Masterarbeit

Integration von Informationssystemen:

Datenaustausch zwischen Teamcenter und Enterprise Applikationen

eingereicht an der
Montanuniversität Leoben

erstellt am
Lehrstuhl für Informationstechnologie

Vorgelegt von:
Klemens Wötzl

Betreuer:
Univ.-Prof. Dipl.-Ing. Dr.techn. Peter Auer

Leoben, 8. Juni 2015

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Leoben, 8. Juni 2015

Klemens Wötzl

Danksagung

Ein ganz besonderer Dank, gilt meinen Eltern Christina und Leopold Wötzl. Sie haben mir ermöglicht meinen eigenen Weg im Leben zu finden, ohne sie wäre mein Studium nicht möglich gewesen.

Außerdem gilt mein Dank meiner Freundin Sophie Strobl, die mich bei Rückschlägen und auch am späten Abend stets motiviert und aufgeheitert hat. Sie hatte immer ein offenes Ohr und musste auf viel gemeinsame Zeit verzichten.

Des weiteren danke ich meinen Freunden und Kollegen die mich unterstützt und entlastet haben und es mir auch nicht nachtragen, dass ich zuletzt wenig Zeit für sie hatte.

Abschließend möchte ich mich noch bei Univ.-Prof. Dr. Peter Auer bedanken, der mich bei der Erstellung dieser Diplomarbeit unterstützt hat und mir die Möglichkeit gab die Diplomarbeit nach meinen eigenen Vorstellungen zu entwicklen.

Kurzfassung

Product Lifecycle Management (PLM) Systeme wie TEAMCENTER sind in vielen Unternehmen ein fester Bestandteil der IT-Systemlandschaft. Ein Ziel von PLM Systemen ist das Verwalten von Produktdaten über den gesamten Lebenszyklus. Um dieses Ziel zu erreichen, ist ein Datenaustausch mit anderen Informationssystemen unerlässlich. Die vorliegende Arbeit zeigt, wie eine flexible Lösung zur Kopplung von TEAMCENTER - dem PLM System von Siemens Product Lifecycle Management Software Inc. - mit weiteren Anwendungen realisiert werden kann. Der erste Teil dieser Arbeit beschäftigt sich mit den Herausforderungen der *Enterprise Application Integration* und den aktuell verfügbaren Integrationslösungen. Anschließend wird im zweiten Teil TEAMCENTER vorgestellt. Im Hauptteil der Arbeit wird gezeigt, wie Integrationslösungen für TEAMCENTER mit Hilfe des APACHE CAMEL Frameworks umgesetzt werden können. Zusätzlich werden ausgewählte Anwendungsfälle in Form von Fallbeispielen diskutiert. Dazu werden Lösungen mit Hilfe der *Enterprise Integration Pattern* erstellt und mit dem APACHE CAMEL Framework umgesetzt.

Abstract

Product Lifecycle Management (PLM) systems are a common component in the IT landscape of many companies. To manage products over the whole product lifecycle, a data exchange between PLM systems and other enterprise applications is essential. The objective of this thesis is to provide a flexible solution that integrates the PLM system of Siemens Product Lifecycle Management Software Inc, TEAMCENTER, with enterprise applications. The first part of this thesis discusses the challenges of enterprise application integration and gives an overview of the currently available integration solutions. In the second part, some basics of TEAMCENTER are summarized. The main part of the thesis demonstrates an integration solution for TEAMCENTER that is built with the APACHE CAMEL framework. Additionally, use cases examples are discussed. For each use case, enterprise integration patterns are used to design a solution, which is then realized in the APACHE CAMEL framework.

Inhaltsverzeichnis

1. Einleitung 1
1.1. Wissenschaftliche Fragestellung 1
1.2. Aufbau der Arbeit 1

2. Enterprise Application Integration 3
2.1. Loose Coupling 4
2.2. Messaging 5
2.2.1. Asynchronous Messaging 6
2.2.2. Messaging Systeme 7
2.2.3. Message Exchange Pattern 7
2.2.3.1. Request/Response 8
2.2.3.2. Publish/Subscribe 8
2.2.3.3. Broadcast/Multicast 9
2.3. Enterprise Integration Pattern 9
2.3.1. Grundlegende Elemente einer Integrationslösung 10
2.3.1.1. Message Channel 11
2.3.1.2. Message 11
2.3.1.3. Pipes and Filters 12
2.3.1.4. Message Router 12
2.3.1.5. Message Translator 12
2.3.1.6. Message Endpoint 13
2.3.2. Entscheidungshilfe bei der Routerwahl 13
2.4. Integrationsarchitektur 14
2.5. Service-Oriented Architecture 18
2.6. Bestehende Integrationslösungen 20

3. Teamcenter 23
3.1. Teamcenter Architektur 23
3.2. Teamcenter Datenmodell 26
3.2.1. Grundlegende Geschäftsobjekte 28
3.3. System Administration 29
3.4. Produktstrukturen 30
3.5. Automatisierung mit Workflows 31
3.6. Lizenzierung 31
3.7. Teamcenter Services 31
3.7.1. Voraussetzungen 33
3.7.2. Teamcenter Connection 33
3.7.3. Service Invocation 34

3.7.4. Fehlerhandling . . . 34
3.7.5. Client Data Model . . . 35
3.7.6. Object Property Policy . . . 37
3.7.7. Gliederung der Teamcenterservices . . . 37

4. Technologien und Frameworks **39**
4.1. Apache Camel . . . 39
4.1.1. Eine einfache Applikation . . . 42
4.1.2. Expressions und Predicates . . . 44
4.1.3. Type Converter . . . 44
4.1.4. Zusammenfassung . . . 44
4.2. Spring . . . 45
4.3. Java Message Service . . . 46
4.4. Webservices . . . 47
4.4.1. SOAP . . . 48
4.4.2. REST . . . 49

5. Entwicklung einer Integrationlösung für Teamcenter **50**
5.1. Auswahl eines Frameworks . . . 50
5.2. Apache Camel Komponente für Teamcenter . . . 51
5.2.1. Anforderungen . . . 51
5.2.2. Erstellung einer Projektstruktur . . . 53
5.2.3. TcComponent . . . 54
5.2.4. TcEndpoint . . . 57
5.2.5. TcProducer . . . 62
5.2.6. TcFacade . . . 64
5.3. Erweiterungen für den Einsatz der Teamcenterkomponente . . . 71
5.3.1. Custom Teamcenter Datenformat . . . 73
5.3.1.1. Default Type Converter für TcComponent . . . 73
5.4. Die erste Teamcenter Integration . . . 74

6. Fallbeispiele **79**
6.1. Fallbeispiel 1 - Artikelinformationen abfragen . . . 79
6.2. Fallbeispiel 2 - Artikelinformationen aktualisieren . . . 81
6.3. Fallbeispiel 3 - Erstellen von Objekten in Teamcenter . . . 85
6.4. Fallbeispiel 4 - Load Balancing . . . 86
6.5. Fallbeispiel 5 - Persistenz mit JMS . . . 88
6.6. Fallbeispiel 6 - Exceptions and Error Handling . . . 90

7. Fazit **92**
7.1. Erstellen einer Integrationslösung . . . 93

7.2. Ausblick 94

A. Erweiterte Optionen für die TcComponent 98

B. Weiterführende Listings für TcComponent 102

C. Weiterführende Dokumente zu den Fallbeispielen 106

Abbildungsverzeichnis

1. Request/Response MEP . . . 8
2. Publish/Subscribe MEP . . . 8
3. Grundlegende Elemente einer Integrationslösung Hohpe and Woolf [2012] 10
4. Message Channel . . . 11
5. Message . . . 11
6. Pipes and Filters . . . 12
7. Message Router . . . 12
8. Message Translator . . . 12
9. Message Endpoint . . . 13
10. Übersicht Routerpattern Hohpe and Woolf [2012] . . . 14
11. Point to Point . . . 15
12. Hub and Spoke . . . 16
13. Message Bus . . . 17
14. Enterprise Service Bus nach Oracle . . . 18
15. 4-Tier Architektur . . . 24
16. File Management System . . . 25
17. Mögliches Deployment für Lastverteilung . . . 26
18. Auszug aus der Teamcenter Klassen Hierarchie . . . 28
19. System administration model Siemens PLM Software [2014b] . . . 30
20. Exchange in Apache Camel . . . 41
21. Arten von Consumer . . . 42
22. MyComponent Projektstruktur . . . 53
23. Vereinfachtes Klassendiagram der Teamcenter Komponente für Camel . . 57
24. Auszug aus Klassendiagramm für TcProducer . . . 62
25. Fallbeispiel 1 . . . 80
26. Fallbeispiel 2 . . . 81
27. Fallbeispiel 3 . . . 85
28. Fallbeispiel 4 . . . 87
29. Fallbeispiel 5 . . . 89
30. Fallbeispiel 6 . . . 90

Tabellenverzeichnis

1. Tighty vs Loose Coupling nach Kaye [2003] 4
2. Teamcenter Server Adresse 55
3. URI Parameter für TcComponent 56
4. Erlaubte Werte für URI Option tcMethod - Teil 1 59
5. Erlaubte Werte für URI Option tcMethod - Teil 2 60
6. Allgemeine Optionen für TcComponent 63
7. Erweiterte Optionen für TcComponent - Teil 1 98
8. Erweiterte Optionen für TcComponent - Teil 2 99
9. Erweiterte Optionen für TcComponent - Teil 3 100
10. Erweiterte Optionen für TcComponent - Teil 4 101

