Logging Bibliotheken ermöglicht. In Form eines Logging Adapters (vgl. Adapter Muster [GHJV95]) lassen sich die Loggingbibliothek spezifischen Funktionalitäten abstrahieren. Die spezifischen Funktionalitäten unterteilen sich in die Entdeckung der Logger, die Konfiguration der Logger und die Veröffentlichung der Log Events. Wie in Abbildung 7.2 abgebildet, wird jede Aufgabe von einer eigenen Komponente pro Adapter realisiert, sodass das unter anderem von Ludewig und Lichter empfohlene Prinzip *Trennung der Zuständigkeiten* [LL10] beachtet ist.

Der **LogDiscoverer** ist zuständing für die Entdeckung der Logger seiner Bibliothek, der **ConfigHandler** zur Anwendung der Loggerkonfiguration und der **LogAppender** zur bibliotheksspezifischen Publizierung der Log-Events auf dem entsprechenden Medium. Das Konzept des LogAppenders ist dabei an das Vorgehen von Blitz4J (Unterabschnitt 4.4.2) orientiert.
Gemäß dem Offen-geschlossen Prinzip [LL10], registriert sich jeder Adapter selbstständig in der LoggingAdapterRegistry des Logging Agenten.
Da für jede konkrete Adapterkomponente eine eigene Registry (LoggerDiscovererRegistry, ConfigHandlerRegistry, LogAppenderRegistry) existiert, lassen sich neue Adapter mit bestehenden LoggerDiscoverer-, ConfigHandler- und LogAppender-Implementierungen kombinieren.
Da wie bei den Strategien des Merge Service eine Registrierung über den Klassennamen erfolgt und eine Instanzierung erst beim Zugriff durchgeführt wird (vgl. Unterabschnitt 6.1.3), werden Seiteneffekte minimiert und der Overhead reduziert.
Nachdem die Struktur des Logging Adapters vorgestellt wurde, können im Folgenden die einzelnen Komponenten eines Logging Adapters bzw. ihre Aufgaben angelehnt an das Vertragsmodell (Design by Contract [Mey92]) beschrieben werden.

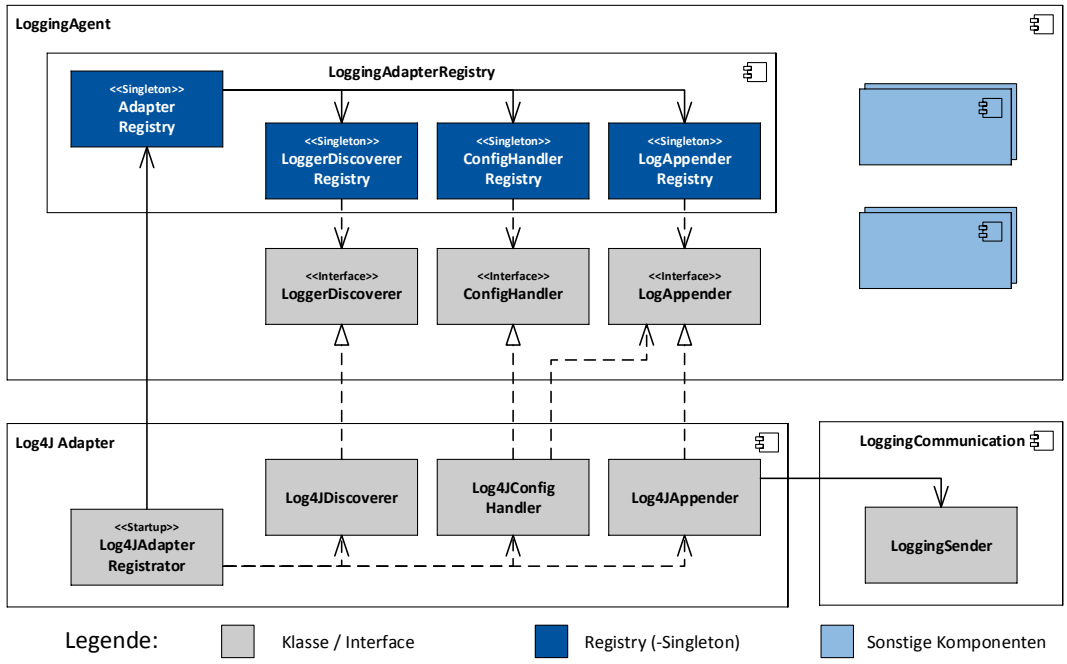

Abbildung 7.2: Aufbau eines Logging Adapter

LoggerDiscoverer

Jeder Logging Adapter benötigt einen LoggerDiscoverer. Der LoggerDiscovierer ist zuständig für die Loggingbibliothek spezifische Entdeckung und Abbildung der Logger dieser Loggingbibliothek.

Logger Entdeckung Mit der Logger Entdeckung wird die Aufgabe eines LoggerDiscoverer bezeichnet, alle konkreten Logger seiner Loggingbibliothek innerhalb seines Systemkontextes zu finden. Wie der Entdeckungsvorgang konkret realisiert wird, ist jedem LoggerDiscoverer überlassen. Typischerweise wird eine Form der statischen Codeanalyse verwendet, um auch noch nicht instantiierte Logger zu entdecken. Nach einem Entdeckungsvorgang kennt ein LoggerDiscoverer alle Logger seiner Loggingbibliothek innerhalb seines Systemkontextes. Wie in Abschnitt 7.2 beschrieben, wird der Entdeckungsvorgang beim Starten des zugehörigen Logging Agenten durchgeführt.

Logger Abbildung Nachdem ein LoggerDiscoverer die Logger seiner Loggingbibliothek entdeckt hat, muss er diese zur späteren Konfigurationsverwaltung im Rahmen des LoggerConfigCache auf die Loggerdomäne der Logging Infrastruktur abbilden. Neben dem reinen Mapping muss der LoggerDiscoverer auch die spezifische Hierachieeigenschaft seiner Loggingbibliothek beachten und entsprechend abbilden, sodass im Transformationsprozess keine Hierachieinformationen verloren

gehen.

ConfigHandler

Zusätzlich zum LoggerDiscoverer benötigt ein LoggingAdapter einen ConfigHandler für die Loggingbibliothek spezifischen Konfigurationsaufgaben. Konkret umfassen die Aufgaben des ConfigHandlers den Logger Zugriff, die Abbildung der Loggerkonfiguration und das Auslesen sowie Anwenden einer Loggerkonfiguration. Insgesamt realisiert hauptsächlich der ConfigHandler *FA5 - Konfiguration zur Laufzeit.*

Logger Zugriff Um die Konfiguration eines Loggers verändern oder auslesen zu können und da die Konfiguration direkt und nicht über eine Konfigurationsdatei erfolgt, benötigt der ConfigHandler Zugriff auf den Logger. Um keine weiteren Abhängigkeiten einzuführen und der Entdeckungszugriff unterschiedlich vom Konfigurationszugriff sein kann, wird diese Aufgabe vom ConfigHandler (und nicht vom LoggerDiscoverer) durchgeführt. Das Ergebnis des Logger Zugriffs ist die konkrete Loggerinstanz.

Konfigurationsabbildung Damit im Rahmen der Logging Infrastruktur eine Remotekonfiguration, also eine Konfiguration über den Logging Service, durchgeführt werden kann, muss der ConfigHandler die unterschiedlichen Konfigurationen (Loggingbibliothek spezifisch vs. generische LoggerConfig) bidirektional abbilden können. Zum einen muss er das Logging Infrastruktur spezifische Konfigurationsmodell (LoggerConfig) in die konkrete, d.h. Loggingbibliothek-spezifische Konfiguration übersetzen können, um vom Logging Service Konfigurationsänderungen durchführen zu können. Zum anderen muss er die Loggingbibliothek-spezifische Konfiguration in eine LoggerConfig übersetzen können, um die Konfiguration auf dem Logging Bus publizieren zu können. Vorbedingung für die Abbildung der konkreten Loggerkonfiguration in eine LoggerConfig (vgl. 6.1.2) ist das erfolgreiche Auslesen der Loggerkonfiguration. Für die Abbildung des Variabilitätsmodelles auf die konkrete Loggerkonfiguration ist keine Vorbedingung notwendig. Das Resultat des Abbildungsvorganges ist die entsprechend abgebildete Konfiguration.

Loggerkonfiguration auslesen Um die Loggerkonfiguration beispielsweise zur Verwaltung & Veränderung durch den Logging Service abbilden zu können, muss der ConfigHandler die Konfiguration zunächst auslesen. Vorbedingung für das Auslesen der angewandten Loggerkonfiguration ist der erfolgreiche Loggerzugriff. Das Resultat ist die Loggingbibliothek spezifische Konfiguration.

Loggerkonfiguration anwenden Damit innerhalb der Logging Infrastruktur eine Konfiguration zur Laufzeit möglich ist, muss der ConfigHandler die Logger seiner Loggingbibliothek konfigurieren können. Vorbedingung für einen Konfigurationsprozess ist das Vorhandensein einer Bibliothekspezifische Konfiguration. Das Resultat des Konfigurationsprozesses ist dann die erfolgreiche Konfiguration des Loggers mit der gegebenen Konfiguration.

RequiredConfig bereitstellen Eine weitere Aufgabe des ConfigHandlers ist das Bereitstellen der RequiredConfig. Die RequiredConfig ist die Loggingbibliothek spezifische Konfiguration, die notwendig ist, um die Logging Infrastruktur spezifischen Anforderungen (*FA1 - verteiltes Logging* und *FA5 - Konfiguration zur Laufzeit*) zu erfüllen. Jeder entdeckte Logger benötigt diese Konfiguration, um seine LogEvents auf dem LoggingBus zu publizieren und die Konfiguration mit Informationen für den Merge Service (Unterabschnitt 6.1.3) anreichen zu können, d.h. eine RequiredConfig entspricht mindestens einer LoggerConfig mit den LoggerConfigVariabilities der Appenderkonfiguration und der Mergestrategie (vgl. 6.1.2). Woher der ConfigHandler diese Konfiguration bezieht (externe Konfigurationdatei, hart-codiert, externer Service), ist ihm überlassen. Das Resultat des RequiredConfig Bereitstellungsprozess muss die benötigte Konfiguration sein.

LogAppender

Der letzte Bestandteil eines Logging Adapter ist neben dem LoggerDiscoverer und dem ConfigHandler der LogAppender. Der LogAppender sorgt für die Veröffentlichung eines LogEvents auf dem Logging Bus. Da im Rahmen der Logging Infrastruktur ein protokolliertes Ereignis kategorisiert wird (*FA6 - Kategorisierung der Log Einträge*), kategorisiert der LogAppender das Log Event bevor er es publiziert. In Unterabschnitt 7.2.3 wird der Verhalten des LogAppenders beschrieben, wenn ein LogEvent auftritt. Im Folgenden wird daher nur kurz die Funktionsweise der einzelnen Aufgaben beschrieben.

Log Event Detektierung Um ein Log Event publizieren zu können, muss der LogAppender zunächst detektieren, dass ein zu protokollierendes Ereignis eingetreten ist. Die Realisierung der Detektierung ist dem LogAppender überlassen, in der Regel bieten die Logging Bibliotheken jedoch die Möglichkeit dem entsprechenden Logger den LogAppender hinzuzufügen, sodass dieser automatisch benachrichtigt wird (Observer Muster [GHJV95]). Das Resultat des Detektierungsprozesses ist damit das aufgetretene Log Event.

Log Event Kategorisierung Der LogAppender kategorisiert ein Log Event. Nach *FA6 - Kategorisierung der Log Einträge* sind die zu unterstützenden Kategorien der technische und der fachliche Log Eintrag, welche vom Entwickler a priori festgelegt werden. Abhängig von den protokollierten Daten, kategorisiert der Appender es entsprechend als fachlichen oder technischen Log Eintrag. Das Resultat des Kategorisierungsprozesses ist daher entweder ein fachlicher oder ein technischer Log Eintrag.

Log Event Veröffentlichung Zur Veröffentlichung eines Log Events, kapselt der LogAppender den kategorisierten Log Eintrag in einer LogEventMessage (vgl. Unterabschnitt 5.3.3) und verschickt diese mithilfe des LoggingSenders.

7.1.2 Config Observer

Im vorangegangenen Abschnitt wurden die Komponenten des Logging Agenten zur Überwindung der Loggerheterogenität vorgestellt wurde. In diesem Abschnitt wird eine weitere Komponente des Logging Agenten beschrieben: der Config Observer. Wie in Abschnitt 7.1 angedeutet, dient der Config Observer zur Detektierung einer lokalen Konfigurationsänderung. Lokal bedeutet hierbei, dass die Konfiguration nicht über die Mechanismen der Logging Infrastruktur durchgeführt wurde, d.h. vom Logging Service ausgelöst wurde, sondern durch die Loggingbibliothek spezifischen Schnittstellen (beispielsweise über eine Konfigurationsdatei).

Grundsätzlich existieren zwei Arten der Änderungsdetektierung: die Detektierung durch Beobachtung (aktiv) und die Detektierung durch Benachrichtigung (passiv). Die passive Detektierung setzt voraus, dass die Logging Bibliotheken entsprechend erweitert werden können, um den Config Observer benachrichtigen zu können. Da diese Erweiterbarkeit nicht vorausgesetzt werden kann, verwendet der Config Observer die passive Variante, d.h. die Detektierung erfolgt durch Beobachtung und Vergleich der Konfiguration. Da aus Integrierbarkeitsgründen nicht von einem Zugriff auf die Konfigurationsdatei ausgegangen werden kann, beobachtet der ConfigObserver nicht die Konfigurationsdatei, sondern vergleicht in regelmäßigen Abständen die aktuell angewandten Konfigurationen (verfügbar über die ConfigHandler) mit den im LoggerConfigCache (siehe Unterabschnitt 7.1.2) vorgehaltenen Konfigurationen um Unterschiede zu erkennen.
Detektiert der Config Observer eine Konfigurationsänderung, löst er den entsprechenden Konfigurationsvorgang aus, der in Unterabschnitt 7.2.2 beschrieben ist.

7.1.3 LoggerConfigCache

Die letzte wichtige Komponente eines Logging Agenten ist der LoggerConfigCache. Der LoggerConfigCache repräsentiert einen Schnappschuss der angewandten Loggerkonfigurationen. Dazu verwendet der LoggerConfigCache die gleiche Datenstruktur wie der LoggerContext (vgl. Unterabschnitt 6.1.2). Dieser Schnappschuss wird primär benötigt, damit der Config Observer die aktuellen Konfigurationen mit dem Schnappschuss vergleichen kann, um Änderungen zu detektieren. Wie in Unterabschnitt 7.1.2 geschildert, ist diese Form der aktive Änderungsdetektierung die einzig anwendbare. Durch die regelmäßige Überprüfung des Config Observers spiegelt der LoggerConfigCache den aktuellen Konfigurationstand wider und kann daher zur schnellen Synchronisation der Konfiguration mit Logging Service verwendet werden. Wie in Unterabschnitt 7.2.2 dargestellt wird, bestätigt der Logging Agent eine Remotekonfigurationsänderung nämlich nicht nur mit der entsprechenden Konfiguration, sondern antwortet mit seinem gesamten LoggerConfigCache.

7.2 Dynamische Sicht

Nachdem der vorangegangen Abschnitt den Logging Agent unter statischen Gesichtspunkten und insbesondere das Konzept zur Lösung des Problems der Loggerheterogenität vorgestellt hat, beschreibt dieser Abschnitt das Laufzeitverhalten des Logging Agenten, sodass der Logging Agent vollständig dargestellt ist.

Das grundsätzliche Verhalten des Logging Agenten, nämlich die Detektierung einer Konfigurationsänderung und die Veröffentlichung eines LogEvents, hat der Architekturentwurf (Kapitel 5) im Rahmen der Kollaborationsbeschreibung der Komponenten der Infrastruktur (Unterabschnitt 5.3.2) aus Verständlichkeitsgründen bereits grob beschrieben. In diesem Abschnitt wird dieses Verhalten detaillierter beschrieben und insbesondere Ausnahme und Fehlerfälle beachtet.

Grundsätzlich sind zur Modellierung des Laufzeitverhalten des Logging Agenten vier Szenarien relevant, die im Rahmen der Agententheorie mit der Exploration seiner Umwelt, der Reaktion auf Umweltveränderung (selbstständig detektiert oder reaktiv), sowie die Reaktion auf externe Ereignisse (Erhalt eines Befehls) verglichen werden können. Konkret handelt es sich hierbei um das Verhalten beim Starten / Deployment des Logging Agenten, die Detektierung einer Konfigurationsänderung, das Auftreten eines Log Events und den Empfang einer LoggerConfigSynchronisationMessage (vgl. Unterabschnitt 5.3.3). Im Folgenden werden diese Szenarien betrachtet.

7.2.1 Starten / Deployment

Dieser Abschnitt beschreibt das Laufzeitverhalten des Logging Agenten beim (Neu-) Starten bzw. Deployment seines adaptierten Systems. Das Ziel dieses Vorganges ist Einleitung eines Synchronisationsporzesses der Loggerkonfigurationen mit dem Logging Service, sodass die lokal angewandten Loggerkonfiguration letztendlich den Logging Service seitigen entsprechen.

Wie in Abbildung 7.3 modelliert, entdeckt der Logging Agent dazu mithilfe der registrierten Logging Adapter (genauer LoggerDiscoverer) zunächst alle Logger des adaptierten Service. Mithilfe des ConfigHandlers des entsprechenden Loggertyps wird jeder Logger mit seiner RequiredConfig (vgl. 7.1.1) konfiguriert. Da die RequiredConfig in der Regel einer Teilmenge der angewandten Loggerkonfiguration entspricht, liest der Logging Agent wieder mithilfe des entsprechenden ConfigHandlers die angewandte Konfiguration aus. Tritt während des Konfigurations- oder Auslesevorganges ein Fehler auf, typischerweise weil der Loggerzugriff fehlschlug, überspringt der Logging Agent diesen speziellen Logger.
Die Konfigurationen der erfolgreich konfigurierten Logger hält der Logging Agent anschließend in seinem LoggerConfigCache vorrätig um eine lokale Konfigurationsänderung detektieren zu können (vgl. Unterabschnitt 7.1.2). Anschließend publiziert der Logging Agent den Inhalt des LoggerConfigCaches in Form einer

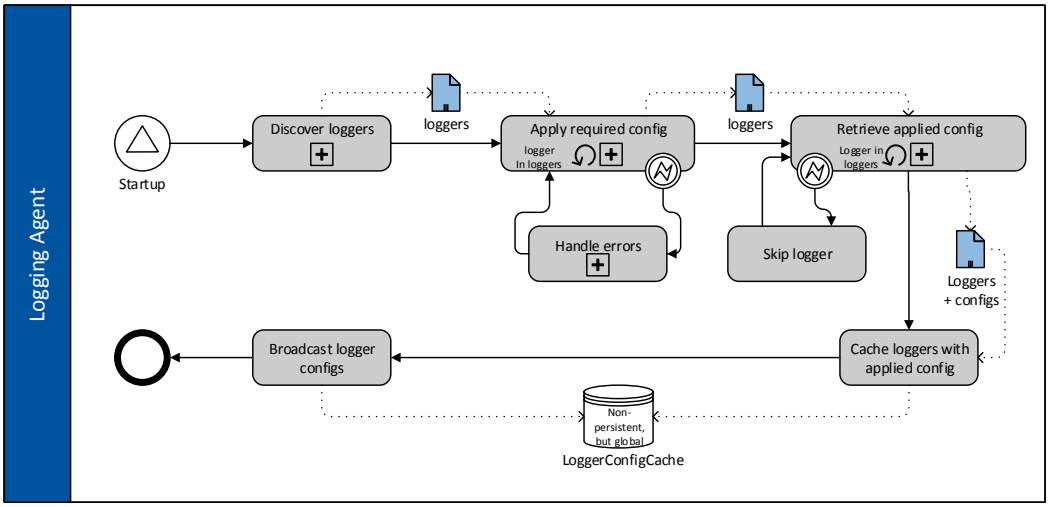

Abbildung 7.3: Logging Agent - Verhalten beim Starten

LoggerConfigSynchronisationMessage (Unterabschnitt 5.3.3) über den LoggingSender auf dem Logging Bus um einen Konfigurationssynchronisationsvorgang einzuleiten, d.h. dem Logging Service die aktuellen Konfigurationen mitzuteilen und potentielle Konfigurationsunterschiede zu beheben.

7.2.2 Konfigurationsänderung

Nachdem der vorherige Abschnitt das Startverhalten des Logging Agenten beschrieben hat und damit die Exploration seiner Umwelt beschrieben wurde, modelliert dieser Abschnitt, wie der Logging Agent Veränderungen seiner Umwelt detektiert bzw. wie er auf Änderungen reagiert. Konkret betrachtet dieser Abschnitt das Szenario der lokalen Konfigurationsänderung.
Wie in Unterabschnitt 7.1.2 angedeutet, detektiert der Logging Agent mithilfe des ConfigObserver eine Konfigurationsänderung über regelmäßige Vergleiche der aktuelle angewandten Konfigurationen mit den im LoggerConfigCache vorrätig gehaltenen Loggerkonfiguration. In Abbildung 7.4 ist dieser Vorgang dargestellt.

Für jeden Logger des LoggerConfigCache liest der ConfigObserver über den entsprechenden ConfigHandler die aktuelle Konfiguration aus und vergleicht diese mit der vorgehaltenen Konfiguration im LoggerConfigCache. Anschließend ersetzt der ConfigObserver alle unterschiedlichen Konfigurationen im LoggerConfigCache durch die angewandten Konfigurationen. Tritt während des Auslesens der angewandten Konfigurationen ein Fehler auf, markiert der ConfigObserver die entsprechende Konfiguration im LoggerConfigCache als nicht angewandt. Sollten Änderungen am LoggerConfigCache stattgefunden haben, signalisiert der ConfigObserver, dass Änderungen stattgefunden haben. Daraufhin veröffentlicht der Logging Agent über den

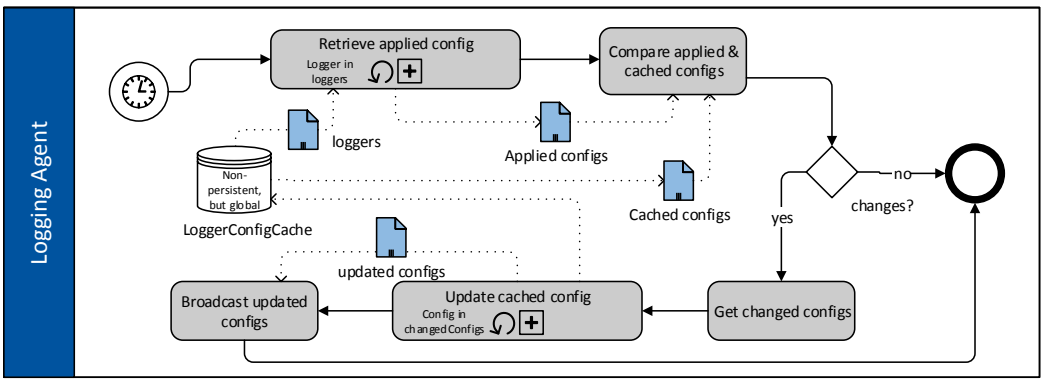

Abbildung 7.4: Logging Agent - Detektierung einer Konfigurationänderung

LoggingSender diese geänderten Konfigurationen in einer LoggerConfigSynchronisation Nachricht. Wie in 6.2.3 beschrieben, wird der Logging Service mithilfe des Merge Service entscheiden, welche Konfigurationen zu verwenden sind und mit einer LoggerConfigSynchronisation Nachricht antworten. Die Reaktion des Logging Agenten auf den Empfang einer LoggerConfigSynchronisationMessage wird in Unterabschnitt 7.2.4 beschrieben.

7.2.3 Log Event

Typischerweise tritt die im vorangegangen Abschnitt beschrieben Konfigurationsänderung selten auf. Viel häufiger wird ein Log Event auftreten. Dieser Abschnitt widmet sich diesem Szenario.
Durch die initiale Konfiguration jedes Loggers mit einem LogAppender (Anwendung der RequiredConfig beim Starten) und der Realisierung eines LogAppenders in Form des Observer Entwicklungsmusters, wird der LogAppender beim Auftreten eines Log Events direkt benachrichtigt. Der LogAppender kategorisiert dieses Log Event anhand bestimmter Eigenschaften (z.B. ob das EOM protokolliert wurde) und veröffentlicht über den LoggingSender eine LogEvent Nachricht auf dem Logging Bus.

7.2.4 Empfang LoggerConfigSynchronisationMessage

Damit im Rahmen der Logging Infrastruktur eine Remotekonfiguration durchgeführt werden kann, muss der Logging Agent auf Nachrichten reagieren können. Konkret werden Konfigurationsänderungen über eine LoggerConfigSynchronisationMessage publiziert (vgl. Unterabschnitt 6.2.2). In diesem Abschnitt wird dafür das entsprechende Verhalten des Logging Agenten in Reaktion auf den Empfang einer LoggerConfigSynchronisation Nachricht modelliert.

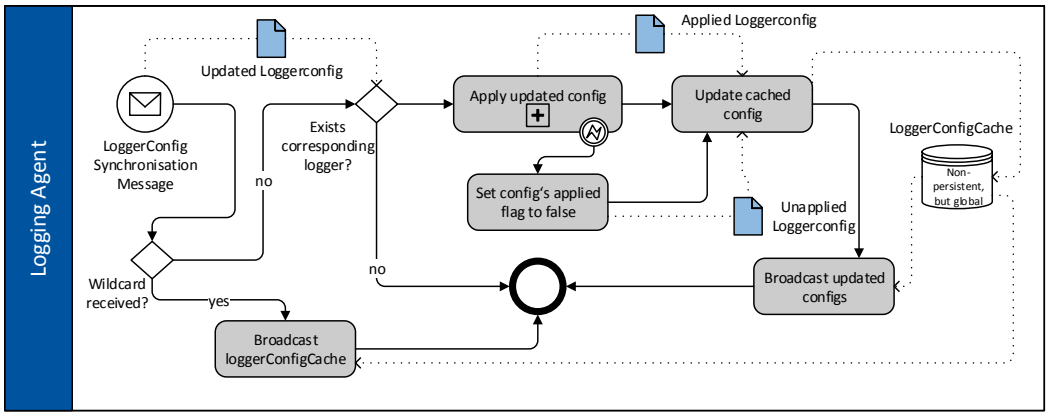

Abbildung 7.5: Logging Agent - Empfang einer LoggerConfigSynchronisationMessage

In Abbildung 7.5 ist die Reaktion des Logging Agenten auf den Empfang einer LoggerConfigSynchronisation Nachricht veranschaulicht. Zur Vereinfachung der Modellierung wurde angenommen, dass die empfange Nachricht nur eine Loggerkonfiguration beinhaltet. Würde die Nachricht mehrere Konfigurationen enthalten, entspräche das Verhalten grob dargestellt einer mehrfachen Ausführung des in Abbildung 7.5 Prozesses.
In einem ersten Schritt überprüft der Logging Agent, ob es sich bei der empfangenen Nachricht um eine Wildcard (vgl. Unterabschnitt 6.2.1) handelt, also der Aufforderung entspricht, die aktuell angewandten Konfigurationen zu verschicken. Da der LoggerConfigCache in regelmäßigen Abständen durch den ConfigObserver aktualisiert wird (Unterabschnitt 7.2.2), verschickt der Logging Agent in Reaktion auf den Erhalt einer Wildcard Nachricht aus Geschwindigkeitsgründen seinen LoggerConfigCache (anstatt die angewandten Konfiguration erneut auszulesen) in einer LoggerConfigSynchronisation Nachricht.

Handelt es sich um keine Wildcard, sondern um eine (geänderte) Loggerkonfiguration, prüft der Logging Agent zunächst mithilfe des entsprechenden ConfigHandler, ob der Logger existiert und verfügbar ist. Sollte dies nicht der Fall sein, ist ein Konfigurationsvorgang überflüssig und der Logging Agent beendet den Prozess. Ist eine Konfiguration des Loggers möglich, wendet der Logging Agent über den ConfigHandler die erhaltene Konfiguration an, aktualisiert den LoggerConfigCache entsprechend und broadcastet eine LoggerConfigSynchronisation Nachricht mit der als angewandt (applied) markierten Konfiguration, um den Logging Service die Konfigurationsänderung zu bestätigen (vgl. Anfrage/Bestätigung-Konfigurationsprotokoll in 5.3.2). Tritt während der Konfigurationsanwendung ein Fehler auf, publiziert der Logging Agent die Loggerkonfiguration entsprechend mit der nicht-angewandt Markierung.

7.3 Beispielintegration

Nachdem in den beiden vorherigen Abschnitten der Logging Agent sowohl unter statischen, als auch dynamischen Gesichtspunkten modelliert wurde, wird in diesem Abschnitt die Integration des Logging Agenten in ein externes System verdeutlicht und am Beispiel der bekannten Java Logging Bibliothek Log4J veranschaulicht, wie im Rahmen der realisierten Logging Infrastruktur ein technischer und ein fachlicher Log Eintrag generiert werden können.

Die Beschreibung der Logging Agent Integration basiert dabei auf der am Lehr- und Forschungsgebiet 3 Informatik mit Java EE realisierten EMI. Wie in Abschnitt 7.1 erwähnt, erfolgt die Integration bzw. Konfiguration über den LoggingAgentIntegrator. Da häufig die Standarteinstellung ausreichend sind, ermöglicht die in dieser Arbeit entstandene Realisierung des Logging Agenten die Adaptierung eines externen Systemes durch das bloße Deployment im gleichen Kontext, wie Unterabschnitt 7.3.1 beschreibt. Um das Verhalten des Logging Agenten steuern zu können, bietet die Realisierung außerdem eine flexible Möglichkeit zur Integration. Diese Möglichkeit ist in Unterabschnitt 7.3.2 beschrieben. Anschließend verdeutlicht Unterabschnitt 7.3.3, wie im Rahmen der Logging Infrastruktur Log-Einträge generiert werden können.

7.3.1 Standart Integration

In diesem Abschnitt wird die Integration des Logging Agenten in ein externes System mit den Standarteinstellungen beschrieben. Da die Realisierung des Logging Agenten den sogenannten SimpleIntegrator anbietet, muss lediglich die in Quellcode 7.1 aufgeführte Komponente dem externen System als Abhängigkeit hinzugefügt werden (beispielsweise über Maven).
Benötigte Informationen zur Identifikation des Logging Agenten, die abhängig vom adaptierten Service sind, liest der SimpleIntegrator aus einer Konfigurationsdatei (EMI.properties), die nach der EMI Referenzarchitektur für jedem EMI-Service vorhanden ist. In der aktuellen Version (1.1.6) ist der SimpleIntegrator mit java.util.logging und Log4J kompatibel.

```
1 <!-- pom.xml -->
2 <dependency>
3     <groupId>de.rwth.swc.emi.logging.framework</groupId>
4     <artifactId>de.rwth.swc.emi.logging.framework.loggingagent.simple.
5                 integrator</artifactId>
6     <type>ejb</type>
7 </dependency>
```

Quelltext 7.1: pom.xml - SimpleIntegrator Dependency

7.3.2 Flexible Integration

Neben der im vorherigen Abschnitt beschriebenen einfachen Integration des Logging Agenten mit den Standarteinstellungen, bietet die Realisierung des Logging Agenten auch eine flexible Integration an. Flexibel bedeutet dabei, dass sowohl die unterstützten Logging Bibliotheken als auch das Verhalten des Logging Agenten konfiguriert werden kann, wie dieser Abschnitt beschreibt.

Selektive Logging Adapter Wahl

In diesem Abschnitt wird beschrieben, wie die unterstützten Logging Bibliotheken eines Logging Agenten flexibel konfiguriert werden können. Da jeder Logging Adapter ein Plugin darstellt und sich selbstständig registriert (vgl. Abschnitt 7.1), lässt sich selektiv über die definierten Abhängigkeiten steuern, welcher Adapter zusammen mit dem Logging Agenten gebündelt werden soll. Für Maven ist diese Selektion beispielhaft in Quellcode 7.2 dargestellt. Der entsprechende Logging Agent würde folglich Javalogging und Log4J unterstützen.

```
1  <!-- pom.xml -->
2  <dependency>
3      <groupId>de.rwth.swc.emi.logging.framework</groupId>
4      <artifactId>de.rwth.swc.emi.logging.framework.loggingagent.adapter.
5                  javalogging</artifactId>
6      <type>ejb</type>
7  </dependency>
8
9  <dependency>
10     <groupId>de.rwth.swc.emi.logging.framework</groupId>
11     <artifactId>de.rwth.swc.emi.logging.framework.loggingagent.adapter.
12                 log4j</artifactId>
13     <type>ejb</type>
14 </dependency>
```

Quelltext 7.2: pom.xml - Selektive Logging Adapter Auswahl

BaseLoggingIntegrator

Nachdem im vorherigen Abschnitt beschrieben wurde, wie die unterstützten Logging Bibliotheken pro Logging Agent selektiv ausgewählt werden können, stellt dieser Abschnitt die zweite Dimension der Flexibilität des Logging Agenten dar, nämlich wie das Verhalten des Logging Agent gesteuert werden kann.