Abkürzungsverzeichnis

AOP .. Aspect Oriented Programming
API ... Application Programming Interface
BMIDE Business Modeler Integrated Development Environment
BOM .. Bill of Materials
CAD .. Computer-aided design
CAM .. Computer-aided manufacturing
CDM .. Client Data Model
CORBA Common Object Request Broker Architecture
DI ..... Dependency Injection
DSL ... Domain-specific language
EAI ... Enterprise Application Integration
EIP ... Enterprise Integration Pattern
ESB ... Enterprise Service Bus
FCC .. File Client Cache
FMS .. File Management System
FQN .. Fully Qualified Name
FSC ... File Server Cache
HTML Hypertext Markup Language
HTTP . Hypertext Transfer Protocol
IIS .... Microsoft Internet Information Services
IoC .... Inversion of Control
IP ..... Internet Protocol
J2EE .. Java Platform Enterprise Edition
JAR ... Java Archive
JAXB . Java Architecture for XML Binding
JDBC . Java Database Connectivity
JMS ... Java Message Service
JSON . JavaScript Object Notation
MEP .. Message Exchange Pattern
MOM . Message Oriented Middleware
MVC .. Model-View-Controller
PLM .. Product Lifecycle Management
POJO . Plain Old Java Object
POM .. Persistent Object Manager
REST . Representational State Transfer
RPC .. Remote Procedure Call
SOA .. Service Oriented Architecture
TCP .. Transmission Control Protocol
URI ... Uniform Resource Identifier
WAN .. Wide Area Network
WSDL Web Services Description Language
XML .. Extensible Markup Language

YAML YAML Ain't Markup Language

1. Einleitung

Die Komplexität in der Produktentwicklung steigt heutzutage stetig an. Kürzere Entwicklungszyklen, höhere Qualitätsanforderungen, genauere Kundenausrichtung und große Mengen an Produktdaten stellen Unternehmen vor neue Herausforderungen. Um diese neuen Anforderungen bewältigen zu können und einen Wettbewerbsvorteil zu erlangen, werden Product Lifecycle Management (PLM) Systeme wie TEAMCENTER in vielen Unternehmen eingesetzt. Jedoch ist ein PLM System mehr als nur ein Informationssystem, es ist ein strategischer Ansatz zum Verwalten von Produkten über ihren gesamten Lebenszyklus hinweg. Um diese Durchgängigkeit erreichen zu können, ist ein Datenaustausch mit anderen Informationssystemen notwendig.

Die Firma ACAM Systemautomation[1], ein Systemanbieter von PLM Software, hat regelmäßig die Anforderung, Informationssysteme mit TEAMCENTER, dem PLM System von Siemens, zu koppeln. TEAMCENTER bietet eine offene Plattform, mit der Fremdsysteme integriert werden können. Diese Kopplung ist mit entsprechendem Aufwand verbunden, sodass viele Kunden davor zurück schrecken. Jedoch sind viele der Anforderungen für eine Systemkopplung wiederkehrend. Auch brauchen viele Fremdsysteme für die Kopplung nur einen kleinen Teilbereich der angebotenen Funktionen, müssen aber trotzdem die komplexen Dienste von TEAMCENTER für die Kopplung implementieren. Eine generische, einfach anpassbare Integrationslösung würde für viele Kunden einen erheblichen Mehrwert darstellen.

Diese Arbeit zeigt, welche Softwarekomponenten für eine Anbindung von Teamcenter an Informationssysteme benötigt werden und wie diese erstellt werden können. Auf Basis dieser Softwarekomponenten wird eine Anwendung erstellt, die Daten zwischen Teamcenter und einem Fremdsystem austauschen kann. Bei der Realisierung der Anwendung wird ein Schwerpunkt auf die einfache Anbindung und Erweiterung gelegt.

1.1. Wissenschaftliche Fragestellung

Ziel der Arbeit ist es zu zeigen, wie eine flexible Lösung für Integrationsaufgaben realisiert werden kann, mit der Teamcenter und weitere Enterprise Applikationen gekoppelt werden können. Die Lösung soll mit geringem Anpassungsaufwand in unterschiedlichen Szenarien einsetzbar sein.

1.2. Aufbau der Arbeit

Der erste Teil der Arbeit beschäftigt sich mit den Herausforderungen der Anwendungsintegration. Zuerst wird der aktuelle Stand der Forschung beschrieben. Dann werden die

[1] http://www.acam.at

bekanntesten Frameworks und Services für die Anwendungsintegration vorgestellt und die wichtigsten Unterschiede hervorgehoben.

Der zweite Teil beschäftigt sich mit Teamcenter, dem PLM System von Siemens Product Lifecycle Management Software Inc. Neben der historischen Entwicklung von Teamcenter, werden der Aufbau, die grundlegenden Prinzipen, sowie wichtige Funktionen und die Möglichkeiten zur Eingliederung von Teamcenter in die Systemlandschaft eines Unternehmens diskutiert.

Bevor mit der Realisierung einer Integrationslösung begonnen wird, werden im dritten Teil der Arbeit einige Technologien und Frameworks zum Betreiben und Implementieren von Integrationslösungen vorgestellt. Dazu gehören unter anderem Apache Camel, das Spring Framework, das Java Messaging Service und Webservices.

Im Hauptteil der Arbeit wird ein Konzept zur Integration von Teamcenter mit Fremdsystemen vorgestellt und begründet, warum Apache Camel als Basis für die Realisierung gewählt wurde. Um eine Interaktion zwischen Apache Camel und Teamcenter zu ermöglichen, wird das Apache Camel Framework um eine Komponente für Teamcenter erweitert und dessen Einsatz demonstriert. Abschließend werden in Form von Fallbeispielen typische Szenarien für die Integration von Teamcenter mit anderen Systemen diskutiert und Lösungen mit Apache Camel und der Teamcenterkomponente vorgestellt.

2. Enterprise Application Integration

Enterprise Application Integration (EAI) ist kein Produkt oder spezielles Framework, sondern beschreibt die Kombination von Prozessen, Software, Standards und Hardware um mehrere Anwendungen in einem Unternehmen miteinander zu verknüpfen (Lam [2007]). Anders ausgedrückt ist EAI die Aufgabe, heterogene Systeme zu verknüpfen, um eine Menge von einheitlichen Funktionen zu realisieren.

Aus technischer Sicht ist EAI ein Prozess, der Geschäftsfunktionen und Daten heterogener Anwendungen miteinander verknüpft, sodass die Daten in allen verbundenen Anwendungen zur Verfügung stehen. Eine solche Integration sollte, wie in Schmutz et al. [2010] beschrieben wird, möglichst ohne große Eingriffe in die einzelnen Anwendungen erreicht werden. Immer mehr Unternehmen erkennen den Wettbewerbsvorteil, den sie erringen können, indem ihre Anwendungen zu einem großen Informationssystem integriert werden. Dadurch ist in den letzten Jahren der Bedarf für EAI stetig gestiegen.

In Schmutz et al. [2010] werden drei grundlegende Arten von Integrationsaufgaben beschrieben.

- *Information Portals*: Informationen aus unterschiedlichen Anwendungen werden an einer Stelle angezeigt.
- *Shared Data*: Mehrere Anwendungen nutzen eine gemeinsame Datenresource.
- *Shared Business Function*: Mehrere Anwendungen teilen Funktionen miteinander.

Selten existieren Anwendungen in Unternehmen in Isolation. Sobald es notwendig wird Anwendungen mit Anderen kommunizieren zu lassen, entstehen grundlegende Herausforderungen wie Datenübertragung oder Datentransformation. Oft müssen Informationen über ein Netzwerk von einem Computer, zu einem Anderen übertragen werden. An einem Netzwerk sind meist viele unterschiedlichen Komponenten beteiligt. In jeder Komponente können Probleme auftreten, die zu Verzögerungen oder Unterbrechungen führen können. Zusätzlich ist eine Datenübertragung über ein Netzwerk ist wesentlich langsamer als von einem lokalen Datenträger. Eine weitere Herausforderung sind die unterschiedlichen Datenformate der verschiedenen Anwendungen. Auch muss die Integration berücksichtigen, dass sich Anwendungen durch Funktionserweiterungen oder Verbesserungen verändern. Wenn sich eine Anwendung ändert, kann auch eine andere Anwendung beeinflusst werden. Um die vielen Herausforderungen der Anwendungsintegration überwinden zu können, wurden über die Zeit verschiedene Ansätze entwickelt. In Hohpe and Woolf [2012] werden die Folgenden beschrieben.

- File Transfer
- Shared Database
- Remote Procedure Invocation
- Messaging

Alle Ansätze versuchen das gleiche Problem zu lösen, haben aber unterschiedliche Stärken und Schwächen. Aus diesem Grund werden in Integrationslösungen oft mehrere Ansätze gleichzeitig genutzt und für jeden Teilbereich des Integrationsproblems der am besten passende Ansatz verwendet. *File Transfer* und *Shared Database* ermöglichen Anwendungen, ihre Daten, aber nicht ihre Funktionen zu teilen. Mit *File Transfer* können Anwendungen lose aneinander gebunden werden, jedoch ist der Datenaustausch weniger performant. *Shared Database* ermöglicht einen performanten Zugriff, bindet Anwendungen aber stark an die Datenbank. Im Gegensatz zu den ersten zwei Ansätzen, ermöglicht *Remote Procedure Invocation* das Teilen von Funktionen zwischen Anwendungen, bindet diese jedoch gleichzeitig stark aneinander. *Remote Procedure Invocation* bring noch weitere Probleme mit sich, dazu gehören auch essenzielle Probleme von verteilen Anwendungen. Sie sind langsamer und können fehlschlagen. *Messaging* ermöglicht den verlässlichen und asynchronen Transfer von Datenpaketen, in einem anpassbaren Datenformat, ohne Anwendungen stark aneinander zu binden. Jedoch entstehen auch mit der Verwendung von *Messaging* neue Hürden und Herausforderungen.

2.1. Loose Coupling

Loose Coupling ist ein Ansatz der erlaubt unabhängige Anwendungen zu verknüpfen und ist im Gegensatz zu den *Tightly Coupled* Anwendungen, unempfindlicher gegen unerwartete Ereignisse oder Änderungen. Die Idee von *Loose Coupling* ist es, die Abhängigkeiten zwischen Anwendungen zu reduzieren. Lose gekoppelte Anwendungen sind von internen Änderungen, der jeweilig anderen Anwendung, entkoppelt. In Kaye [2003] wird *Loose Coupling* als eine Methode beschrieben um agile, anpassbare Anwendungen zu entwickeln.

	Tightly Coupled	**Loosely Coupled**
Interaction	Synchronous	Asynchronous
Messaging Style	RPC	Document
Message Paths	Hard Coded	Routed
Technology Mix	Homogeneous	Heterogeneous
Data Types	Dependent	Independent
Syntactic Definition	By Convention	Published Schema
Software Objective	Re-use, Efficiency	Broad Applicability

Tabelle 1: Tightly vs Loose Coupling nach Kaye [2003]

In Tabelle 1 werden einige der in Kaye [2003] beschriebenen Unterschiede zwischen lose und stark gekoppelten Anwendungen aufgelistet. Zwar können verteilte Systeme auch erfolgreich mit stark gekoppelten Anwendungen realisiert werden, jedoch erläutert Kaye [2003] wie diese Anwendungen durch lose Kopplung verbessert werden können und wie vorallem die folgenden Methoden zu lose gekoppelten Anwendungen führen.

- Asynchronous Messaging
- Document-Style Messaging
- Delayed Bindings and Published Schema

2.2. Messaging

In Hohpe and Woolf [2012] wird *Messaging*, als Technologie beschrieben, die es Anwendungen ermöglicht miteinander zu kommunizieren. Mit Messaging können Daten schnell, asynchron und verlässlich ausgetauscht werden. Die Kommunikation funktioniert, indem eine *Message* (Datenpaket) von einem *Producer* (Versender) über einen *Channel* an einen oder mehrere *Consumer* (Empfänger) versendet wird. Ein *Channel* ist eine Sammlung von Messages und kann gleichzeitig von mehreren Anwendungen genutzt werden (siehe Abschnitt 2.3.1.1). Producer können Messages zu einem Channel hinzufügen, Consumer können Messages aus dem Channel lesen und entfernen. In Folge werden einige der in Hohpe and Woolf [2012] beschrieben Vorteile von Messaging gegenüber anderen Arten des Datenaustauschs aufgelistet.

- *Remote Communication.* Messaging ermöglicht entfernten Anwendungen zu kommunizieren und Daten auszutauschen.
- *Platform/Language Integration.* Werden verteilte Systeme/Anwendungen integriert, so ist es sehr wahrscheinlich, dass diese unterschiedliche Sprachen, Technologien und/oder Plattformen verwenden. *Messaging Systeme* können als Übersetzer (Mediator) dienen.
- *Asynchronous Communication.* Der Producer (Sender) muss nicht auf eine Antwort des Consumer (Empfängers) warten. Es muss nur gewartet werden, bis die Nachricht erfolgreich versendet wurde, sprich dem Messaging System übergeben wurde. Dieses ist für die verlässliche Zustellung der Nachricht zuständig.
- *Variable Timing.* Die asynchrone Kommunikation ermöglicht dem Producer Nachrichten zu versenden, ohne auf Antworten warten zu müssen. Consumer können anschließend die Nachrichten in einer beliebigen Geschwindigkeit bearbeiten ohne den Producer zu blockieren.
- *Throttling.* Ein Problem von RPC ist, dass zu viele Anfragen zu einem Zeitpunkt einen Empfänger überfordern können. Durch asynchrone Kommunikation können Empfänger steuern mit welcher Geschwindigkeit Anfragen verarbeitet werden.
- *Reliable Communication.* Im Gegensatz zu RPC kann Messaging die Auslieferung von Nachrichten garantieren. Messaging verwendet einen *store-and-forward* Ansatz bei der Übermittlung von Nachrichten. Die Nachricht wird im Messaging System gespeichert und an den Empfänger übergeben, die Übergabe wird solange wiederholt bis sie erfolgreich war.

- *Disconnected Operation.* Manche Anwendungen sollen ohne ständige Netzwerkverbindungen arbeiten, jedoch sich von Zeit zu Zeit mit einer Gegenstelle synchronisieren. Durch die Verwendung von Messaging können die zu synchronisierenden Daten in einer Warteschlage eingereiht werden, sobald eine entsprechende Verbindung vorhanden ist, werden die Nachrichten übermittelt.
- *Mediation.* Messaging Systeme agieren als Mediatoren vergleichbar mit den Designmuster Vermittler in Gamma et al. [2004] und ermöglichen High Availability von Resourcen, Load Balancing, Rerouting, Redelivery, Quality of Service und vieles mehr.

Messaging bietet viele Vorteile, bringt jedoch auch neue Herausforderungen mit sich. So wird für die Kommunikation ein Messaging System benötigt, fällt dieses aus ist keine Kommunikation mehr möglich. Zusätzlich führt der Einsatz von Messaging Systemen zu einem Overhead bei der Kommunikation. Weitere Herausforderungen entstehen, wie in Abschnitt 2.2.1 beschrieben, durch den Einsatz von Asynchronous Messaging.

2.2.1. Asynchronous Messaging

In Ramanathan [2013] wird ein Überblick über Asynchronous Messaging und der zugehörigen Mechaniken gegeben. Im Folgenden werden einige dieser Punkte aufgegriffen und kurz beschrieben. In Unternehmen ist die Laufzeit von Prozessen mitunter sehr lange und oft sind manuelle Bestätigungsschritte erforderlich. So können Tage vergehen bevor ein Prozess abgeschlossen wird. Synchrone Anwendungen würden Zeitüberschreitungen melden und können in diesen Fällen nicht ohne weiteres eingesetzt werden. Asynchrone Services hingegen können verwendet werden um diesen Anforderungen zu begegnen. Dabei wird eine Anfrage an einen Service gestellt und anschließend der Prozess fortgesetzt, ohne darauf zu warten, dass die Anfrage verarbeitet wurde. Die Ergebnisse werden zu einem späteren Zeitpunkt asynchron empfangen. Um Asynchronous Messaging zu ermöglichen wird eine Art von Wartenschlange (*Queue*) für Anfragen benötigt. Anfragen werden an diese Queue gesendet und nicht direkt an den Service. Zusätzlich wird eine Methode zur Korrelation benötigt, um später die Antwort mit der ursprünglichen Anfrage verknüpfen zu können. Wenn ein Service bereit ist eine Anfrage zu verarbeiten, wird eine Nachricht aus der Warteschlange entnommen und verarbeitet. So werden Anwendungen von einander entkoppelt und es kann verhindert werden, dass Fehler in einer Anwendung oder bei der Übertragung der Nachricht, eine andere Anwendung beeinflussen. Zusätzlich wird durch die Entkopplung der Anwendungen, die Skalierbarkeit vereinfacht.

Hohpe and Woolf [2012] beschreiben Vorteile eines Asynchronen Datenaustausches und wie durch den Einsatz viele Probleme bei der Integration von verteilten Systeme umgegangen werden können Jedoch entstehen durch den Einsatz von Asynchronous Messaging neue Herausforderungen. So wird für die Implementierung von Asynchronous Messaging

die Verwendung von *event-driven programming* benötigt. Solche Systeme sind komplexer, schwieriger zu entwickeln und aufwendiger zu warten. Eine weitere Herausforderung ergibt sich bezüglich der Reihenfolge der Nachrichten. Message Channel garantieren zwar die Übertragung, jedoch nicht wann eine Nachricht übergeben wird. So kann die ursprüngliche Reihenfolge von Nachrichten nicht garantiert werden. Wenn die Reihenfolge der Nachrichten eine Rolle spielt, muss diese wieder hergestellt werden. Außerdem entsteht durch Messaging Systeme bei der Kommunikation ein Overhead und mit dieser zusätzlichen Belastung muss umgegangen werden.

2.2.2. Messaging Systeme

Die nötigten Funktionen für die Übertragung von Nachrichten, werden typischerweise von *Messaging Systemen* oder *Message-oriented Middleware* (MOM) bereitgestellt. Die Hauptaufgabe von Messaging Systemen ist die Übertragung der Nachrichten vom Versender zum Empfänger. Ab dem Zeitpunkt der Übergabe einer Nachricht an ein Messaging System übernimmt dieses die Verantwortung für die Nachrichtenübertragung. Dadurch wird die Komplexität der Übertragung vom Versender und Empfänger entkoppelt. Messaging Systeme sorgen für eine verlässliche Auslieferung der Nachricht.