Die Steuerung erfolgt dabei über den LoggingAgentIntegrator Erweiterungspunkt (Hotspot) (vgl. Abschnitt 7.1). Zur Konfiguration wird also eine Implementierung des LoggingAgentIntegrator Schnittstelle benötigt. Da sich nahezu die gesamte Funktionalität des Logging Agenten über diese Schnittstellenimplementierung steuern lässt, kapselt die Realisierung des Logging Agenten in Form des BaseLoggingAgent

bereits die wichtigste Funktionalität, sodass sich das Verhalten selektiv steuern lässt.

Das Integrationsbeispiel des Logging Agenten in das River Projekt ([Cha13]) einer EMI in Quellcode 7.3 zeigt, dass sich beispielsweise die Identifikationsargumente des Logging Agenten anpassen lassen, der Entdeckungsvorgang der Logger eine Filterung über einen regulären Ausdruck ermöglicht, oder bestimmte Hooks (beim Starten des Service und nachdem alle Logging Adapter registriert sind) verwendet werden können, sowie das Verhalten auf den Empfang Logging Nachrichten konfigurierbar ist.

```
1
2  package de.rwth.swc.river.emi.integration;
3
4  import javax.ejb.EJB;
5  import javax.ejb.Stateless;
6
7  import de.rwth.swc.emi.framework.base.LocalServiceProperties;
8  import de.rwth.swc.emi.logging.framework.domain.logger.ServiceType;
9  import de.rwth.swc.emi.logging.framework.loggingagent.
10     adapter.BaseLoggingIntegrator;
11 import de.rwth.swc.emi.logging.framework.loggingagent.
12     adapter.LoggingAgentIntegrator;
13
14 @Stateless
15 public class RiverCoreLoggingAgent extends BaseLoggingAgent
16     implements LoggingAgentIntegrator {
17
18     private @EJB
19     LocalServiceProperties properties;
20
21     @Override
22     public ServiceType getServiceType() {
23         return properties.getServiceTyp();
24     }
25
26     @Override
27     public String getLoggerRepositoryName() {
28         return properties.getServiceName();
29     }
30
31     @Override
32     public String getNodeName() {
33         return properties.getServiceNodeName();
34     }
35
36     @Override
37     // only discover logger with matching names (regex-based)
38     public String getLoggerNameDiscoverRegex() {
39         return "de.rwth.swc.river.*";
40     }
```

```
41
42      @Override
43      // EJB @Startup hook
44      public void onDeployment() {
45          //for demonstration purposes
46          super.onDeployment();
47      }
48
49      @Override
50      // hook when every logging adapter is registered
51      public void onLoggerAdapterRegistrationComplete() {
52          //for demonstration purposes
53          super.onLoggerAdapterRegistrationComplete();
54      }
55
56      @Override
57      public void onMessage(LoggingMessage msg) {
58          //for demonstration purposes
59          super.onMessage(msg);
60      }
61  }
```

Quelltext 7.3: RiverCoreLoggingAgent.java

7.3.3 Logerzeugung

Nachdem in den beiden vorherigen Abschnitten die Möglichkeiten zur Verhaltensbeeinflussung des Logging Agenten vorgestellt und das Verhalten des Logging Agenten in Abschnitt 7.2 erläutert wurde, sollen in diesem Abschnitt die Auswirkungen der gewählten Realisierung in Hinblick auf die Logerzeugung beschrieben werden.

Da der Logging Agent die Logger direkt zur Veröffentlichung ihrer Log Events mithilfe des Log Appenders konfiguriert und der Log Appender das Log Event entsprechend kategorisiert (vgl. Unterabschnitt 7.2.3), können bestehende Log Statements verwendet werden.
In Quellcode 7.4 sind für die Java Logging Bibliothek Log4J Beispielstatements aufgeführt. Der realisierte Log4J Appender kategorisiert die Log Events anhand der protokollierten Informationen. Wird ein EOM protokolliert, handelt es sich um einen fachlichen Log, ansonsten wird das Log Event als ein technisches kategorisiert.

```
1  import org.apache.log4j.Logger;
2  private Logger logger = Logger.getLogger("de.rwth.swc.dummylogger");
3
4  //Technical Log Events
5  logger.info("This results in a technical log");
6  logger.warn(new TechnicalLog("This also results in a technical log"));
7
```

```
8  //Functional Log Event
9  logger.debug(new FunctionalLog("I am a functional log", "EOM1"));
10
11
12 }
```

Quelltext 7.4: Beispiellogstatement

8 Visualisierungs- & Konfigurationswerkzeug

It's not a bug; it's an undocumented feature!

Author unknown

Inhalt

In Abschnitt 5 wurde die Architektur der Logging Infrastruktur modelliert und die einzelnen Komponenten eingeführt. Die Komponenten der Logging Infrastruktur sind dabei der Logging Agent zur Adaptierung externer Systeme, der Logging Service zur Datenverwaltung, der Merge Service zur Konfliktlösung von Logger Konfigurationen und das Visualisierungs- & Konfigurationswerkzeug zur Darstellung der Daten. Während der Logging Servce zusammen mit dem Merge Service in Kapitel 6 und der Logging Agent in Kapitel 7 vorgestellt wurde, soll in diesem Kapitel die letzte Komponente, nämlich das Visualisierungs- & Konfigurationswerkzeug vorgestellt werden, welches direkt durch *FA7 - Visualisierungs- & Konfigurationswerkzeug* motiviert wird.

Insgesamt muss das Visualisierungs- & Konfigurationswerkzeug alle Logger bzw. Loggerkonfigurationen und die protokollierten Log Events darstellen und insbesondere eine Remotekonfiguration (vgl. 5.3.2) auslösen können.

Neben diesen funktionalen Anforderungen wurden in Abschnitt 3.4 (p. 14) folgende nicht-funktionale Anforderungen an das Visualisierungs- & Konfigurationswerkzeug definiert:

NFA-T-1 - Usability (Bedienbarkeit) Das Visualisierungs- & Konfigurationswerkzeug soll alle Funktionen zur effizienten Aufgabenerfüllung bieten, konsistent und zur schnellen Informationsbefriedigung übersichtlich sein.

NFA-T-2 - Aktualität Die vom Visualisierungs- & Konfigurationswerkzeug dargestellten Informationen (Loggerkonfigurationen & Log-Einträge) sollen immer aktuell sein.

Basierend auf diesen Anforderungen wurde prototypisch eine HTML5 und Javascript basierte Webapplikation realisiert. Durch die weitverbreitete Unterstützung dieser Art von Anwendung, verletzt das Werkzeug damit nicht *FA2 - Integrierbarkeit* der gesamten

Logging Infrastruktur und entspricht den aktuellen Technikstandards.

Wie in Kapitel 5 dargestellt, dient der Logging Service dabei als Datenschnittstelle des Visualisierungs- & Konfigurationswerkzeug. Zur loosen Kopplung des Werkzeuges mit dem Logging Service wird eine RESTful Schnittstelle verwendet, die wegen *NFA-T-2 - Aktualität* um eine Websocket Schnittstelle ergänzt wird, damit der Logging Service das Visualisierungs- & Konfigurationswerkzeug bei Änderungen benachrichtigen kann. Für eine detaillierte Kollaborationsdarstellung sei auf Unterabschnitt 5.3.2 verwiesen.

Anhand der von Ludewig und Lichter beschriebenen Entwicklungsrichtung *outside-in* [LL10] wurden in einem ersten Iterationsschritt grafische Papierprototypen entwickelt, die im nachfolgenden Abschnitt vorgestellt werden. Die *outside-in* Entwicklungsrichtung wurde dabei aufgrund der bereits definierten Schnittstellen verwendet.
Ausgehend von den Papierprototypen wurde anschließend das Visualisierungs- & Konfigurationswerkzeug realisiert, wie es in Abschnitt 8.2 beschrieben wird.

8.1 Grafische Papierprototypen

Dieser Abschnitt veranschaulicht die Funktionalitäten des Visualisierungs- & Konfigurationswerkzeuges anhand von grafischen Papierprototypen, die in einem ersten Iterationschritt entworfen wurden.
Anhand der Anforderungen wurden drei verschiedene Ansichten identifiziert, die das Werkzeug bereitstellen muss. Zum einen die Konfigurationsseite der Logger und zum anderen die Log Darstellung, die sich als Konsequenz aus *FA6 - Kategorisierung der Log Einträge* und aufgrund *NFA-T-1 - Usability (Bedienbarkeit)* in die Darstellung funktionaler und technischer Logs aufspaltet.
Im Folgenden werden diese Ansichten beschrieben.

8.1.1 Konfigurationsansicht

Die Konfigurationansicht ermöglicht dem Anwender sowohl die Betrachtung der Loggerkonfigurationen als auch die Durchführung einer Remotekonfiguration (vgl. 5.3.2) der Logger aller angeschlossenen Logging Agenten.
Wegen NFA-T-1 - Usability (Bedienbarkeit) soll eine übersichtliche Darstellung erfolgen. Dazu erfolgt im linken Bereich eine grobe Kategorisierung der einzelnen Logging Agenten in die verschiedenen Service Typen einer EMI. Pro Logging Agent werden die einzelnen Logger baumartig dargestellt. Die Auswahl eines Loggers links ermöglicht auf der rechten Bildschirmseite die Betrachtung und Veränderung der im Logging Service gespeicherten Konfiguration. Zusätzlich zu der konkreten Konfiguration stellt ein Indikator dar, ob die angezeigte Konfiguration der angewandten entspricht (vgl. Angewandt-Markierung 6.1.2).
Der Papierprototyp ist in Abbildung 8.1 dargestellt.

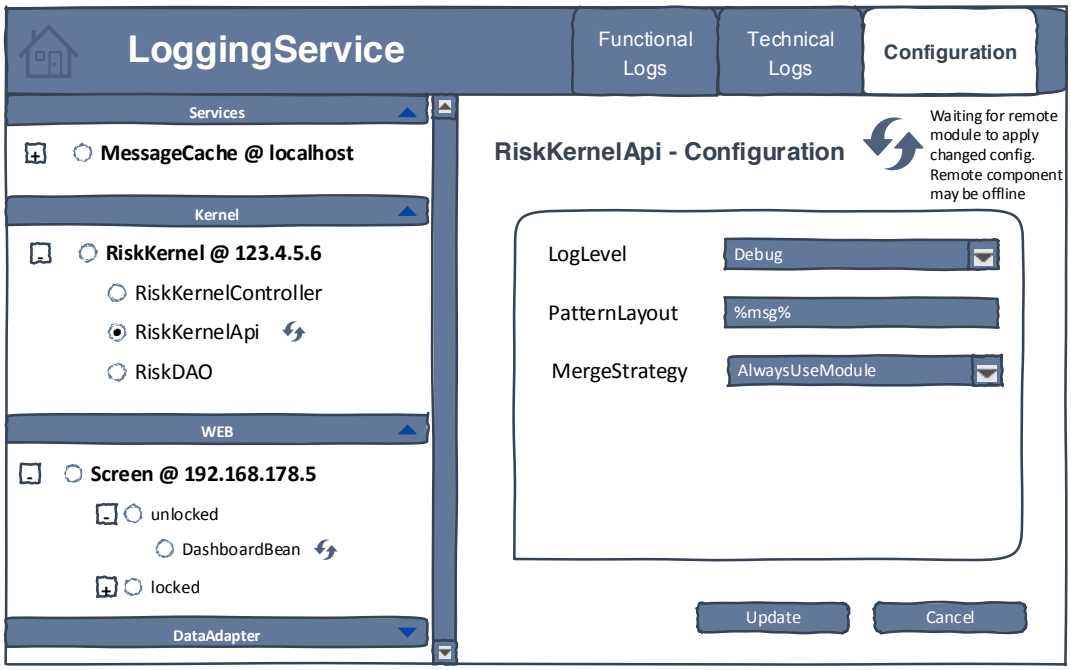

Abbildung 8.1: GUI Prototyp - Konfigurationsansicht

8.1.2 Log Darstellung

Nachdem im vorigen Abschnitt die Konfigurationsansicht des Papierprototypen dargestellt wurde, beschreibt dieser Abschnitt die Log Darstellung des Visualisierungs- & Konfigurationswerkzeuges. Dabei unterteilt sich diese Ansicht in die Darstellung technischer und fachlicher Log-Einträge. Da der Aufbau beider Ansichten identisch ist, werden beide zunächst gemeinsam beschrieben.

Zur Visualisierung der Log Einträge unterteilt sich die Ansicht in die einzelnen Logger auf der linken und die konkrete Darstellung der Log-Einträge auf der rechten Seite.
Die Logger auf der linken Seite werden dabei abhängig von der konkreten Log Ansicht kategorisiert. Während für fachliche Logs eine Kategorisierung nach EOM, also der vermessen Einheit innerhalb der EMI, sinnvoll ist, werden für die technische Log Darstellung die Logger bzw. Logging Agenten nach EMI Service Typ kategorisiert.
Die angezeigten Log-Einträge lassen sich durch eine Auswahl der Logger einschränken, da nur Log-Einträge der ausgewählten Logger angezeigt werden. Die Darstellung der Log-Einträge erfolgt tabellenweise, wobei die dargestellten Informationen abhängig von der Art des Log-Eintrages (fachlich vs. technisch) sind, wie in Abbildung 8.2 bzw. Abbildung 8.3 zu erkennen ist. Um Übersichtlichkeit zu gewährleisten, werden zum einen nur begrenzt viele Log-Einträge auf einmal angezeigt und zum anderen nur die wesentlichen Felder dieser Log-Einträge dargestellt. Erst durch Auswählen des

entsprechenden Log-Eintrages werden alle Felder dieses Eintrages angezeigt.
Unabhängig von der konkreten Ausprägung des Log-Seite, bietet das Werkzeug außerdem Komfortfunktionen wie eine Suche um *NFA-T-1 - Usability (Bedienbarkeit)* zu erleichtern und Log-Managementfunktionen wie die Löschmöglichkeit von Log-Einträgen.

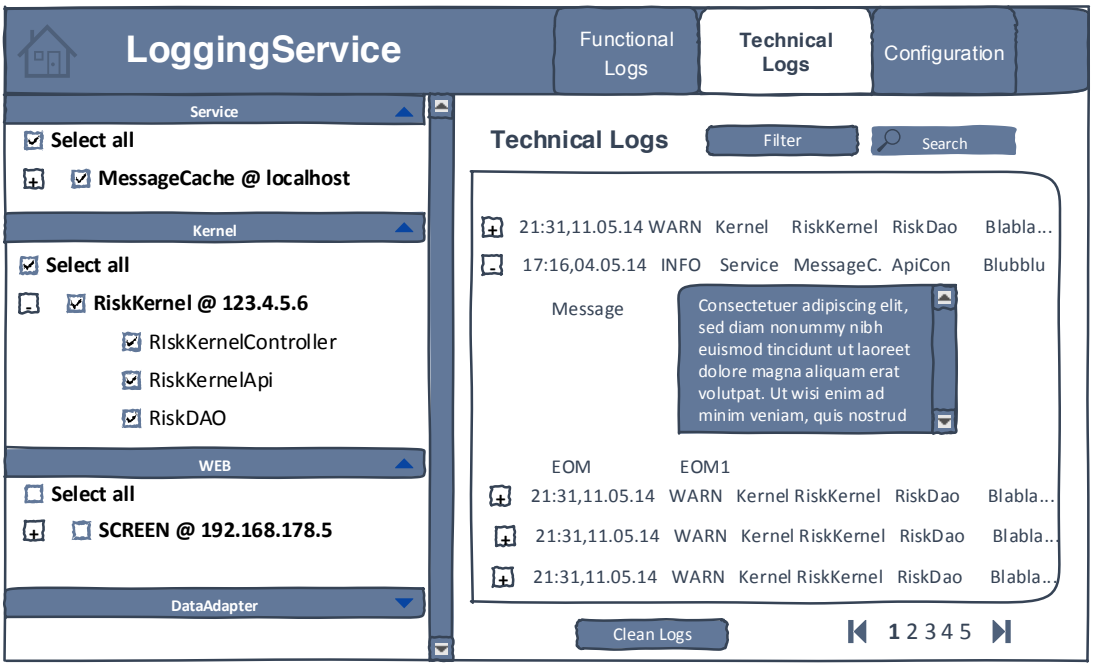

Abbildung 8.2: GUI Prototyp - Technische Log Ansicht

8.2 Realisierung

Dieser Abschnitt beschreibt die Realisierung des Visualisierungs- & Konfigurationswerkzeug. Ausgehend von den grafischen Prototypen, die im voherigen Abschnitt beschrieben wurden (vgl. Abschnitt 8.1) und basierend auf den Anforderungen wurde das Visualisierungs- & Konfigurationswerkzeug umgesetzt.