2.2.3. Message Exchange Pattern

Services und speziell WebServices bauen darauf auf, dass Nachrichten kombiniert und ausgetauscht werden. Über die Zeit wurde eine Menge von wiederkehrenden Mustern identifiziert, die unter dem Namen *Message Exchange Patterns* (MEPs) bekannt sind. In Kaye [2003] wird beschrieben, wie die MEPs durch die zwei Kriterien, *Synchronisation* und *Korrelation*, kategorisiert werden können .

Synchronisation. Wenn Endpunkte Nachrichten austauschen, gibt es drei Möglichkeiten wie der Austausch durchgeführt wird.

- *Synchron.* Nach dem Senden einer Nachricht, wartet der Endpunkt bis eine Antwort empfangen wird.
- *Asynchron.* Nach dem Senden einer Nachricht erwartet der Endpunkt zwar eine Antwort, wartet aber nicht auf diese. Die Antwort kann zu einem belieben Zeitpunkt empfangen werden.
- *Fire-and-forget.* Der Versender der Nachricht erwartet keine Antwort.

Korrelation. Werden in einer einzelnen Transaktion mehrere Nachrichten versendet, gibt es drei Möglichkeiten wie diese assoziiert werden können

- *Keine Korrelation.* Alle Nachrichten sind selbständig, es müssen keine Nachrichten kombiniert werden.

- *Korrelation über die Transportschicht.* Transportprotokolle wie HTTP übernehmen die Korrelation von zusammenhängenden Nachrichten.
- *Korrelation über die Nachricht.* Die Nachricht oder der Nachrichtenkopf (Header) beinhalten die Information wie Nachrichten kombiniert werden sollen.

In Folge werden einige MEPs beschrieben.

2.2.3.1. Request/Response Das *Request/Response* Pattern ist einer der einfachsten MEPs, es kann sowohl asynchron, als auch synchron implementiert werden. Da nur zwei Nachrichten (Anfrage und Antwort) involviert sind, kann eine einfache Korrelation verwendet werden, meist wird HTTP verwendet um Anfrage und Antwort zuzuordnen.

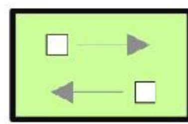

Abbildung 1: Request/Response MEP

2.2.3.2. Publish/Subscribe Bei *Publish/Subscribe* sendet ein *Subscriber* (Abonnent) eine Anfrage an einen *Publisher* um bei gewissen Ereignissen informiert zu werden. Jedes Mal wenn das Event auftritt, sendet der Publisher eine Nachricht an den Abonnenten, erwartet aber keine Antwort von den Subscribern (*Fire-and-forget*). Da keine Antwort erwartet wird, wird auch keine Korrelation benötigt.

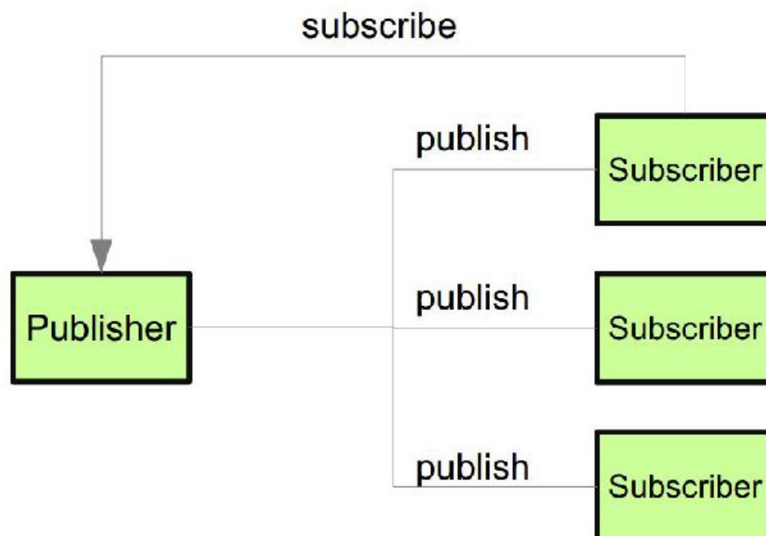

Abbildung 2: Publish/Subscribe MEP

2.2.3.3. Broadcast/Multicast Bei *Broadcast* wird eine Nachricht an alle Beteiligten eines Netzwerks geschickt. *Mutlicast* arbeitet ähnlich, nur wird die Nachricht nicht an alle im Netzwerk, sondern nur an eine ausgewählte Gruppe von Beteiligten gesendet. Auch bei diesem Pattern wird der *Fire-and-forget* Ansatz verwendet und somit keine Korrelation benötigt.

2.3. Enterprise Integration Pattern

Gregor Hohpe und Bobby Woolf geben in ihrem Werk *Enterprise Integration Patterns* (EIP) (Hohpe and Woolf [2012]) einen Einblick in die Thematik der Enterprise Integration. Dabei werden nicht nur typische Probleme sondern auch Lösungsansätze beschrieben. Für die Beschreibung werden Muster (*Pattern*) verwendet. Die Verwendung von Mustern, hat sich als ein geeigneter Weg gezeigt, eine unbegrenzte Anzahl von Varianten von Problemen/Lösungen in einem begrenzten Problem-/Lösungsbereich zu beschreiben. Muster wurden auch in bekannten Arbeiten wie Gamma et al. [2004] erfolgreich vorgestellt. In Hohpe and Woolf [2012] repräsentiert jedes Muster eine Entscheidung, die es zu treffen gilt und die Überlegungen die zu dieser Entscheidung führen. Die Muster sind miteinander verbunden und bilden ein Netz, welches von Hohpe und Woolf als *Pattern Language* bezeichnet wird. Durch die Auswahl von Mustern wird der Entscheidungsraum immer weiter eingeschränkt, so helfen die Muster bei der Lösungsfindung. Grundsätzlich konzentrieren sich die Arbeit Hohpe and Woolf [2012] auf Messaging zum Lösen von Integrationsproblemen.

Ein Grund für die Entwicklung der Enterprise Integration Patterns war für Hohpe und Woolf die Erstellung eines einheitlichen Vokabulars zur Beschreibung von Integrationsproblemen. So bieten sie eine Möglichkeit Integrationsprobleme allgemein zu beschreiben und fördern die Zusammenarbeit zwischen allen Beteiligten bei der Erstellung einer Integrationslösung.

Jedes Muster in Hohpe and Woolf [2012] wird durch einen Namen, einem Problem, einer visuellen Illustration des Problems und einer Lösung beschrieben. Die Namen sind meist so gewählt, dass sie einfach in Sätzen verwendet werden können. Das Problem und die Lösung werden jeweils durch einen Satz formuliert.

Die Enterprise Integration Pattern Icons Bibliothek ist unter anderem in Form von Open Office Stencils verfügbar. Ein speziellen Dank gilt Marco Garbelini für die Erstellung der Open Office Stencils und für die Erlaubnis zur freien Verwendung. Die Stencils für Open Office und für weitere Programme sind im Apache Camel Wiki[2] zu finden.

[2] https://cwiki.apache.org/confluence/display/CAMEL/Enterprise+Integration+Patterns

2.3.1. Grundlegende Elemente einer Integrationslösung

Um ein bessers Verständnis für die Aufgaben einer Integrationslösung und der auftretenden Herausforderungen zu schaffen, werden in Hohpe and Woolf [2012] die grundlegenden Elemente einer Integrationslösung beschrieben. Da der Einsatz von Enterprise Integration Pattern ist ein wichtiger Bestandteil dieser Arbeit ist und das Verständnis dieser für die Arbeit essenziell ist, werden die grundlegenden Elemente in diesem Kapitel beschrieben.

Damit Anwendungen Daten untereinander austauschen können, müssen einige Rahmenbedingungen erfüllt sein. Eine Grundanforderung ist Daten von unterschiedlicher Art auszutauschen. Das können einfache Texte wie XML oder HTML Dokumente sein, einfache Zahlen, oder auch entfernte Aufrufe (remote calls). Unabhängig davon, welche Daten ausgetauscht werden sollen, müssen diese von einer Anwendung zur Nächsten übertragen werden. Dazu ist ein Transportkanal (*Channel*) nötig, der Informationen übertragen kann. Das können TCP/IP Verbindungen sein, eine gemeinsame Datei, eine gemeinsame Datenbank, aber auch eine Warteschlange in einem Messaging System.

Eine der einfachsten Arten der Integration kann erreicht werden, in dem eine Anwendung Informationen in einem Kanal platziert. Die zweite Anwendung entnimmt die Nachricht aus diesem Kanal.

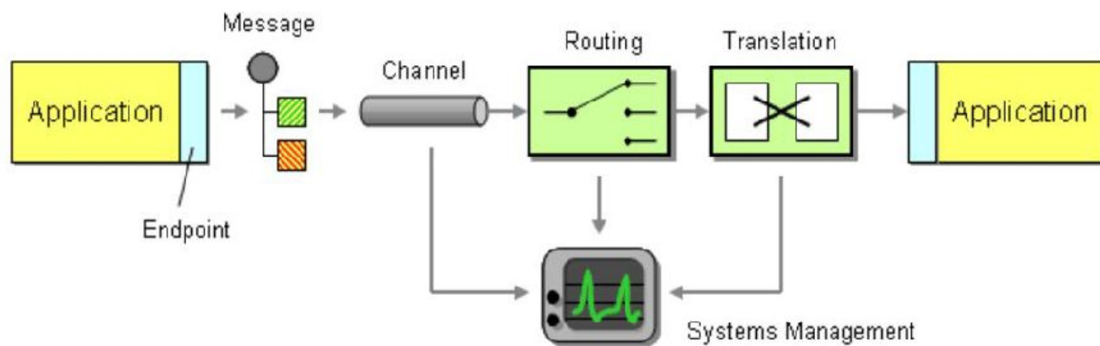

Abbildung 3: Grundlegende Elemente einer Integrationslösung Hohpe and Woolf [2012]

Integration ist jedoch meistens mit weiteren Herausforderungen verbunden. Abbildung 3 zeigt die grundlegenden Element einer Integrationslösung, die in den meisten Integrationslösungen benötigt werden. Anwendungen verwenden oft unterschiedliche Datenformate. Um Nachrichten einer Anwendung für Andere verständlich zu machen, muss die Nachricht in ein für die Zielanwendung verständliches Format übersetzt werden, dieser Schritt wird als Übersetzung (*Translation*) bezeichnet.

Sind mehr als zwei Anwendungen an einen Datenaustausch beteiligt muss entschieden werden, welche Nachricht an welchen Empfänger weitergeleitet werden soll. Natürlich könnte für jedes Anwendungspaar ein Transportkanal erstellt werden, jedoch wird das

mit der Anzahl der zu verknüpfenden Anwendung sehr aufwendig und unübersichtlich. Einfacher ist es die Integrationslösung (Middleware) diese Aufgabe übernehmen zu lassen. Mit Hilfe eines *Routers* (*Message Broker*) können Nachrichten an den jeweiligen Empfänger verteilt werden.

Integrationslösungen können mitunter sehr komplex werden. Um eine Übersicht über die Vorgänge zu behalten wird eine Form der Überwachung (*System Management*) benötigt.

Bis jetzt wurde angenommen, dass Anwendungen die Möglichkeit bieten, Informationen direkt an einen Kanal zu übergeben. Leider sind viele Anwendungen nicht dafür ausgelegt, an einer Integrationslösung teilzunehmen. Um diese Anwendungen trotzdem koppeln zu können, wird ein Nachrichten Endpunkt (*Message Endpoint*) benötigt, der die Anwendung mit der Integrationslösung verbindet.

Anschließend werden einige der wichtigsten Enterprise Integration Patterns aus Hohpe and Woolf [2012] beschrieben.

2.3.1.1. Message Channel *Wie können Anwendungen miteinander kommunizieren, wenn sie Messaging verwenden?*

Anwendungen können über einen Message Channel kommunizieren, indem eine Anwendung Daten in den Channel schreibt und die andere Anwendung die Daten aus dem Channel liest.

Abbildung 4: Message Channel

2.3.1.2. Message *Wie können zwei, über einen Message Channel verbundene Anwendungen, ein Stück Information austauschen?*

Die Information wird als Message verpackt, eine Message ist ein Datensatz der von dem Messaging System durch einen Message Channel übermittelt werden kann.

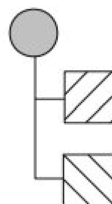

Abbildung 5: Message

2.3.1.3. Pipes and Filters *Wie können komplexe Bearbeitungsschritte an einer Message durchgeführt werden und gleichzeitig Unabhängigkeit und Flexibilität erhalten werden?*
Mit dem Pipes-and-Filter Muster, werden größere Bearbeitungsschritte, in eine Sequenz von kleineren, unabhängigen Schritten (Filters) unterteilt. Miteinander verbunden sind die einzelnen Schritte durch Kanäle (Pipes).

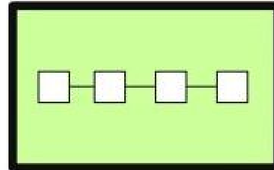

Abbildung 6: Pipes and Filters

2.3.1.4. Message Router *Wie können Messages, basierend auf einer Menge von Bedingungen, an unterschiedliche Filter weitergeleitet werden?*
Eine spezielle Art von Filter empfängt eine Nachricht und gibt sie basierend auf einer Menge von Bedingungen, an einen anderen Message Channel weiter.

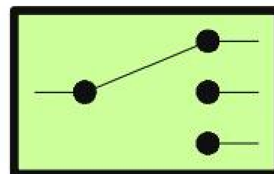

Abbildung 7: Message Router

2.3.1.5. Message Translator *Wie können Systeme mit unterschiedlichen Datenformate miteinander über Messaging kommunizieren?*
Eine spezielle Art von Filter übersetzt ein Datenformat in ein Anderes.

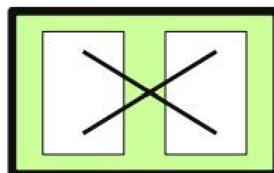

Abbildung 8: Message Translator

2.3.1.6. Message Endpoint *Wie kann eine Anwendung mit einem Messaging Channel verbunden werden, um Nachrichten zu senden oder zu empfangen?*

Ein Messaging Endpoint ist ein Client des Messaging Systems, der von Anwendungen verwendet wird, um Nachrichten zu senden, oder zu empfangen.

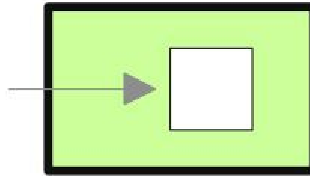

Abbildung 9: Message Endpoint

2.3.2. Entscheidungshilfe bei der Routerwahl

Hohpe und Woolf beschreiben 10 verschiedene Muster im Kapitel Message Routing. Die Abbildung 10 hilft, den Entscheidungsprozess bei der Auswahl eines Routers, zu vereinfachen.

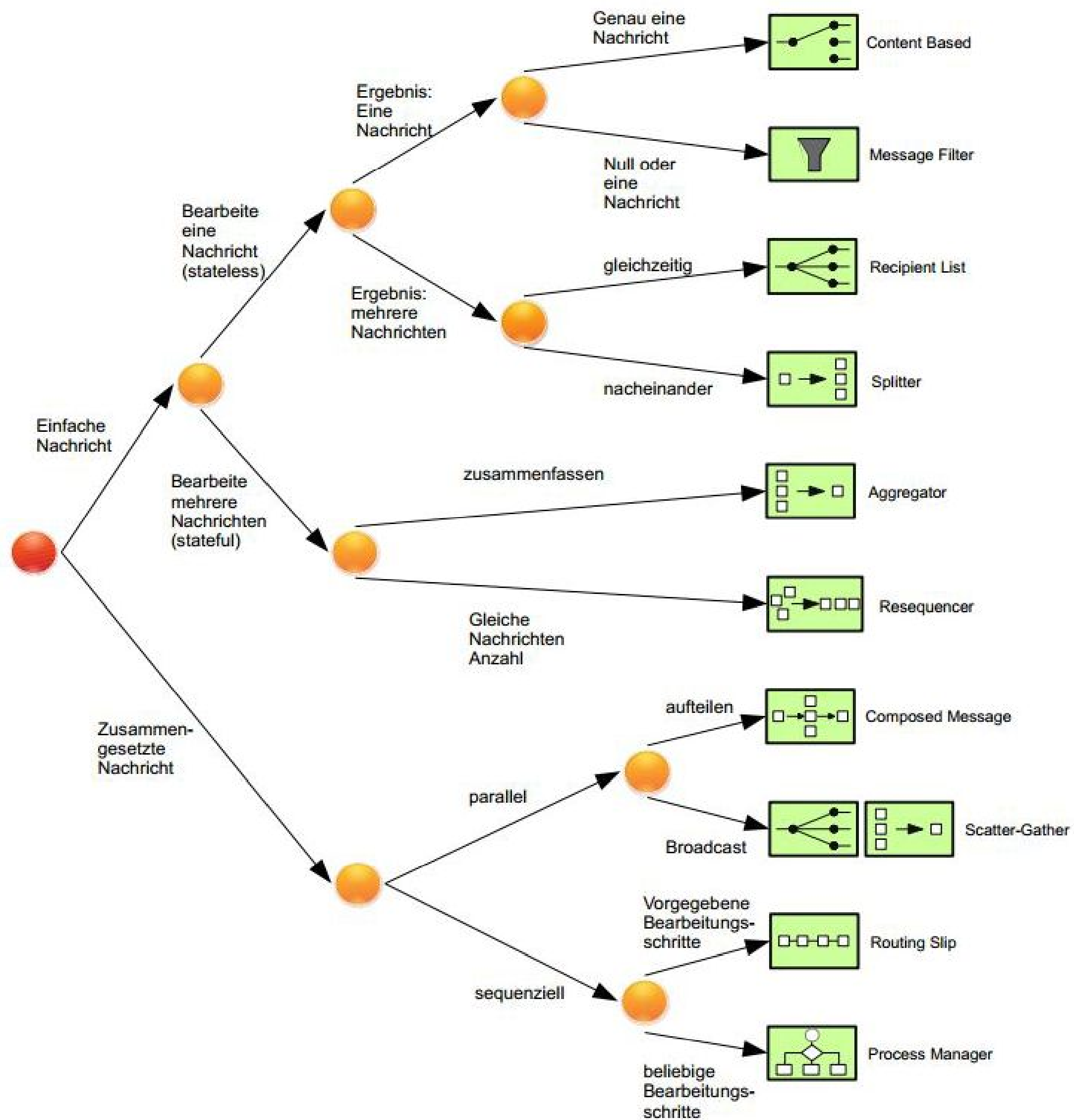

Abbildung 10: Übersicht Routerpattern Hohpe and Woolf [2012]

2.4. Integrationsarchitektur

Für die Lösung von Integrationsaufgaben, werden in Binildas [2008] vier grundlegende Architekturen für EAI identifiziert.