Wie in Kapitel 5 konzeptioniert, sind Werkzeug und das Backend, der Logging Service, loose über eine RESTful Api gekoppelt. Aufgrund *NFA-T-2 - Aktualität* interagiert das Werkzeug zusätzlich über eine Websocket-Schnittstelle, um über Datenveränderung vom Logging Service informiert zu werden. Die Datenadaptierung erfolgt hierbei ähnlich des **invoke-push** Datenadaptierungsmuster [LVS14], d.h. der Logging Service benachrichtigt das Werkzeug im Falle einer Aktualisierung und das Werkzeug fragt die aktualisierten Daten ab.

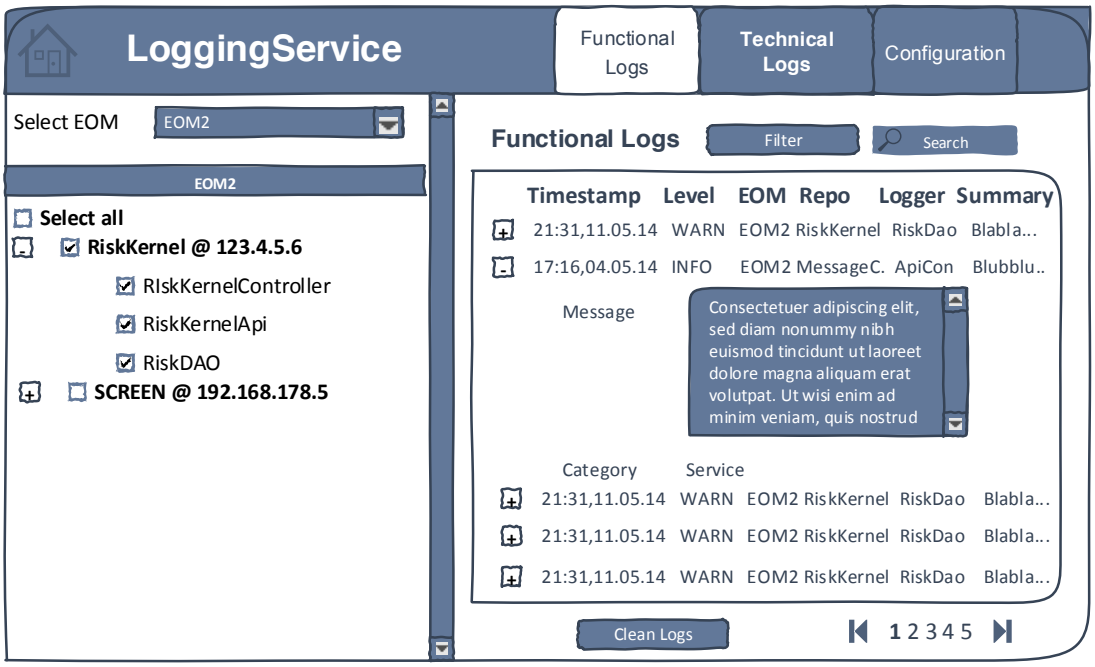

Abbildung 8.3: GUI Prototyp - Funktionale Log Ansicht

Basierend auf den unterschiedlichen Sichten des Werkzeuges, der Konfigurationsansicht und der technischen, sowie fachlichen Log-Darstellung wird das realisierte Visualisierungs- & Konfigurationswerkzeug im Folgenden vorgestellt und anschließend die verwendeten Technologien erläutert.

8.2.1 Konfigurationsansicht

An dieser Stelle wird die Konfigurationsansicht des realisieren Visualisierungs- & Konfigurationswerkzeug vorgestellt. Basierend auf den grafischen Papierprototypen werden auf der linken Seite (vgl. Unterabschnitt 8.1.1) sortiert nach EMI Servicetyp des adaptierten Systemes des Logging Agenten die einzelnen Logger aufgeführt. Über einen farblichen Indikator jedes Logger wird der Konfigurationsstatus abgebildet.
Grün, als Bestätigungsfarbe, signalisiert, dass alle Logger Konfigurationen dieses EMI Service angewandt sind. Die rote Warnfarbe weist den Anwender auf ein Synchronisationsproblem bzw. eine oder mehrere nicht-angewandte Konfigurationen hin.

Da in einer Produktivumgebung aufgrund der Mikroservicestruktur einer EMI viele adaptierte Services typisch sind, lassen sich zur Übersichtlichkeit sowohl die einzelnen Logger Darstellungen eines EMI Services, als auch die einzelnen EMI Servicetypen auf- und zuklappen. Eine zugeklappte Logger Darstellung, wie beispielsweise der

MergeService in Abbildung 8.4, stellt nur noch die wichtigsten Informationen, nämlich den Namen und den farblichen Angewandtindikator der Konfigurationen dar.
Ein aufgeklappter EMI Service stellt die zugehörige Loggerhierachie dar und bietet Komfortfunktionen zum Auf- und Zuklappen der Loggerhierachie, sowie zum Falten der Hierachie, was in 8.2.1 näher beschrieben wird.

Selektiert der Anwender einen Logger, wird auf der rechten Seite dessen Konfiguration dargestellt. Hauptsächlich besteht diese Ansicht aus einer Tabartigen-Darstellung der einzelnen Konfigurationsvariabilitäten (siehe 6.1.2). Anhand des Variabilitätsmodells jeder Konfigurationsvariablität wird der Tabinhalt und die Konfigurationsmöglichkeiten vollkommen dynamisch erzeugt, wie in 8.2.1 beschrieben wird. Verändert der Anwender die Konfiguration, wird über den Speicherbutton der Aktualisierungsvorgang gestartet. In der oberen rechten Ecke von Abbildung 8.4 werden dem Anwender die über den Websocket empfangenen Benachrichtigungen visualisiert und in einem Benachrichtungscenter verwaltet.

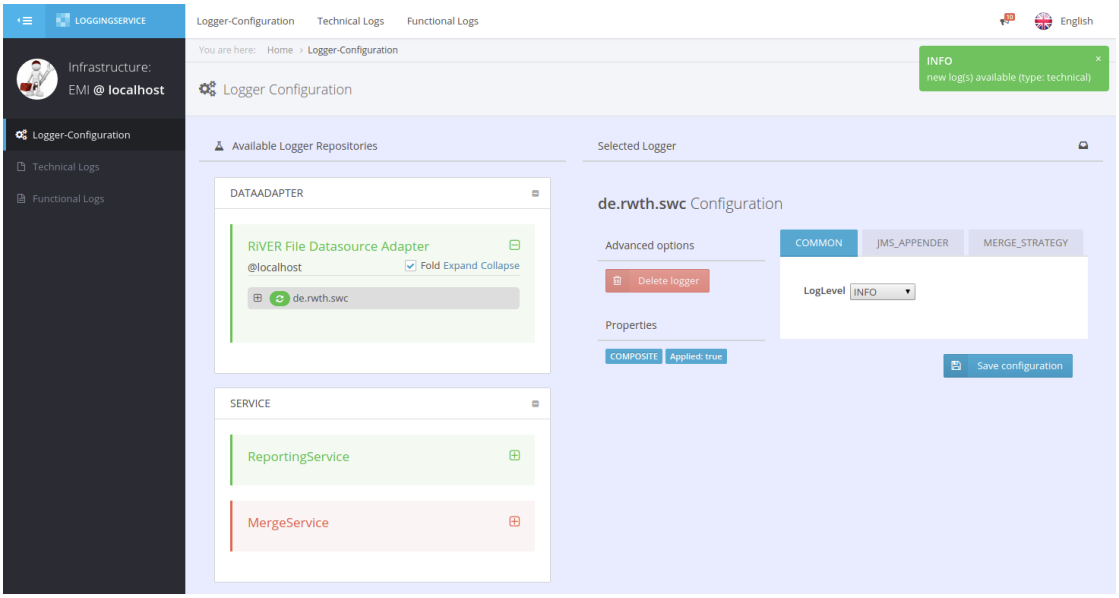

Abbildung 8.4: Werkzeug - Konfigurationsansicht

Um diese Ausführungen zu verdeutlichen, werden in den beiden folgenden Unterabschnitten, die Loggerhiearchie eines EMI Service und die konkrete Konfigurationsansicht genauer beschrieben.

EMI Service Logger Darstellung

In diesem Abschnitt wird die Darstellung der Logger eines adaptierten Service dargestellt. Da die Logger eines Service typischerweise hierarisch angeordnet sind, unterstützt die Darstellung ebenfalls eine Hierarchie.

In Abbildung 8.5 sind die Logger des MergeService veranschaulicht. Wie zu erkennen ist, existiert neben dem globalen Synchronisationsindikator für den MergeService pro Logger ein Indikator, ob die entsprechende Konfiguration angewandt ist. Dieser Indikator wird ausgehend von den Blättern des Hierachiebaumes bestimmt und pro Hierachieebene aus den Zustände der Kinder berechnet.

Da eine Hierachie viele Hierachieeben umfassen kann, die genau ein Kind beinhalten, lässt sich wegen *NFA-T-1 - Usability (Bedienbarkeit)* jede Hierachieeben auf- und zuklappen, sowie zusammenfalten. Der Zusammenfaltvorgang (fold) ist in Abbildung 8.5 veranschaulicht. Er fasst Hierachieebenen, die genau ein Kind haben, zu einer gemeinsamen Hierachieebene zusammen.

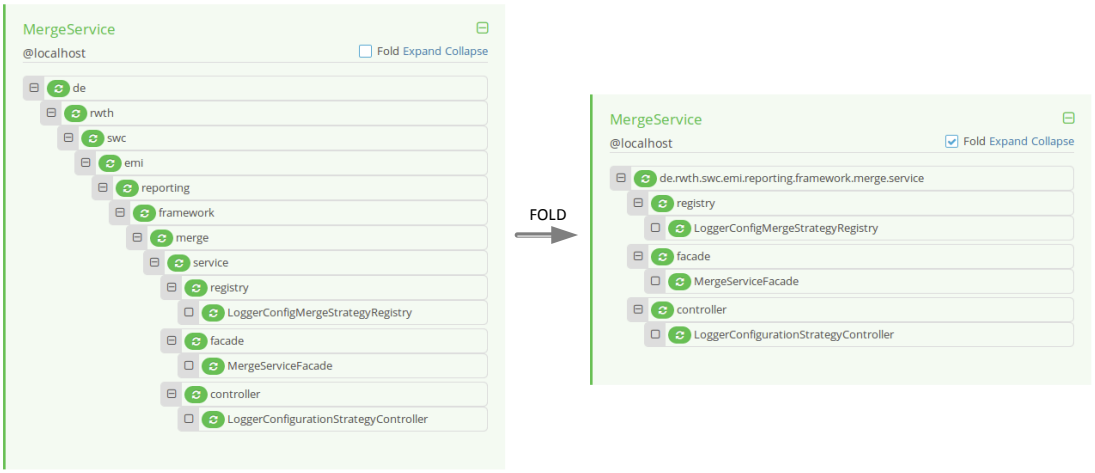

Abbildung 8.5: Werkzeug - Konfigurationsansicht - Loggerhierachie

Loggerkonfiguration

Selektiert der Anwender einen Logger der Loggerdarstellung (siehe 8.2.1), so wird auf der rechten Seite die Loggerkonfiguration dargestellt, die in diesem Abschnitt beschrieben wird.

Die Darstellung der Loggerkonfiguration (Abbildung 8.6) umfasst, neben einer Auflistung der Eigenschaften des Logger und der Möglichkeit den Logger aus dem entsprechenden LoggerContext (siehe Unterabschnitt 6.1.2) zu löschen, eine tabartige Darstellung der Konfigurationsvariabilitäten der Loggerkonfiguration (siehe 6.1.2). Pro Tabinhalt werden die Variabilitätspunkte dieser Konfigurationsvariabilität visualisiert. Die Visualisierung ist dabei dynamisch: Ein geschlossener Variabilitätspunkt resultiert in einem Dropdown, welches die Auswahl der Variabilitäten des Variabilitätspunkt ermöglicht. Die Darstellung eines offenen Variabilitätspunkt ermöglicht eine

freie, textuelle Konfiguration des Variabilitätspunktes. Durch das Auslösen eines Speichervorganges wird der Konfigurationsvorgang des Logging Service gestartet (Unterabschnitt 6.2.2)

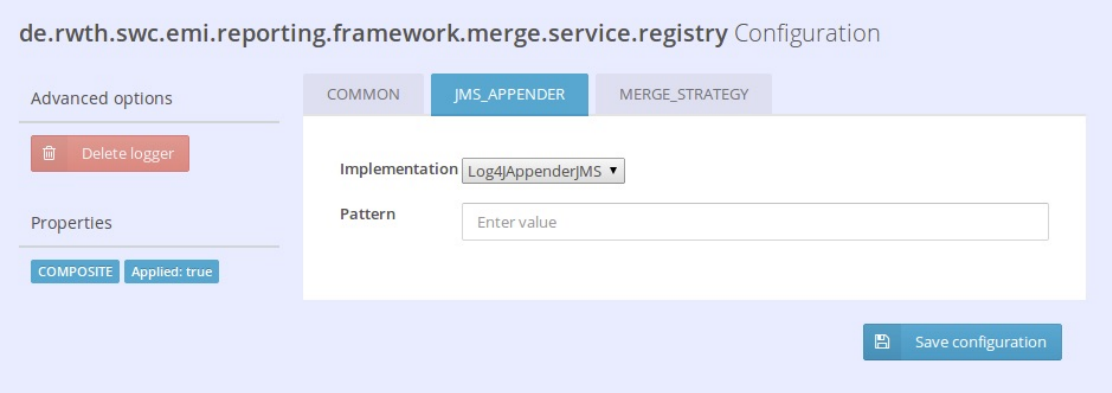

Abbildung 8.6: Werkzeug - Konfigurationsansicht - Loggerkonfiguration

8.2.2 Technische Log Darstellung

Der vorangegange Abschnitt hat die Konfigurationsansicht der Realisierung des Visualisierungs- & Konfigurationswerkezug dargestellt. In diesem Abschnitt wird die nächste Ansicht, nämlich die Darstellung der technische Logs betrachtet.

Der in Abbildung 8.7 präsentierte Screenshot der technischen Logansicht, zeigt, dass die Realisierung sehr nah an den grafischen Prototypen erfolgt ist. Auf der linken Seite werden die Logger eines adaptieren Service nach EMI Servicetyp gruppiert dargestellt. Pro adaptierten Service sind die zugehörigen Logger hierachisch abgebildet und lassen sich wie in der Konfigurationsansicht zusammenfalten (siehe 8.2.1). Die Darstellung der Logger verzichtet auf sämtliche Indikatoren, sondern zeigt die für den Anwender relevante Information: den Namen des Loggers. Wählt der Anwender einen oder mehrere Logger aus, werden auf der rechten Seite die zugehörigen Log-Einträge angezeigt. Zugehörig bedeutet dabei, dass der Erzeuger des Log-Eintrags ein ausgewählter Logger ist.