- *Point-to-Point Solution.* Diese Lösung ist die einfachste Arte der Integration. Es wird für jedes Paar von Anwendungen eine Direktverbindung erstellt. Sie ist noch oft in Unternehmen zu finden, meist entsteht sie, wenn Inseln von Anwendungen über die Zeit wachsen.

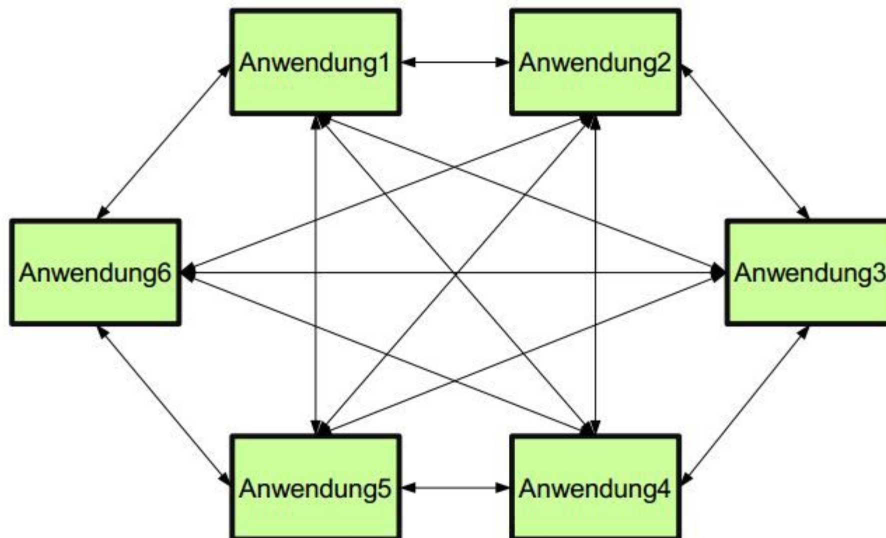

Abbildung 11: Point to Point

- *Hub-and-Spoke Solution*. Es wird ein zentraler Broker (Hub) verwendet, zu diesem können sich Anwendungen über Adapter (Spoke) verbinden. Der Adapter ist neben der Bereitstellung einer Verbindung zum Hub, auch für die Konvertierung der Daten in ein für den Hub verständliches Format zuständig. Der Hub ist für das Routing und die Transformation zuständig. Durch den Einsatz von *Hub-and-Spoke* können, im Vergleich zur *Point-to-Point*, die nötigen Verbindungen von $\frac{n(n-1)}{2}$ auf n reduziert werden. Eine zentrale Middleware, welche alle Intergationsaufgaben übernimmt erleichtert die Verwaltung, jedoch wird zugleich die Skalierbarkeit der Lösung erschwert.

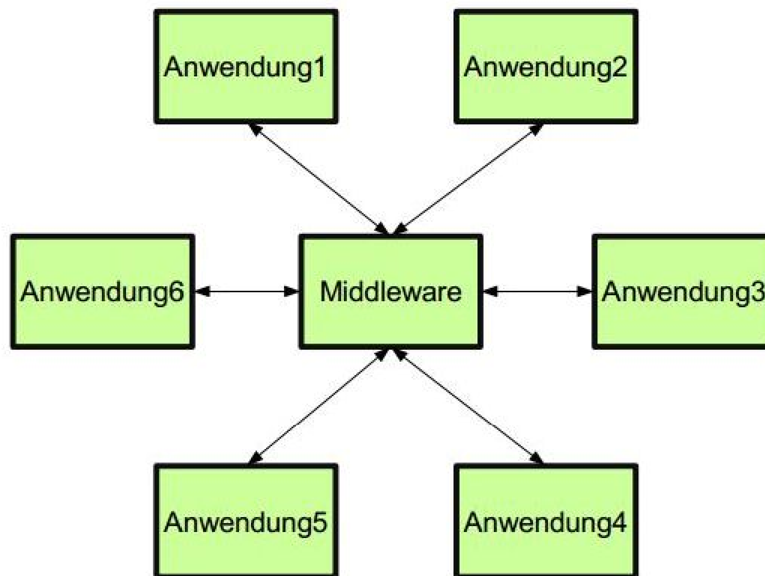

Abbildung 12: Hub and Spoke

- *Enterprise Message Bus Integration Solution.* Diese Lösung ist ähnlich der *Hub-and-Spoke* Lösung. Es gibt einen zentralen Message Bus für die Nachrichtenübermittlung. Einzelne Anwendungen können mit einem Adapter Nachrichten an den Message Bus übergeben, der Bus übernimmt die Weiterleitung der Nachricht. Adapter können Nachrichten vom Message Bus entnehmen und übernehmen die Umwandlung der Nachricht in ein für die Anwendung verständliches Format. Der wesentliche Unterschied zu *Hub-and-Spoke* ist, dass die Integrationsarbeit, wie die Umwandung der Nachrichten, oder das Routing von dem jeweiligen Adapter übernommen werden. Durch die Verteilung der Aufgaben an Adapter können die Lösungen besser skaliert werden, als in einer Hub-and-Spoke Architektur.

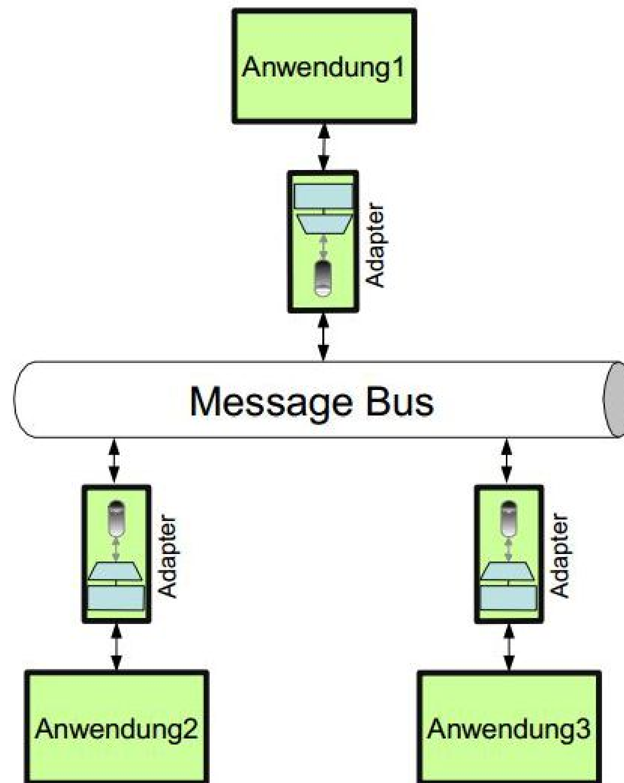

Abbildung 13: Message Bus

- *Enterprise Service Bus Intergration.* Technisch gesehen ist ein Enterprise Service Bus (ESB) eine Messaging Middleware und beinhaltet Aufgaben wie Protokoll Umwandlungen, Nachrichten Transformation, Routing, oder die Übernahme/Übergabe von Nachrichten an Services oder verknüpften Anwendungen. Obwohl ein ESB für die Realisierung von *Service Oriented Architecture* (SOA) verwendet werden kann, ist durch den Einsatz eines ESB nicht automatisch eine SOA realisiert (siehe Abschnitt 2.5). Bei einem ESB und einer SOA spielt der Einsatz von Services eine zentrale Rolle.

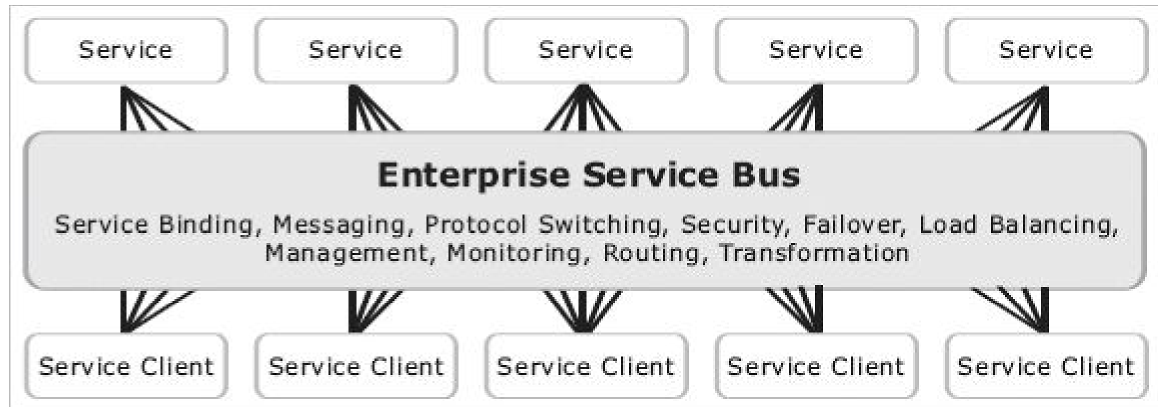

Abbildung 14: Enterprise Service Bus nach Oracle

Ein ESB und ein Message Bus haben einige Gemeinsamkeiten, haben aber im Kern unterschiedliche Ansätze. Beide Ansätze versuchen eine einheitliche Kommunikationsinfrastruktur zu bieten und die Mediation in einer lose gekoppelten Art ist eine zentrale Aufgabe beider Architekturen. In Binildas [2008] wird als wesentlicher Unterschied der beiden Architekuren, die Art wie Konsumenten mit der Messaging Middleware interagieren, angeführt. Ein Message Bus beschreibt keine Services, es können Nachrichten an einen Message Bus übergeben werden oder von dem Bus empfangen werden, es gibt jedoch keine Information über die Art des Nachrichtenaustauschs (siehe Abschnitt 2.2.3), dem Schema der ausgetauschten Nachrichten oder Informationen über den darunterliegenden Service. Im Gegensatz dazu stehen bei einem ESB Services im Mittelpunkt. In einer idealen ESB Architektur suchen Konsumenten beim ESB nach einem Service der ihren abstrakten Erwartungen entspricht. Der ESB routet die Anfrage an eine konkrete Serviceimplementierung weiter, welche anschließend die Anfragen bearbeitet.

Im Groben stimmen die Werke von Binildas [2008] und Schmutz et al. [2010], in Bezug auf die Integrationsarchitekuren überein. In Schmutz et al. [2010] wird der Begriff *Pipeline Architecture* statt *Enterprise Message Bus* und *Service-oriented Architecture* statt *Enterprise Service Bus* verwendet. Jedoch verwendet Binildas [2008] den Begriff SOA weitreichender und vergleicht ihn nicht mit einem Enterprise Service Bus.

2.5. Service-Oriented Architecture

Keen et al. [2004] beschreibt *Service-Oriented Architecture* (SOA) als einen Ansatz zur Erstellung von Architekturen aus integrierten Anwendungen, basierend auf dem Konzept eines *Service*. Eine Service-Oriented Architecture definiert keine zugrundeliegende Technologie. Es ist an dem Architekten der jeweiligen Lösung, zu entscheiden, mit welchen Technologien die Lösungen/Services realisiert werden. Heutzutage wird SOA jedoch oft mit Hilfe von Web Services realisiert. Nach Maamar et al. [2014] erlaubt ein Web Service einem Anbieter (*Provider*), klar abgegrenzte Funktionen über ein Netzwerk zur Verfü-

gung zu stellen. Konsumenten (*Consumer*) können einen Web Service aufrufen und somit dessen Funktionen nutzen. Im Abschnitt 4.4 werden Web Services genauer beschrieben. Unter *Service Orientierung* wird in Credle et al. [2007], eine Methode zur Integration von Geschäftsfunktionen und Prozessen durch die Verknüpfung von Services verstanden.

In Rotem-Gal-Oz [2012] sind einige der Basiskomponenten und Zusammenhänge in der Verwendung von Services in einer SOA dargestellt. Dazu gehören:

- Services - Ein *Service* wird in Keen et al. [2006] als Funktion definiert, die einem externen Konsumenten angeboten wird. Die Funktion kann eine einfache Geschäftsfunktion, aber auch ein gesamter Geschäftsprozess sein.
- Contract - Rotem-Gal-Oz [2012] beschreibt einen Servicevertrag als Analogie zum Interface eines Objekts in der Objekt Orientierten Programmierung. Ein Servicevertrag beschreibt eine Menge von *Nachrichten* (Messages), die von dem Service bearbeitet werden können. Mit Hilfe des Vertrags werden Services definiert.
- Messages - Nachrichten sind ein grundlegender Bestandteil von SOA und werden für den Informationsaustausch genutzt.
- Endpoint - Um Services nutzen zu können, müssen diese adressiert werden können. Dazu werden *Endpunkte* verwendet. Ein *Endpunkt* (Endpoint) ist eine Adresse oder eine spezifischer Platz, wo ein Service gefunden werden kann. Oft werden Endpunkte mit Universal Resource Identifier (URI) angegeben.
- Policy - So wie ein Vertrag oder ein Interface die Spezifizierung von der Implementierung trennt, trennen *Policies* dynamisch Spezifikationen von Statischen. Das besondere an *Policies* ist, dass diese zur Laufzeit aktualisiert und geändert werden können.
- Service Consumer - Services haben erst eine Bedeutung, wenn diese auch genutzt werden. Ein Servicekonsument (Service Consumer) ist die Komponente, die über Nachrichten mit Services interagiert. Konsumenten können wiederum selbst Services sein.

In der Literatur gibt es viele unterschiedliche Meinungen was SOA ist und wie eine SOA oder ein Service aussehen soll. In Keen et al. [2004] werden einige Aspekte beschrieben, die laut allgemeiner Meinung ein Service erfüllen sollte. Dazu gehört, dass ein Service durch ein von der Implementierung unabhängiges Interfaces definiert ist. Durch die Verwendung eines Interfaces wird die spezifische Implementierung des Services von der Interaktion zwischen zwei Systemen entkoppelt. So kann die Implementierung geändert werden ohne aufrufende Systeme zu beeinflussen. Außerdem sollten Services wiederverwendbare, klar abgegrenzte Geschäftsfunktionen sein.

In Heutschi [2007] werden vier Klassen von Designprinzipien für SOA identifiziert:

- *Schnittstellenorientierung.* Serviceschnittstellen nehmen eine funktionale Abstraktion vor Sie definieren was ein Service leistet und wie er genutzt wird, aber nicht

wie der Service implementiert ist. Durch diese Trennung wird die Anpassbarkeit der Serviceimplementierung erhöht. Um die Wiederverwendbarkeit zu erhöhen, müssen Services umfassend definiert und beschrieben sein.

- *Interoperabilität.* In einer SOA muss definiert werden wie Softwareelemente Informationen austauschen und interpretieren können. Es ist empfehlenswert offene und verbreitete Industriestandards zu verwenden.
- *Autonomie und Modularität*: Es ist eine hohen Kohäsion und eine schwache logische Kopplung der Services anzustreben. Asynchrone, nachrichtenbasierte Kommunikation und statuslose Serviceinteraktionen führen zu einer lose gekoppelten Kommunikation und reduzieren Abhängigkeiten zur Laufzeit der Services.
- *Bedarfsorientierung.* Eine grobe, an geschäftlichen Konzepten orientierte Servicegranularität, hilft eine bedarfsgerechte Menge von Services zu definieren.

In der Literatur gibt es eine breite Palette an Werken, welche sich mit Designprinzipien von SOA und Services beschäftigen. In Vasiliev [2007] ist eine Liste von Basisprinzipien aufgeführt, welche einen guten Einstieg in die Thematik bieten.

Obwohl EAI und SOA viele Gemeinsamkeiten haben, gibt es doch grundlegende Unterschiede. Es gibt unterschiedlichste Technologien und Methoden, wie ESB oder Message Bus Systeme, mit deren Hilfe eine SOA designed und realisiert werden kann. Aber nicht jede Integrationslösung die Services, oder Systeme wie ESB verwendet, ist automatisch eine SOA. Richtig designed und umgesetzt ermöglicht SOA den Unternehmen, ihre IT an die sich schnell ändernden Geschäftsprozesse anzupassen.

Für weitere Informationen wie SOA realisiert werden kann, wird auf Rotem-Gal-Oz [2012] verweisen, die dort beschrieben Pattern helfen bei der Erstellung von SOA. In Heutschi [2007] werden einige Fallbeispiele zur Umsetzung von SOA gezeigt.

2.6. Bestehende Integrationslösungen

Aktuell existiert am Markt eine breite Palette an Lösungen, die bei der Integrationen von Anwendungen unterstützen können. Aufgrund der Vielzahl an Produkten und unterschiedlichen Bezeichnungen ist es schwierig, die einzelnen Lösungen voneinander abzugrenzen. Es gibt auch keine klaren Richtlinien, in welchen Fällen welche Integrationslösung zu bevorzugen ist.

In Hohpe and Woolf [2012] wird eine grobe Einteilung von Messaging Systemen über Betriebssystem, Applikationsserver, EAI Suites und WebService Toolkits vergenommen.

Ohne Anspruch auf Vollständigkeit zeigt die folgende Liste einige der berühmten EAI Lösungen bzw. Frameworks.

- Apache Camel
- Mule ESB
- Apache Service Mix

- BizTalk Server
- Talend ESB
- Fuse ESB
- JBOSS ESB
- Spring Integration Framework
- SAP NetWeaver
- Talend ESB
- WebSphere ESB

Alle diese Lösungen haben viele Gemeinsamkeiten, alle implementieren die EIPs und bieten ein durchgängiges Datenmodell sowie ein Messaging System um unterschiedliche Technologien integrieren zu können. Alle Frameworks basieren auf den gleichen grundlegenden Ideen, verwenden jedoch andere Bezeichnungen. So entspricht eine Camel Route, einem Mule Flow, oder eine Camel Component entspricht einem Adapter in Spring Integration. Teilweise sind die oben genannten Lösungen als OpenSource Projekt verfügbar. Jedoch ist die Lizenz bei Talend ESB oder Mule ESB wesentlich restriktiver als bei Apache Camel oder Spring. Ist es notwendig sich in die Entwicklung der Projekte einzubringen, geht das bei Projekten unter der Apache Lizenz, wesentlich einfacher als bei Talend oder Mule.