Die Darstellung der Log-Einträge erfolgt in einer paginierbaren Tabelle. Paginierbar bedeutet, dass zur Übersichtlichkeit nicht alle Log-Einträge gleichzeitig dargestellt werden, sondern lediglich eine feste, aber variable, Anzahl an Einträgen pro Seite visualisiert wird. Die Paginierung wurde serverseitig realisiert um *NFA-T-1 - Usability (Bedienbarkeit)* auch bei vielen Log-Einträgen zu gewährleisten. Die einzelnen Spalten der Tabelle zeigen die wichtigsten Felder eines Logeintrages und ermöglichen eine mehrdimensionale Sortierung. Selektiert der Anwender einen Logeintrag, werden die Detailinformationen des ausgewählten Log-Eintrages sichtbar (siehe Abbildung 8.7).

Die konzeptionierten Logmanagement-Funktionen (vgl. Unterabschnitt 8.1.2) werden durch eine Suche in Form einer Volltextsuche über alle Felder der Log-Einträge realisiert, wobei die Suche auch nicht angezeigte Log-Einträge analysiert.

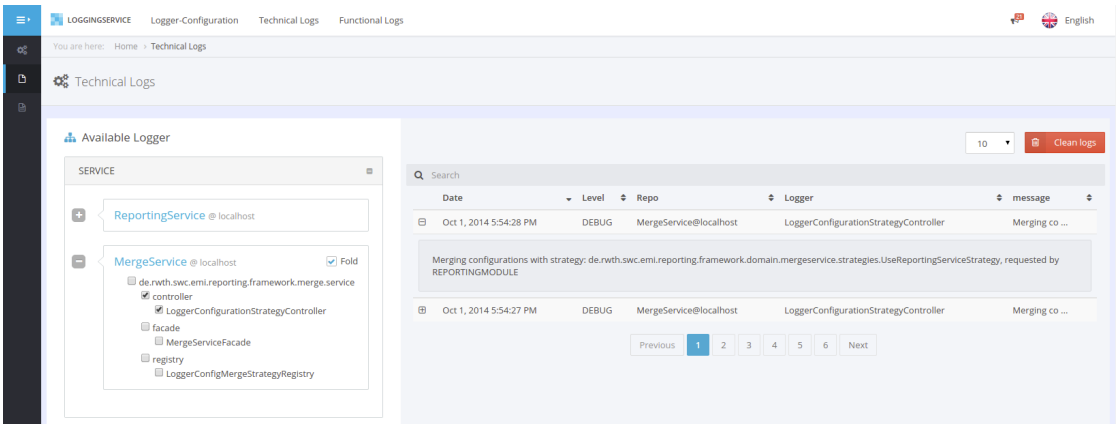

Abbildung 8.7: Werkzeug - Technische Logansicht

Neben der Visualisierung der Log-Einträge ermöglicht das Werkzeug diese zu löschen. Über den in Abbildung 8.8 dargestellten Löschdialog kann der Anwender zunächst eine Filterung der Log-Einträge vornehmen und anschließend ausgewählte Log-Einträge löschen.

8.2.3 Fachliche Log Darstellung

Nachdem der vorherige Abschnitt die Realisierung der technischen Log Darstellung beschrieben hat, stellt dieser Abschnitt die Ansicht der fachlichen Log Einträge vor. Da die Darstellung der fachlichen Logs (Abbildung 8.9) ähnlich zu der technischen Log-Ansicht (8.2.2) ist, werden im Folgenden nur die Unterschiede beschrieben.

Im Gegensatz zu technischen Log Einträgen haben fachliche Logs einen Projektbezug bzw. einen Bezug zu einer vermessenen Einheit (EOM). Aus diesem Grund selektiert der Anwender zunächst das EOM, bevor die entsprechenden Logger gefiltert nach diesem EOM auf der linken Fensterseite angezeigt werden. Durch die Filterung werden nur Logging Agenten angezeigt, deren Logger einen fachlichen Log für das gewählte EOM erzeugt haben.

Abgesehen von dieser EOM-Filterung und der Darstellung des EOMs innerhalb der Log-Eintrag-Tabelle unterscheidet sich die Ansicht der fachlichen Logs nicht von der Darstellung der technischen Logs.

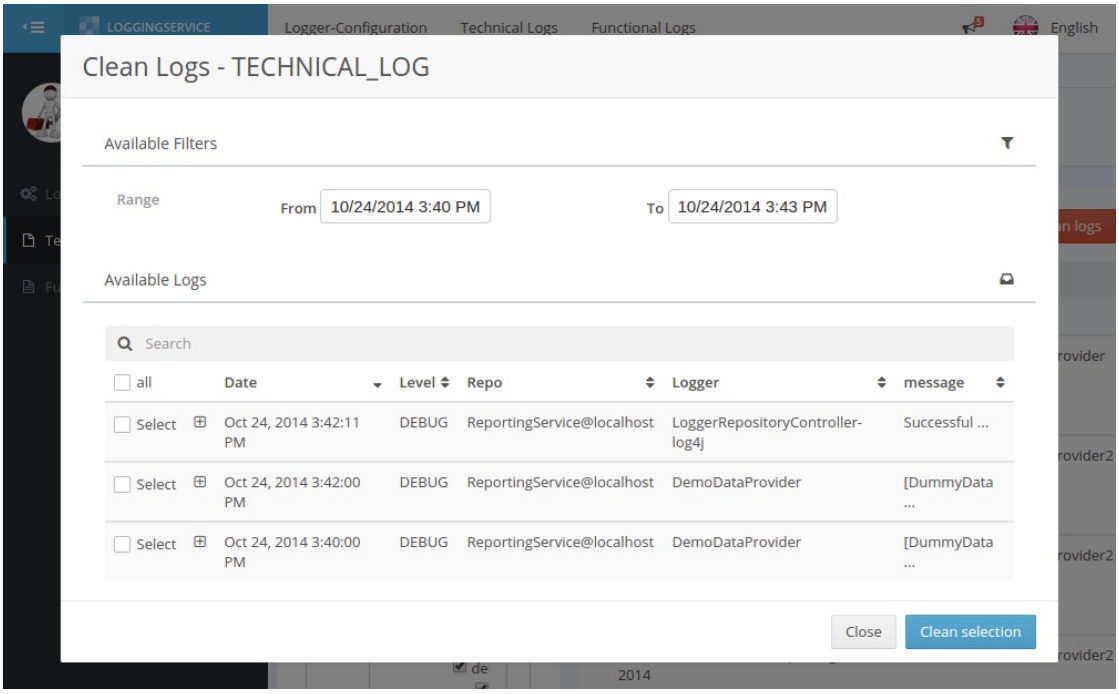

Abbildung 8.8: Werkzeug - Logansicht - Logs löschen

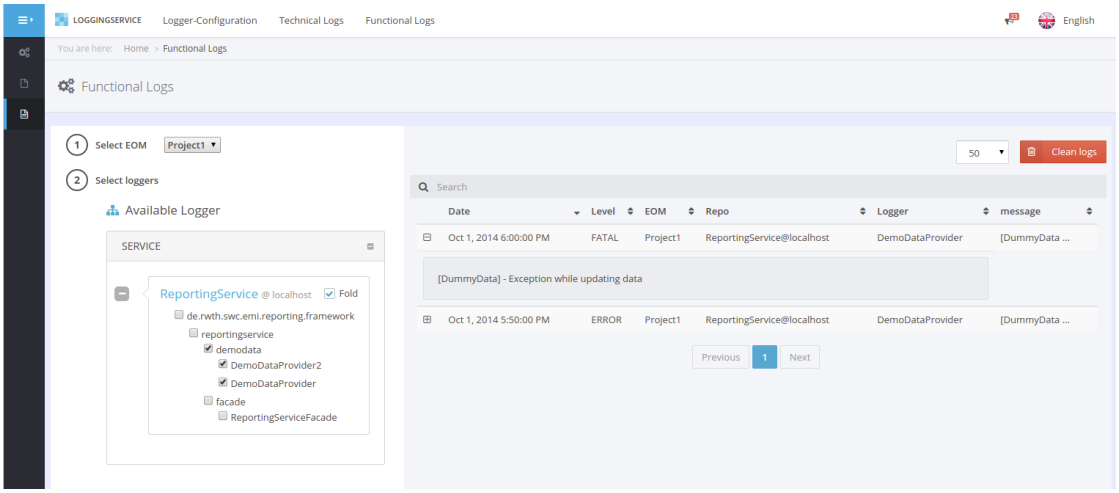

Abbildung 8.9: Werkzeug - Fachliche Logansicht

8.2.4 Technologien

In den bisherigen Abschnitten dieses Kapitels wurde die Gestaltung des Visualisierungs- & Konfigurationswerkzeugs beschrieben, sowie die Anwenderinteraktion dargestellt. Dieser Abschnitt widmet sich der zugrunde liegenden Technik dieses Werkzeugs.

Das Visualisierungs- & Konfigurationswerkzeug wurde als Webapplikation realisiert. Basierend auf dem MVC (Model-View-Controller) Framework, AngularJS [1], wurden diese Webapplikation mithilfe der Standard Webtechnologien (HTML, JS, Sass bzw. CSS) entwickelt, sodass die Betriebsvoraussetzungen gering sind und zur Verwendung lediglich ein Webbrowser benötigt wird.

Die Daten des Logging Service konsumiert das Werkzeug über dessen RESTful API. Der Datenkonsum folgt dabei dem lazy-loading Entwurfsmuster [Fow02], sodass nur notwendige Daten abgefragt werden und auch bei großen Datenmengen *NFA-T-1 - Usability (Bedienbarkeit)* erfüllt werden kann.

Wegen *NFA-T-2 - Aktualität* baut das Visualisierungs- & Konfigurationswerkzeug über einen Websocket einen bidirektionalen Kommunikationskanal zum Logging Service auf, um bei Aktualisierungen invoke-push orientiert [LVS14] die aktuellen Daten anzuzeigen.

[1] https://angularjs.org/

9 Evaluation

> I have not failed. I've just found 10,000 ways that won't work.
>
> Thomas Edison

Inhalt

Ausgehend von den in Kapitel 3 definierten Anforderungen an die Logging Infrastruktur, wurde in Kapitel 5 die Architektur der Logging Infrastruktur modelliert. Basierend auf dieser Architektur beschäftigten sich die nachfolgenden Kapiteln mit den modellierten Komponenten und Services der Logging Infrastruktur (siehe Kapitel 6, Kapitel 7 und Kapitel 8). In diesem Kapitel wird die vorgestellte Lösung evaluiert und bewertet.

In den folgenden Abschnitten gliedert sich die Evaluation der realisierten Logging Infrastruktur in eine Untersuchung der Anforderungsüberdeckung, sowie einer Analyse der technischen Realisierung, um die in dieser Arbeit entstandene Lösung bewerten zu können. Neben der isolierten Betrachtung der Logging Infrastruktur, wird in Abschnitt 9.3 im Rahmen der am Lehr- und Forschungsgebiet 3 eingesetzen EMI, die Auswirkung der Logging Infrastruktur auf bestehende Komponenten einer EMI untersucht.

9.1 Anforderungen

Nach dem Institute of Electrical and Electronics Engineers (IEEE) bemisst sich die Qualität eines Softwaresystems nach seiner Eignung für den vorgesehenen Zweck [IEE90]. Die Eignung für den vorgesehenen Zweck lässt sich mithilfe der Anforderungen untersuchen. Aus diesem Grund betrachtet dieser Abschnitt die Anforderungsüberdeckung der realisierten Logging Infrastruktur anhand der in Kapitel 3 definierten Anforderungen. Jede Anforderung wird zunächst zusammengefasst und anschließend der Grad der Überdeckung bewertet.

9.1.1 FA1 - verteiltes Logging

Zusammenfassung Abgeleitet aus der Herausforderung der Log Integration in einem verteilten System, fordert die Anforderung des verteilten Loggings, dass Log Einträge aller angeschlossenen Services integriert und persistiert werden können.

Bewertung Die in dieser Arbeit realisierte Logging Infrastruktur ermöglicht verteiltes Logging durch einen push-basierten, zentralistischen Ansatz. Die Log-Erzeugung ist Aufgabe der Anwendung. Über Loggingbibliothek-spezifische Appender werden die Logger des externen Systemes zur Veröffentlichung ihrer Log Events auf dem Logging Bus automatisch durch den Logging Agenten konfiguriert. Veröffentlichte Log Events werden anschließend vom Logging Service empfangen und zentral gesammelt, sodass insgesamt FA1 - verteiltes Logging erfüllt wird.

9.1.2 FA2 - Integrierbarkeit

Zusammenfassung FA2 - Integrierbarkeit fordert, dass die Logging Infrastruktur in einer EMI verwendet werden kann.

Bewertung Die Logging Infrastruktur stellt die Integrierbarkeit durch mehrere Aspekte sicher. Zum einen ist die gesamte Logging Infrastruktur ähnlicher einer EMI aufgebaut. Jede Komponente der Logging Infrastruktur orientiert sich architektonisch an dem entsprechenden EMI Pendant. Der Logging Agent lässt sich mit einem Datenadapter vergleichen, der Logging Service ähnelt einem Metrikkernel und das Visualisierungs- & Konfigurationswerkzeug kann als ein Dashboard zur Visualisierung der Log Daten verstanden werden. Neben dieser strukturellen Ähnlichkeit, die die Gesamtkonsistenz einer EMI erhält und damit das zur Integration benötigte Verständnis erhöht, kommunizieren Logging Agenten und Logging Service über einen Logging Bus, der wie in Abschnitt 5.4 beschrieben als dedizierter Kanal des EMDB realisiert werden kann. Da der Logging Agent außerdem keine Anforderungen an den zu integrierenden Service stellt (keine Datenbank, kein Dateizugriff nötig), ist die einfache Integration der Logging Infrastruktur in eine EMI sichergestellt.

9.1.3 FA4 - Kompatibilität

Zusammenfassung Da die Logging Infrastruktur die Log Erzeugung der Anwedung überlässt und um keine Codeanpassung vorauszusetzen, fordert FA4 - Kompatibilität, dass die Logging Infrastruktur mit verschiedenen Logging Bibliotheken hinsichtlich der Log Veröffentlichung und Logger Konfiguration umgehen kann und somit dem Problem der Loggerheterogenität begegnet.

Bewertung Die Logging Infrastruktur löst das Problem der Loggerheterogenität über Loggingbibliothek-spezifische Adapter. Jeder Logging Agent besitzt verschiedene Logging Adapter, die über entsprechende Komponenten unter Beachtung des

Trennung der Zuständigenkeiten Prinzips [LL10] die Aufgaben der Loggerentdeckung, Loggerkonfiguration und Log Event Veröffentlichung übernehmen und somit die Kompatibilitätsanforderung erfüllt ist.

9.1.4 FA5 - Konfiguration zur Laufzeit

Zusammenfassung Die zweite wichtige Aufgabe der Logging Infrastruktur ist neben dem verteilten Logging die Konfiguration der Logger zur Laufzeit. Dabei setzt die Anforderung eine flexible Konfiguration voraus, d.h. die Konfiguration eines Loggers kann lokal am adaptierten System über die Loggingbibliothek-spezifischen Mechanismen (z.B. per Konfigurationsdatei) oder entfernt über den Logging Service ausgelöst werden. Auch soll eine veränderte Konfiguration nach einem Neustart des entsprechenden Service weiterhin angewandt sein.

Bewertung FA5 - Konfiguration zur Laufzeit besitzt mehrere Unteranforderungen. Zum einen benötigt die Logging Infrastruktur die Möglichkeit, einen Logger konfigurieren zu können. Diese Aufgabe übernimmt der Loggingbibliothek-spezifische ConfigHandler des entsprechenden Logging Agenten. Zum anderen muss die Logging Infrastruktur aufgrund der Kompatibilität zu unterschiedlichen Logging Bibliotheken heterogene Loggerkonfigurationen abbilden können, um diese speichern, kommunizieren und darstellen zu können. Diese Herausforderung löst die Logging Infrastruktur durch Verwendung eines Variabilitätsansatzes
Die dritte Unteranforderung der Konfigurationsanforderung ist die Detektierung von extern durchgeführten Konfigurationsänderungen. Da sich nicht jede Logging Bibliothek um einen Benachrichtigungsmechanismus für Konfigurationsänderungen erweitern lässt, vergleicht jeder Logging Agent regelmäßig die aktuell angewandte Konfiguration mit der im LoggerConfigCache vorrätig gehaltenen Konfiguration, sodass er Änderungen detektieren kann. Die vierte Unteranforderung der Konfigurationsanforderung resultiert aus Konflikten zwischen der lokal angewandten Konfigurationen und der Logging Service seitig gespeicherten. Nach einem Neustart eines Service ist beispielsweise nicht deterministisch bestimmt, welche Konfiguration (lokal vs. remote) verwendet werden soll. Diesen Konflikt löst die Logging Infrastruktur mithilfe des Merge Service, sodass sie insgesamt alle Unteranforderungen der Konfigurationsanforderung erfüllt und damit ganzheitlich FA5 - Konfiguration zur Laufzeit erfüllt ist.

9.1.5 FA6 - Kategorisierung der Log Einträge

Zusammenfassung Bedingt durch die Zielumgebung einer EMI in der projektspezische bzw. fachliche und technische Informationen protokolliert werden, setzt FA6 - Kategorisierung der Log Einträge voraus, dass sich die veröffentlichten Log Events kategorisieren lassen, um eine spätere Filterung abhängig vom Stakeholder durchführen zu können.

Bewertung Im Rahmen der Logging Infrastruktur werden die Log Events vor der Veröffentlichung durch den entsprechenden Appender kategorisiert. Die Kategorisierung erfolgt auf Basis der protokollierten Informationen, sodass der Entwickler apriori die Kategorie festlegt. Insgesamt ist damit die Kategorisierungsanforderung der Logs erfüllt.

9.1.6 FA7 - Visualisierungs- & Konfigurationswerkzeug

Zusammenfassung Um dem Anwender die integrierten Log Einträge darstellen zu können und die Konfiguration der Logger zu ermöglichen, wird ein Visualisierungs- & Konfigurationswerkzeug gefordert. Konkret soll dieses Werkzeug eine fachliche und technische Sicht auf die Log Einträge ermöglichen, sowie heterogene Loggerkonfigurationen darstellen und konfigurieren können.

Bewertung Das im Rahmen der Logging Infrastruktur realisierte Visualisierungs- & Konfigurationswerkzeug erfüllt diese Anforderung. In seinen drei verschiedenen Ansichten stellt es die Loggerkonfigurationen dar, visualisiert die technischen Log Einträge und zeigt die fachlichen Log Einträge. Die Darstellung und Konfiguration der Loggerkonfigurationen ist dabei aufgrund des Variabilitätsansatzes generisch, sodass heterogene Loggerkonfigurationen abgebildet werden können. Insgesamt erfüllt die Logging Infrastruktur daher die Anforderung des Visualisierungs- & Konfigurationswerkzeuges.

9.1.7 NFA-I-1 - Stabilität

Zusammenfassung Die Nicht-funktionale Stabilitätsanforderung verlangt, dass ein Ausfall der Logging Infrastruktur die Funktionalität der restlichen EMI Services nicht beeinflusst.

Bewertung Zur Bewertung der Anforderungsüberdeckung muss zunächst der Ausfall der Logging Infrastruktur definiert werden. Da die Logging Infrastruktur ein verteiltes System darstellt, fällt sie aus, wenn ihre zentralen Komponenten ausfallen. Die zentrale Komponenten der Logging Infrastruktur stellt der Logging Service dar. Zur Bewertung der Anforderungsüberdeckung wurde daher ein Ausfall des Logging Service simuliert. Diese Deaktivierung über den Mechanismus des Application Server zeigte aufgrund der entworfenen, loosen Kopplung zwischen Logging Agenten und Logging Service keine negativen Auswirkung. Selbstverständlich konnte das Visualisierungs- & Konfigurationswerkzeug nicht mehr auf die Logdaten und Konfiguration zugreifen, aber die restlichen EMI Services wurden nicht beeinflusst. Nach einer anschließenden Reaktivierung des Logging Service wurden zwischenzeitlich lokal durchgeführte Konfigurationsänderungen erfolgreich synchronisiert. Allerdings gingen Log Events, die während des „Ausfalls"der Logging Infrastruktur auftraten, verloren. Dieses Verhalten ist durch das fehlende Caching der Log Event Nachrichten seitens des Logging Agenten zu erklären. Da der in der EMI Referenzarchitektur definierte MessageCache diese Aufgabe übernehmen kann, wird der Verlust von Log Events minimiert, die während

eines Ausfalls der Logging Infrastruktur auftreten. Insgesamt beeinflusst der Ausfall des Logging Service also nicht die restlichen EMI Services und die Logging Infrastruktur erfüllt die Stabilitätsanforderung.

9.1.8 NFA-I-2 - Robustheit

Zusammenfassung NFA-I-2 - Robustheit fordert, dass der Ausfall eines oder mehrerer Logger oder Logging Agenten nicht zu einem Ausfall der Logging Infrastruktur führen darf.

Bewertung Zur Bewertung der Robustheitsanforderung sind unterschiedliche Szenarien zu betrachten. Entweder kann ein Logger ausfallen, während der zugehörige Logging Agent noch verfügbar ist, oder der Logging Agent fällt komplett aus (beispielsweise weil der adaptierte EMI Service ausfällt). Zusätzlich zu diesen beiden Szenarien kann auch noch der Merge Service ausfallen. Diese drei Szenarien werden nachfolgend bewertet.

Ausfall eines Loggers Szenario 1, also der Ausfall eines Logger, kann die Logging Infrastruktur nur beeinflussen, wenn dieser Logger remote konfiguriert werden soll. Der zugehörige Logging Agent kann mit dem Ausfall des Loggers jedoch umgehen, und antwortet dem Logging Service mit der als nicht-angewandt markierten Konfiguration, sodass die Funktionalität der Logging Infrastruktur nicht eingeschränkt wird und dem User mithilfe der Markierung ein Feedback gegeben werden kann. Ist der Logger später wieder verfügbar, wird die Konfiguration nachträglich durchgeführt.

Ausfall eines Logging Agenten Szenario 2, also der Ausfall eines Logging Agenten würde die Logging Infrastruktur aufgrund der loosen Kopplung des Logging Agenten und des Logging Service nur bei einer Interaktion des Logging Service mit den Logging Agenten beeinflussen, d.h. das Remotekonfigurationsszenario muss zur Bewertung betrachtet werden. Wird über den Logging Service eine Remotekonfiguration durchgeführt, markiert der Logging Service alle Loggerkonfigurationen des Logging Agenten als nicht angewandt, bis er eine Bestätigung der Konfiguration erhält. Ist der Logging Agent aufgrund seines Ausfalls nicht verfügbar, wird dem Anwender mithilfe der nicht-angewandt Markierung dieses Szenario visualisiert. Sobald der Logging Agent wieder verfügbar ist, wird die Remotekonfiguration synchronisiert, sodass die Logging Infrastruktur den Ausfall kompensieren kann.

Ausfall des Merge Service Szenario 3 behandelt ebenfalls das Konfigurationsszenario, nämlich den Ausfall des Merge Service. Fällt der Merge Service aus, verwendet der Logging Service zur Konfliktlösung zweier Konfiguration eine Heuristik, die immer die Remotekonfiguration verwendet, sodass trotz Nichtverfügbarkeit des Merge Service eine Konfiguration möglich ist.

Insgesamt erfüllt die Logging Infrastruktur alle zur Bewertung der Robustheitsanforderung notwendingen Szenarien, sodass sie NFA-I-2 - Robustheit erfüllt.

9.1.9 NFA-I-4 - Erweiterbarkeit

Zusammenfassung Die Erweiterbarkeitanforderung fordert die Möglichkeit einer funktionalen Erweiterung, sowie eine Erweiterung um verschiedenen Logging Bibliotheken.

Bewertung Die Logging Infrastruktur stellt durch zahlreiche Hotspots die Erweiterbarkeit sicher. Durch den modularen Aufbau lassen sich Komponenten schnell und einfach austauschen, sodass beispielsweise das Kommunikationsmedium flexibel gewählt werden kann. Aufgrund der Plugin Architektur der Logging Adapter und der verschiedenen Merge Strategien lässt sich die Logging Infrastruktur leicht um eine Unterstützung weiterer Logging Bibliotheken erweitern und neue Merge Strategien modular hinzufügen. Neue Logkategorien oder neue Nachrichtentypen und damit neue Funktionalitäten lassen sich leicht realisieren und die Verwendung des Variabilitätsansatzes ermöglicht beliebige Konfigurationen. Insgesamt erfüllt die Logging Infrastruktur damit die Erweiterbarkeitsanforderung.

9.1.10 NFA-T-1 - Usability (Bedienbarkeit)

Zusammenfassung Neben Anforderungen an die Logging Infrastruktur im Allgemeinen, wurden in Kapitel 3 Nicht-funktionale Anforderung an das Visualisierungs- & Konfigurationswerkzeug definiert. Die Usability-Anforderung ist eine davon. Sie fordert, dass das Visualisierungs- & Konfigurationswerkzeug alle Funktionen zur effizienten Bewältigung der Aufgaben bietet, konsistent und übersichtlich ist.

Bewertung Die Bewertung der Bedienbarkeitsanforderung erfolgt anhand der DIN EN ISO 9241 - 110 Norm [ISO06]. Die Analyse mithilfe des entsprechenden Fragebogens hat dabei gezeigt, dass das Visualisierungs- & Konfigurationswerkzeug der Ergonomie Norm teilweise entspricht. Es bietet alle Funktionen, um die anfallenden Aufgaben effizient zu bewältigen. Die Seiten der Log Einträge bieten umfangreiche Such- und Sortierfunktionen, die Funktionen sind nicht in mehrstufigen Menüs verschachtelt, sondern direkt ersichtlich und alle Daten werden anhand des lazy-loading Musters geladen, sodass insbesondere nur die betrachteten Log Einträge vom Logging Service abgefragt werden müssen. Die Konfigurationsseite erfordert keine überflüssigen Eingaben, denn durch die Darstellung der Logger in einer Hierachie lassen sich mehrere Logger auf einmal konfigurieren. Durch die Verwendung des Variabilitätsansatzes liefert es durch vorgegebene Auswahlmöglichkeiten bei einem geschlossenen Variabilitätspunkt in ausreichender Menge Informationen darüber, welche Eingaben zulässig oder nötig sind. Insgesamt ermöglicht es durch die einheitliche Gestaltung der Loggerhierachie auf der linken und dem zugehörigen Inhalt auf der rechten Seite eine einfache Orientierung

und die Bedienung nach einem einheitlichen Prinzip (Auswahl links, Interaktion rechts). Durch die immer präsente Navigation auf der linken Seite, ist ebenfalls ein leichter Wechsel zwischen den einzelnen Ansichten möglich.
Die von der ISO Norm geforderten situationsspezifischen Erklärungen oder Hilfestellungen, sowie Fehlerbehandlung im Allgemeinen bietet das Visualisierungs- & Konfigurationswerkzeug nicht. Dadurch ist es auch für unerfahrene Anwender ohne Vorkenntnisse der Logging Domäne nicht leicht zu erlernen und bedienen.
Insgesamt erfüllt das Visualisierungs- & Konfigurationswerkzeug zwar die gestellten Anforderungen hinsichtlich Funktionalität, Konsistenz und Übersichtlichkeit, die DIN EN ISO 9241 - 110 Norm erfüllt es allerdings nur bedingt. Vor dem Hintergrund, dass es sich um ein Expertenwerkzeug handelt und einen prototypischen Charakter besitzt, lässt sich diese Einschränkung akzeptieren.

9.1.11 NFA-T-2 - Aktualität

Zusammenfassung Die Aktualitätsanforderung bezieht sich ebenfalls auf das Visualisierungs- & Konfigurationswerkzeug. Sie fordert, dass die dargestellten Informationen immer aktuell sein sollen. Tritt während der anwenderseitigen Betrachtung der Log Einträge ein Log Event auf, soll der entsprechende Log Eintrag direkt angezeigt werden. Ebenfalls soll eine Konfigurationsänderung direkt dargestellt werden.

Bewertung Der Logging Service stellt neben einer RESTful Schnittstelle eine Websocket API zur Verfügung, die eine bidirektionale Kommunikation des Logging Service mit dem Visualisierungs- & Konfigurationswerkzeug ermöglicht. Über diese Schnittstelle informiert der Logging Service das Visualisierungs- & Konfigurationswerkzeug, sofern ein neues Log Event aufgetreten oder eine Konfigurationsänderung stattgefunden hat. Daraufhin aktualisiert das Visualisierungs- & Konfigurationswerkzeug die dargestellten Informationen, sodass insgesamt die Aktualitätsanforderung erfüllt ist.

9.2 Technische Analyse

Im vorherigen Abschnitt wurde die Anforderungsüberdeckung der Logging Infrastruktur evaluiert, um Aussagen über die (funktionale) Qualität der Logging Infrastruktur, also die Eignung der Logging Infrastruktur für den vorgesehenen Zweck (vgl. Qualitätsdefinition des IEEE [IEE90]), treffen zu können. In diesem Abschnitt wird die technische Qualität betrachtet. Da das Visualisierungs- & Konfigurationswerkzeug einen prototypischen Charakter besitzt und nicht Kern der Logging Infrastruktur ist, erfolgt in den folgenden Abschnitten eine getrennte Analyse. Zunächst wird die Logging Infrastruktur analysiert und anschließend das Visualisierungs- & Konfigurationswerkzeug. Die technische Analyse beruht dabei auf einer statischen Codeanalyse.

9.2.1 Logging Infrastruktur

In diesem Abschnitt wird die technische Qualität der Logging Infrastruktur analysiert. Zur statischen Codeanalyse wurde Codequalitäts-Analysetool SonarQube verwendet.

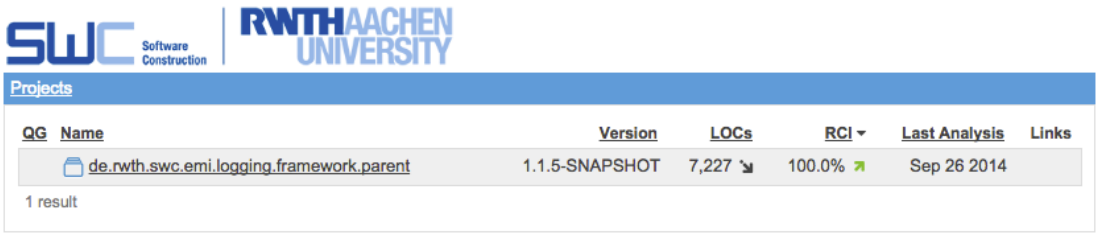

Abbildung 9.1: SonarQube - Projektübersicht - Logging Infrastruktur

Wie die Projektübersicht in Abbildung 9.1 zeigt, wurde die Logging Infrastruktur (exklusiv des Visualisierungs- & Konfigurationswerkzeug) in 7227 Quellcodezeilen (LOC) realisiert. Da die Logging Infrastruktur als Grundlage für weiterführende Logging-Aufgaben innerhalb einer EMI ausgelegt ist, wurde auf eine technisch hochwertige Umsetzung geachtet. Wie in Abbildung 9.1 zu erkennen ist, wurden durch Sonar keine Verstöße gegen Kodierungsrichtliniengefunden, was einem RCI von 100% entspricht.
Die zyklomatische Komplexität (CC) der Logging Infrastruktur Realisierung beträgt durchschnittlich 2,2 pro Methode und maximal 12, wie das Histogramm in Abbildung 9.2 zeigt. Nach dem C4 Software Technology Reference Guide der Carnegie Mellon University [FGR+97] sind Methoden mit einer CC zwischen 1 und 10 einfach und verständlich. Eine CC zwischen 10 und 20 indiziert komplexeren Code, der unter Umständen noch verständlich ist, aber das Testen erschwert. Mit steigender CC wird der Code immer schwieriger zu verstehen und zu testen, bis ab einer CC von 50 die Methode unwartbar ist. Die maximale CC der Logging Infrastruktur von 12 liegt damit im vollkommen akzeptablen Bereich. Insgesamt wurde die sehr gute Codequalität der Logging Infrastruktur damit durch Messungen belegt.

9.2.2 Visualisierungs- & Konfigurationswerkzeug

Nachdem im vorherigen Abschnitt die technischen Qualität des Logging Infrastruktur Kerns analysiert wurde, beschäftigt sich dieser Abschnitt mit der technische Qualität des Visualisierungs- & Konfigurationswerkzeug. Da das Visualisierungs- & Konfigurationswerkzeug nicht zum Kern der Logging Infrastruktur gehört und damit von der Analyse exkludiert wurde, besitzt dieser Abschnitt einen informativen und keinen qualitativen Charakter.
Zur statischen Codeanalyse wurde ebenfalls das Analysewerkzeug SonarQube verwendet. Wie die Analyse in Abbildung 9.3 zeigt, wurde das Visualisierungs- & Konfigurationswerkzeug in 2439 Quellcodezeilen realisiert. Die Betrachtung des zyklomatischen Komplexitätswerts verdeutlicht den prototypischen Charakter der

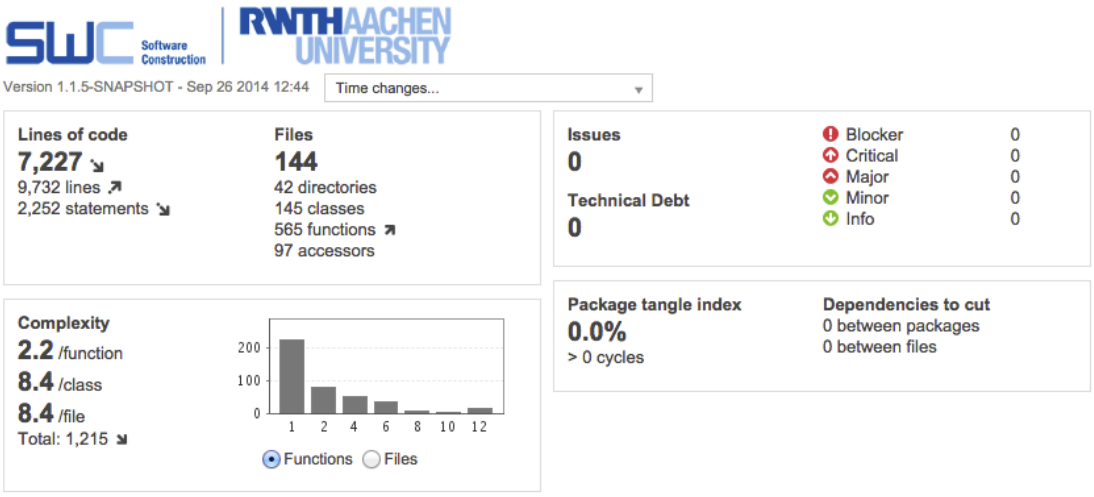

Abbildung 9.2: SonarQube - Projektdetails - Logging Infrastruktur

Realisierung: Während ein CC von 1,9 pro Funktion sehr gut ist, weisen Komplexitätswerte von 20,1 pro Datei und über 20 einiger Methoden stark auf diesen Charakter hin.

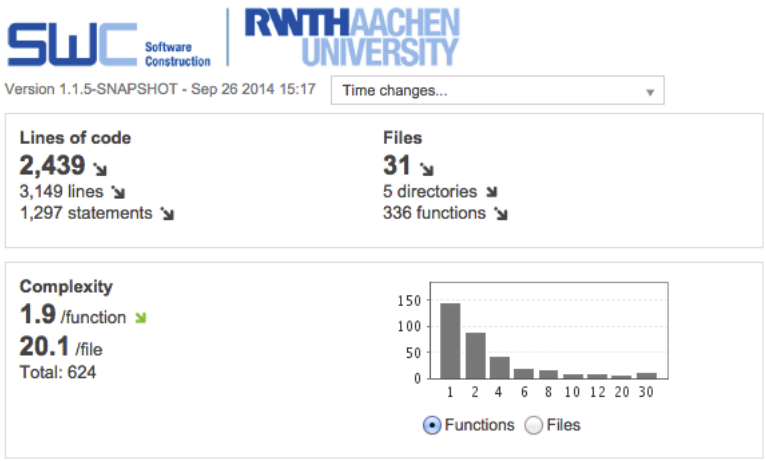

Abbildung 9.3: SonarQube - Visualisierungs- & Konfigurationswerkzeug

Insgesamt wurde die Logging Infrastruktur im Rahmen dieser Arbeit damit in 9666 Quellcodezeilen realisiert.

9.3 Auswirkung des Loggings auf bestehende Komponenten einer EMI

In den vorherigen Abschnitten wurde die Logging Infrastruktur den in Kapitel 3 gestellten Anforderungen gegenübergestellt und anschließend die technische Qualität der Realisierung untersucht. Um die realisierte Logging Infrastruktur vollständig evaluieren zu können, untersucht dieser Abschnitt die Auswirkungen der Logging Infrastruktur auf bestehende Komponenten einer EMI durch Messungen.
Um ein realitätsnahes Ergebnis zu erhalten, wurde als Messinfrastruktur die am Lehr- und Forschungsgebiet 3 eingesetzte EMI verwendet. Die Messdurchführung erfolgte mithilfe des Ticketanalyse Werkzeugs River, welches Christian Charles in seiner Diplomarbeit „Entwurf eines generischen Prozessleitstandes für Change Request Systeme"[Cha13] entwickelt und Endri Gjino [Gji13] überarbeitet hat. River ermöglicht unter anderem den Import von Tickets bzw. Tickettransitionen aus Change-Request-Systemen, wobei eine Tickettransition dem Statuswechsel eines Tickets (z.B. offen -> in Bearbeitung) entspricht. Da der Ticketimport Vorgang über den EMDB durchgeführt wird und viele Logs erzeugt, eignet sich der Ticketimport zur Analyse des zusätzlichen Aufwandes (Overhead) durch die Logging Infrastruktur.

Insgesamt wurden fünf, von der ITErgo bereitgestellte, Testdatensätze zum Ticketimport verwenden. Die Datensätze wurden entsprechend groß gewählt, damit ein eventueller zeitlicher Overhead durch die Logging Infrastruktur messbar ist. Der kleinste Datensatz umfasst 50 Tickettransitionen, was unter Berücksichtigung des INFO Log Levels einer Erzeugung von 142 Log Einträgen entspricht. Als kleiner Datensatz wurden 71 Tickettransitionen verwendet, was einer Erzeugung von 142 Log Einträgen entspricht. Der mittlere Datensatz umfasst 429 Tickettransitionen, d.h. 858 Log Einträge wurden erzeugt. Als großer Datensatz wurden 866 Tickettransitionen (1732 Logs) verwendet und der größte umfasst 1738 Tickettransitionen (3476) Logs.

Pro Datensatz wurde die Dauer eines River Importvorgangs gemessen. Ein Importvorgang beginnt dabei beim Auslösen des Importsvorganges durch den Anwender und endet nachdem die letzte Tickettransition importiert, d.h. in der Datenbank gespeichert wurde. Damit ein zusätzlicher Aufwand (Overhead) durch die Logging Infrastruktur bestimmt werden kann, wurden für jeden Datensatz verschiedene Messungen durchgeführt. Zur Isolierung des reinen River Import Dauer um eine Ausgangsbasis zu definieren, wurden Messungen mit ausgeschaltetem Logging (Log Level OFF) und ohne die Verwendung der Logging Infrastruktur durchgeführt. Anschließend wurde der Overhead des bisherigen Logdatei-Protokollierung (vgl. Unterabschnitt 2.3.2) mithilfe von Zeitmessungen bei aktivierten Logging (Log Level INFO) und ohne Verwendung der Logging Infrastruktur bestimmt. Die letzte Messreihe betrachtete dann die Dauer eines Importvorganges mit aktiviertem Logdatei Logging und gleichzeitiger Verwendung der Logging Infrastruktur (Log Level INFO).

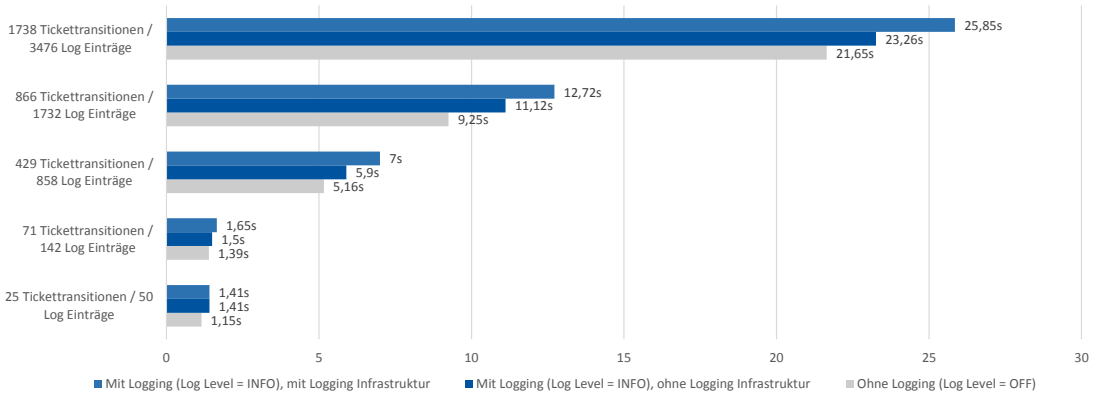

Abbildung 9.4: River Ticketimport Geschwindigkeitsmessung

Die gemittelten Ergebnisse der Messungen sind in Abbildung 9.4 dargestellt. Wird ausschließlich der Overhead der Logging Infrastruktur betrachtet, ergibt sich kein zusätzlicher Zeitaufwand bei Verwendung des kleinsten Datensatzes (50 Log Einträge). Der kleine Datensatz (142 Log Einträge) verursacht absolut einen zusätzlicher Zeitaufwand von 0,15 Sekunden. Durch den mittleren Datensatz (858 Log Einträge) wird ein Overhead von 1,1 Sekunden eingeführt und der Import des größten Datensatzes (3476) führt zu einem absoluten Logging Infrastruktur geschuldetem Overhead von 3,09 Sekunden.

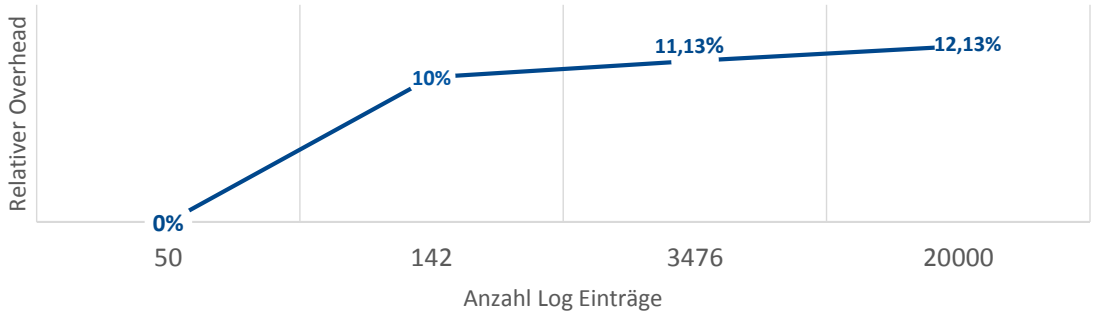

Abbildung 9.5: River Ticketimport - Relativer Overhead durch die Logging Infrastruktur

Zur Betrachtung des relativen Overheads in Abbildung 9.5 wurden weitere Messung mit 10000 Tickettransitionen, d.h. 20000 Logs durchgeführt, um das asymptotische Verhalten abschätzen zu können. Um die Bandbreite des relativen Overheads einschätzen zu können, wurde die anderen Messpunkte über das Spektrum verteilt, sodass neben der Messungen mit 10000 Tickettransitionen noch die Messungen mit 25, 71 und 1738 Tickettransitionen betrachtet wurden.

Wie Abbildung 9.5 zeigt, beträgt der relative Overhead maximal 12,13% bei einem River Import, der 20000 Log Einträge erzeugt und minimal 0%, wenn weniger als 50 Log-Einträge erzeugt werden. Dazwischen entwickelt sich der relative Overhead logarithmisch. Betrachtet man zum Vergleich den relativen Overhead des Monitoring Frameworks Kieker [1], der bei 10% liegt [vHWH12] und von den Autoren als sehr gering bezeichnet wird, so lässt sich der Overhead der Logging Infrastruktur ebenfalls als gering bezeichnen (Die Erzeugung von 20000 Log Einträgen erzeugt einen ähnlichen Durchsatz wie das Monitoring). Des Weiteren wird der Log Daten Durchsatz im Betrieb kleineren EMI Instanzen nicht 50 Log Einträge pro 1,41 Sekunden (vgl. Messung mit dem kleinsten Datensatz) überschreiten, sodass kein Overhead feststellbar ist.

9.4 Bewertung

Dieses Kapitel hat die im Rahmen dieser Bachelorarbeit realisierte Logging Infrastruktur evaluiert. Neben einer funktionalen Evaluation der Logging Infrastruktur durch Gegenüberstellung der Anforderungen mit der realisierten Lösung, wurde die Logging Infrastruktur mithilfe des Codequalitäts-Analysewerkzeug SonarQube technisch evaluiert. Ebenfalls wurde die Effizienz der Logging Infrastruktur durch Betrachtung des eingeführten Overheads beim Ticketimport analysiert.

Insgesamt erfüllt die Logging Infrastruktur alle an sie gestellten Anforderungen und besitzt eine sehr hohe technische (Codierungs-) Qualität. Verbesserungspotential bietet das Visualisierungs- & Konfigurationswerkzeug, dessen prototypischer Charakter sich während der Evaluation gezeigt hat. Ebenfalls lässt sich die Effizienz der Logging Infrastruktur weiter optimieren. Wie die Messungen im Kontext von River gezeigt haben, führt die Logging Infrastruktur bei gleichzeitiger Verwendung des bisherigen Loggingmechanismus einen Overhead von maximal 12,18 % ein. Auch wenn dieser Wert vertretbar ist und insbesondere erst bei der Erzeugung vieler Logeinträge (20000 Stück) auf einmal auftritt, ließe sich die Effizienz durch eine optimierte Ressourcenverwendung (Aggregation von Lognachrichten vor dem Versand, Sender-Pool, zentraler Appender) relativ einfach weiter verbessern. Insgesamt erfüllt die Logging Infrastruktur ihren Zweck und bietet eine breite Grundlage für vielfältige Verwendungsmöglichkeiten hinsichtlich der Loganalyse.

[1] http://kieker-monitoring.net/

10 Zusammenfassung & Ausblick

Ein guter Anfang braucht Begeisterung, ein gutes Ende Disziplin.

Prof. Dr. Hans-Jürgen Quadbeck-Seeger

Inhalt

Nachdem in den vorherigen Kapiteln die Logging Infrastruktur vorgestellt und evaluiert wurde, fasst dieses Kapitel die Arbeit zusammen und bietet einen Ausblick auf mögliche Weiterentwicklungen bzw. Verbesserungen.

10.1 Zusammenfassung

Logging, also die Protokollierung von Ereignissen zur Laufzeit eines Systems, ist eine wichtige Grundlage für viele verschiedene Verwaltungsaufgaben eines Softwaresystem, die auf einer Analyse der protokollierten Ereignisse beruhen. Grundvoraussetzung einer sinnvollen Analyse ist nicht nur die Verfügbarkeit der benötigten Informationen pro protokollierten Ereignis, sondern auch die Verfügbarkeit der protokollierten Ereignisse überhaupt. Während in monolithischen Systemen diese generelle Verfügbarkeit meist implizit erfüllt ist, stellt die Integration aller protokollierte Ereignisse innerhalb eines verteiltes Systems eine große Herausforderung dar.

Neben dieser Integrationsproblematik führen die vielseitigen Verwendungsmöglichkeiten des Logging zu einer weiteren, Stakeholder spezifischen, Herausforderung. Während die Betreiber eines Softwaresystems aus Performancegründen beispielsweise nur an der Protokollierung von wichtigen Ereignissen interessiert sind, benötigen Entwickler zur Fehlersuche möglichst viele Informationen aus allen protokollierbaren Ereignissen. Das Logging muss sich also zur Laufzeit konfigurieren lassen, was in einem verteilten System neben der Synchronisationsproblematik auch durch die Heterogenität der Logging Bibliotheken eine Herausforderung darstellt.

Im Rahmen dieser Arbeit wurde eine Logging Infrastruktur entwickelt, die den Herausforderungen der Log Integration und der Loggerkonfiguration in einem verteilten

System begegnet. Das konkrete Systemumfeld stellt dabei eine Messinfrastruktur dar, die sich aufgrund ihres Mikro-Service Charakter ideal für das Logging Szenario eignet und insbesondere zur späteren Analyse eine zentralen Zugriffsmöglichkeit auf alle, innerhalb dieser Messinfrastruktur aufgetretenen Logs Einträge benötigt.

Die entwickelte Logging Infrastruktur besteht hauptsächlich aus vier Komponenten: Dem Logging Service, dem Merge Service, dem Logging Agent und dem Visualisierungs- & Konfigurationswerkzeug. Der Logging Service stellt die zentrale Persistierungs- & Verwaltungskomponente der Logging Infrastruktur dar. Über seine Schnittstellen kann das Visualisierungs- & Konfigurationswerkzeug auf alle protokollierten Ereignisse zugreifen, dem Anwender visualisieren und Remotekonfigurationsvorgänge, d.h. die Konfiguration eines entfernten Logger eines externen Systemes, auslösen. Für die Adaptierung und Loggerkonfiguration eines externen Systemes ist der Logging Agent zuständig. Mithilfe verschiedener Loggingbibliothek-spezifischer Adapter kann er auf die Logger des externen Systems zugreifen, sie konfigurieren und ihre Log Events über ein Bussystem entsprechend an den Logging Service weiterleiten. Da heterogene Konfigurationen abgebildet werden müssen, verwendet die Logging Infrastruktur einen Variabilitätsansatz, mit dem beliebige Konfigurationen abgebildet und validiert werden können. Aufgrund der eingeschränkten Verfügbarkeit der Systeme in einem verteilten System, müssen die unterschiedlichen Loggeronfigurationen synchronisiert werden. Da im Rahmen des Synchronisationsprozess Konflikte auftreten können, wurde ein Merge Service entwickelt um mithilfe verschiedener Strategien eventuelle Konflikte zu lösen.

Wie die Evaluation gezeigt hat, erfüllt die Logging Infrastruktur alle gestellten Anforderungen und löst die Herausforderungen der Log Integration und verteilten, heterogenen Loggerkonfiguration. Sie bietet eine hohe technische Qualität und vielfältige Erweiterungsmöglichkeiten. Aufgrund ihres Grundlagencharakters kann sie daher für verschiedene Zwecke verwendet und erweitert werden, wie Abschnitt 10.2 beschreibt.

10.2 Ausblick

Im vorherigen Abschnitt wurde der Grundlagencharakter der Logging Infrastruktur angesprochen. Die Logging Infrastruktur bietet dabei eine Lösung für die grundlegende Problematik der Log Integration und verteilten Konfiguration der Logger, was als Basisfunktionalität für verschiedene Aufgaben im Bereich der Log Daten Analyse und Konfigurationsverwaltung verwendet werden kann, die dieser Abschnitt darstellt.

Proaktive Konfiguration Durch Lösung der verteilten Konfigurationsproblematik, lässt sich die Konfigurationsfähigkeit der Logging Infrastruktur zur proaktiven Konfiguration der Logger verwenden. Abhängig von einem bestimmten Stimulus (z.b. von Log-Einträgen, deren Log Level einen bestimmten Schwellenwert überschreitet), könnte die Logging Infrastruktur das Log Level der anderen Logger reduzieren, um im Fehlerfall detaillierte Information zur Fehlerursache bereitstellen zu können. Im

Betrieb ließe sich so ein guter Kompromiss zwischen Informationsbefriedigung und Datenaufkommen erzielen.

Monitoring Integration In Kooperation mit dem EMI Monitoring System ([Yük13]) könnten die Informationsbedürfnisse der EMI Stakeholder hinsichtlich des Monitorings besser befriedigt werden. Abhängig vom Status der EMI Komponenten könnte das Visualisierungswerkzeug beispielsweise zur schnellen Ursachenidentifizierung kritische Log-Einträge von ausgefallen Komponenten hervorheben oder aggregriert darstellen. Das Monitoring könnte mit Heuristiken die Zustandsdetektierung verbessern und Loggingspezifische Zustände einführen. Produziert eine Komponente innerhalb kürzester Zeit nur als Fehler klassifizierte Log-Einträge, könnte das Monitoring einen wahrscheinlichen Ausfall der EMI Komponente indizieren.

Logging Verbesserungen Da die realisierte Logging Infrastruktur dem Entwickler die Erzeugung der Log-Einträge überlässt und die Studie von Ding Yuan et al [YPH+12] zeigt, dass in vielen Fehlerfällen keine relevanten Log Daten vorliegen, könnte die Logging Infrastruktur mithilfe von ErrLog [YPH+12] oder AutoLog [ZGW+11] Mechanismen zur Verbesserung der Logging Anweisungen anbieten oder ein automatisiertes Logging ermöglichen.

Log Management Die Logging Infrastruktur kann zu einer Log Management Infrastruktur ausgebaut werden. Eine Log Management Infrastruktur [KS06] bietet dabei erweiterte Funktionalitäten zur Speicherung der Log-Einträge und Möglichkeiten zur Analyse der Logs.

Log Speicherung Erweiterte Funktionalitäten hinsichtlich der Speicherung umfassen die **Normalisierung**, **Komprimierung**, **Reduzierung** und **Archivierung** von Log-Einträgen.

Log Analyse Die Log Analyse umfasst Möglichkeiten zur Untersuchung und Auswertung der gespeicherten Logs. Insbesondere umfasst die Log Analyse dabei die Event Korrelation, d.h. die Untersuchung auf Zusammenhänge innerhalb der Log-Daten. Da die Logging Infrastruktur im Umfeld einer EMI eingesetzt wird, könnten Metriken zur Vermessung der Log-Einträge entwickelt werden. Diese Metriken bzw. die Resultate der Analyse ließen sich zwecks **Reporting** oder innerhalb eines Dashboards in SCREEN verwenden.

Konfigurationsplattform Aufgrund des Variabilitätsansatzes und des erweiterbaren Merge Service, ließe sich die Logging Infrastruktur zu einer Konfigurationsplattform für eine gesamte EMI ausbauen, um insbesondere die Konfigurationsverwaltung innerhalb der EMI zu erleichtern.

ELK Stack Integration Durch die Konfiguration eines weiteren Appenders können die Logs einer EMI leicht in Log Analyse Werkzeugen wie den im Rahmen der verwandten Arbeiten betrachteten ELK Stack oder Splunk verwendet werden. Dieser Schritt wurde aus Aufwandsgründen im Rahmen dieser Bachelorarbeit nicht mehr durchgeführt, da am Lehrstuhl keine ELK oder Splunk Instanz vorhanden ist.

A Anhang

A.1 Variabilitätsmodell

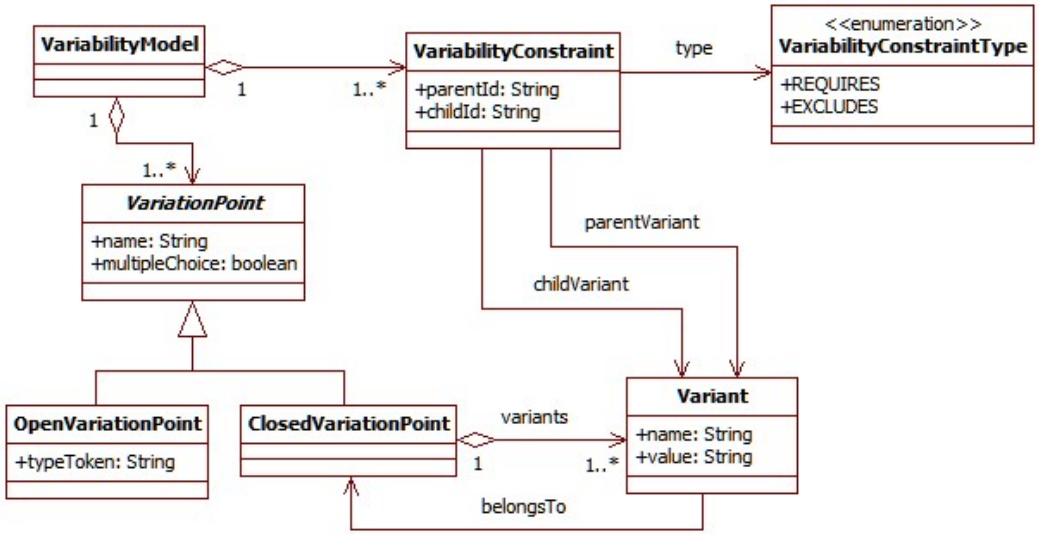

Abbildung A.1: Variabilitätsmodell von Thomas Rölling [Röl14]

A.2 Variabilitätskonfigurationsmodell

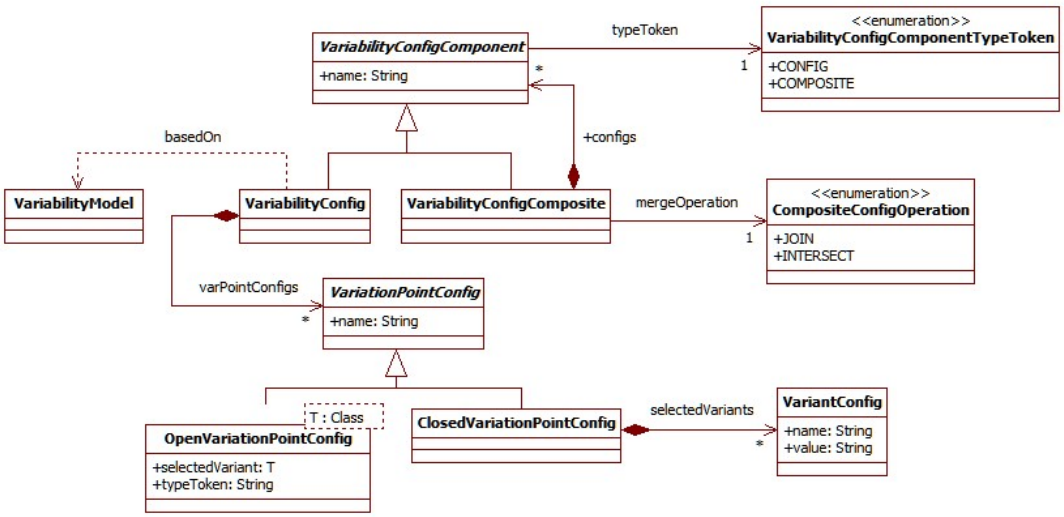

Abbildung A.2: Variabilitätskonfigurationsmodell von Thomas Rölling [Röl14]

Literaturverzeichnis

[Cha04] D.A. Chappell. *Enterprise Service Bus: Theory in Practice*. Theory in practice. O'Reilly Media, 2004.

[Cha13] Christian Charles. Entwurf eines generischen Prozessleitstandes für Change Request Systeme. Diplomarbeit, RWTH Aachen University, 2013.

[CP10] Anton Chuvakin and Gunnar Peterson. How to do application logging right. *IEEE Security & Privacy*, 8(4):82–85, 2010.

[CSP13] Anton Chuvakin, Kevin Schmidt, and Chris Phillips. *Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management*. Syngress Publishing, 2013.

[Dai11] Robert Daigneau. *Service Design Patterns*. Addison-Wesley Professional, 1st edition, 2011.

[Ela14] Elasticsearch. Logstash 1.4.2 documentation. `http://logstash.net/docs/1.4.2/`, October 2014.

[Erl09] Thomas Erl. *SOA Design Patterns*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2009.

[FGR+97] John Foreman, Jon Gross, Robert Rosenstein, David Fisher, and Kimberly Brune. *C4 Software Technology Reference Guide: A Prototype*. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, handbook cmu/sei-97-hb-001 edition, October 1997.

[FLL+13] Qiang Fu, Jian-Guang Lou, Qingwei Lin, Rui Ding, Dongmei Zhang, and Tao Xie. Contextual analysis of program logs for understanding system behaviors. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 397–400, Piscataway, NJ, USA, 2013. IEEE Press.

[FLWL09] Qiang FU, Jian-Guang LOU, Yi WANG, and Jiang LI. Execution anomaly detection in distributed systems through unstructured log analysis. In *International conference on Data Mining (full paper)*. IEEE, December 2009.

[Fow02] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[FZH+14] Qiang Fu, Jieming Zhu, Wenlu Hu, Jian-Guang Lou, Rui Ding, Qingwei Lin, Dongmei Zhang, and Tao Xie. Where do developers log? an empirical study on logging practices in industry. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 24–33, New York, NY, USA, 2014. ACM.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*, volume 47 of *Addison Wesley Professional Computing Series*. 1995.

[Gji13] Endri Gjino. Visually assisted mining for smells in change request systems. Master's thesis, RWTH Aachen University, 2013.

[GKG+09] Kirk Glerum, Kinshuman Kinshumann, Steve Greenberg, Gabriel Aul, Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt. Debugging in the (very) large: Ten years of implementation and experience. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 103–116, New York, NY, USA, 2009. ACM.

[HoEE04] J. Holt and Institution of Electrical Engineers. *UML for Systems Engineering: Watching the Wheels, 2nd Edition*. IEE professional applications of computing series. Institution of Engineering and Technology, 2004.

[Hor14] Thorsten Horn. Eai enterprise application integration. `http://www.torsten-horn.de/techdocs/eai.htm#Integrationstopologien`, November 2014.

[HW03] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[IEE90] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[ISO06] ISO. Grundsätze der dialoggestaltung. ISO 9241-110, International Organization for Standardization, 2006.

[KS06] Karen Kent and Murugiah P. Souppaya. Sp 800-92. guide to computer security log management. Technical report, Gaithersburg, MD, United States, 2006.

[kvG14] karthik-vn (Github). Blitz4j at a glance. `https://github.com/Netflix/blitz4j/wiki/Blitz4j-at-a-glance`, October 2014.

[LFY+10] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. Mining invariants from console logs for system problem detection. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical*

Conference, USENIXATC'10, pages 24–24, Berkeley, CA, USA, 2010. USENIX Association.

[LL10] J. Ludewig and H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken.* dpunkt-Verlag, 2010.

[LVS14] Horst Lichter, Matthias Vianden, and Andreas Steffens. Experience on a microservice-bases reference architecture for measurement systems. In *Proceedings of the 21st Asian Pacific Conference on Software Engineering (APSEC)*, December 2014. to be published.

[Mäd13] Martin Mädler. Variabilität von Metriken und Dashboard-Items im Umfeld von MeDIC. Diplomarbeit, RWTH Aachen University, June 2013.

[Mey92] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992.

[NKN12] Karthik Nagaraj, Charles Killian, and Jennifer Neville. Structured comparative analysis of systems logs to diagnose performance problems. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 26–26, Berkeley, CA, USA, 2012. USENIX Association.

[Röl14] Thomas Rölling. Konfigurationsunterstützung und Infrastrukturansatz für flexible metrik-basierte Konfigurationsunterstützung und Konfigurationsunterstützung und Infrastrukturansatz für flexible metrik-basierte Monitoring-Dashboards. Master's thesis, RWTH Aachen University, Feburary 2014.

[Rum11] Bernhard Rumpe. *Modellierung mit UML.* Springer-Verlag, 2 edition, 2011.

[SCH+11] Bikash Sharma, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat, and Chita R. Das. Modeling and synthesizing task placement constraints in google compute clusters. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 3:1–3:14, New York, NY, USA, 2011. ACM.

[SHLP05] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The enterprise service bus: Making service-oriented architecture real. *IBM Systems Journal*, 44(4):781–797, 2005.

[Ste13] Andreas Steffens. Entwurf eines Architekturmodells zur Integration heterogener Systeme in MeDIC. Diplomarbeit, RWTH Aachen University, 2013.

[Tav11] Ricardo Tavizon. Systematic tool supported tailoring of metrics. Master's thesis, RWTH Aachen University, 2011.

[vHWH12] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 247–248, New York, NY, USA, 2012. ACM.

[Via15] Matthias Vianden. *A Goal-Oriented Metric Systems Engineering Approach.* PhD thesis, RWTH Aachen University, 2015. to be published.

[WJ95] Michael Wooldridge and Nicholas R. Jennings. Agent theories, architectures, and languages: A survey. In *Proceedings of the Workshop on Agent Theories, Architectures, and Languages on Intelligent Agents*, ECAI-94, pages 1–39, New York, NY, USA, 1995. Springer-Verlag New York, Inc.

[YPH+12] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M. Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. Be conservative: Enhancing failure diagnosis with proactive logging. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 293–306, Berkeley, CA, USA, 2012. USENIX Association.

[Yük13] Ahmet Yüksektepe. Entwicklung einer Service-Monitoring-Infrastruktur für die EMI, 2013.

[ZGW+11] Cheng Zhang, Zhenyu Guo, Ming Wu, Longwen Lu, Yu Fan, Jianjun Zhao, and Zheng Zhang. Autolog: Facing log redundancy and insufficiency. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, APSys '11, pages 10:1–10:5, New York, NY, USA, 2011. ACM.