Ein wesentlicher Unterschied zwischen den einzelnen Lösungen ist der Umfang der Integrationslösung, so enthalten die vollwertigen ESB meist umfangreiche IDEs mit denen eine Konfiguration ohne Code möglich ist. Vollwertige ESB bieten oft eine komplette Integrationsplattform, welche sowohl Service Container als auch ein Mediation Framework enthalten. Das ermöglicht ESBs mit Anforderungen bezüglich Verfügbarkeit, Skalierbarkeit und Sicherheit umzugehen. Werden diese Funktionen nicht benötigt können viele der ESBs, auch als leichtgewichtiges Integrations Framework verwendet werden. So verwenden sowohl Fuse ESB, als auch Apache Service Mix, Apache Camel als grundlegende Mediation Engine.

Die einzelnen Lösungen unterscheiden sich im Weiteren durch die Verfügbarkeit von Komponenten. Wird im Integrationsprojekt eine bestimmte Komponente benötigt, sollte ein Framework verwendet werden, dass diese Komponente bereits enthält. Bei fast allen Integrationslösungen können zusätzlich Komponenten entwickelt werden. Aktuell bietet Apache Camel die breiteste Palette an Komponenten, jedoch bieten Mule und Talend exklusive Komponeten die in anderen Lösungen nicht vorhanden sind.

Der Umfang des Integrationsprojekts ist von entscheidender Bedeutung bei der Auswahl des Frameworks. Sind nur zwei Applikationen zu integrieren, wird man zu einer einfachen Mediation Engine wie Apache Camel greifen. Sollen viele Applikationen miteinander integriert werden, ist der Einsatz eines vollwertigen ESB zu empfehlen. Existieren Anforderungen bezüglich Enterprise Support, wie zum Beispiel 24/7 Hotline, dann muss

meist zu proprietären Produkten gegriffen werden.

Pauschal kann keine Empfehlung gegeben werden, welche Integrationslösung, bei welchen Problemstellungen zu bevorzugen ist. In der Masterarbeit Thullner [2008], werden einige der wichtigsten Frameworks miteinander verglichen und gezeigt welche Muster von den Frameworks unterstützt werden. Diese Arbeit unterstützt bei der Auswahl eines geeigneten Frameworks.

3. Teamcenter

TEAMCENTER ist die PLM Suite von Siemens Product Lifecycle Management Software Inc., kurz Siemens PLM Software. „*Smarter Decisions, Better Products*“ ist ein Slogan von TEAMCENTER und repräsentiert die zentrale Idee, die Siemens PLM Software mit TEAMCENTER verfolgt. Dank einer zentralen Quelle für Produkt- und Prozesswissen können *intelligentere Entscheidungen* schneller und mit höherer Sicherheit getroffen werden. Eine der zentralen Aufgaben von TEAMCENTER ist laut Siemens PLM Software [2013], eine intelligente Integration von Informationen über den gesamten Produktlebenszyklus zu erzielen. So kann zu *besseren Produkten* beigetragen werden.

Die ersten Softwareversionen aus denen sich TEAMCENTER entwickelt hat, stammen aus den 80er Jahren, den Anfängen des Computer Aided Designs (CAD). Über die Zeit entwickelten sich Produkte von CONTROL DATA, SDRC und UGS wie METAPHASE und IMAN zu den ursprünglichen Teamcenteranwendungen TEAMCENTER ENTERPRISE und TEAMCENTER ENGINEERING. Gleichzeitig wurden neue Anwendungen für die jeweilige PLM Suite entwickelt und akquiriert, dass führte zu vielen verschiedenen Anwendungen mit unterschiedlichen Architekturen. Anfang 2000 begann UGS PLM SOLUTIONS (jetzt Siemens PLM Software) diese unterschiedlichen Produkte in eine gemeinsame Plattform zusammenzuführen, bekannt unter dem Begriff *Unified Architecture*. Die Architektur basiert auf den neuersten Technologien und aktuellen Standards. Im März 2009 erschien TEAMCENTER 8, das Ergebnis von vielen Jahren Entwicklung und basierend auf der Unified Architectur. Im Sommer 2014 wurde die bisher letzte Version TEAMCENTER 10.1 veröffentlicht. Aktuell ist TEAMCENTER, mit über 7 Millionen lizenzierten Anwendern und mehr als 71000 Kunden, das weltweit meist genutzte PLM System.

Arnold et al. [2011] beschreibt das *Product Lifecycle Management* (PLM) als ein integrierendes Konzept zur IT-gestützten Organisation aller Informationen über Produkte und deren Entstehungsprozesse über den gesamten Produktlebenszyklus hinweg, so dass die Information immer aktuell an den relevanten Stellen im Unternehmen zur Verfügung stehen.

3.1. Teamcenter Architektur

Die *Unified Architecture* besteht aus vier funktional getrennten Ebenen, in Folge als Tiers bezeichnet. Es gibt mehrere Gründe die Funktionen von TEAMCENTER in abgegrenzte Ebenen aufzuteilen. Ein Grund ist, den Betrieb in einer existierenden IT-Umgebung zu ermöglichen. Ein weiterer Grund ist die Aufspaltung von unterschiedlichen Hardwareanforderungen, um eine möglichst hohe Skalierbarkeit der Anwendungen zu erreichen.

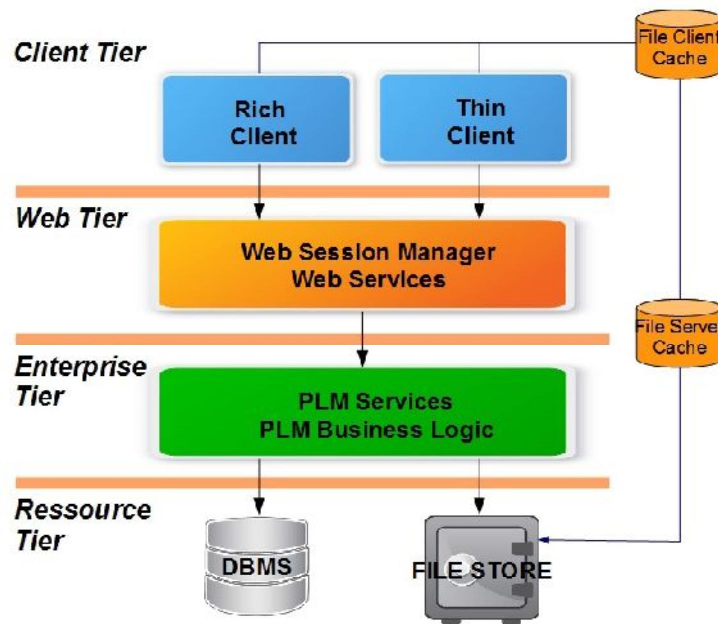

Abbildung 15: 4-Tier Architektur

Abbildung 15 zeigt eine schematische Darstellung der 4-Tier Architektur von TEAMCENTER. Sie besteht aus folgenden Schichten:

- Die *Client-Tier* beherbergt alle Endbenutzer Anwendungen, stellt Benutzerinterfaces und clientseitige File Caches zur Verfügung. Standardmäßig sind sowohl ein Rich-Client, als auch ein webbasierender Thin-Client für TEAMCENTER verfügbar. Mit Hilfe von bereitgestellten SOA Clientbibliotheken können neue Clients implementiert werden.
- Die *Web-Tier* routet Clientanfragen an die Geschäftslogik (Business-Logic) weiter und bietet statische Inhalte an. Die Web-Tier kann mit .NET auf dem Microsoft IIS oder mit J2EE auf einem J2EE-Applikationsserver, wie JBOSS oder Oracle WEBLOGIC betrieben werden.
- Die *Enterprise-Tier* ist für die Bereitstellung von dynamischen Inhalten zuständig und beherbergt die Geschäftslogik. Sie besteht aus einem Pool von Teamcenterserverprozessen und einem Servermanager. Die Serverprozesse sind für das Abfragen und Schreiben von Informationen aus und in die Datenbank zuständig. Der Servermanager ist für die Verwaltung der Serverprozesse zuständig.
- Die *Resoure-Tier* ist für das dauerhafte Speichern von Metadaten und Dateien zuständig. Verwendet werden Datenbankserver, Datenbank, Volumes und Fileserver.

Die TEAMCENTER Clientanwendungen ermöglichen Benutzern den Zugriff auf die von TEAMCENTER verwalteten Daten, Funktionen und Prozesse. TEAMCENTER bietet eine Reihe von unterschiedlichen Clients, zu den wichtigsten gehören der Rich Client und

der browserbasierte Thin Client. Der Rich Client stellt einen großen Funktionsumfang bereit und ist das bevorzugte Werkzeug für alle Benutzer die regelmäßig mit TEAMCENTER arbeiten. Der Rich Client ist eine Javaanwendung und so auf unterschiedlichen Betriebssystemen einsetzbar. Der Thin Client benötigt keine Installation und kann mit den meisten Browsern genutzt werden. Er bietet jedoch im Vergleich zu dem Rich Client einen abgespeckten Funktionsumfang. Fordert ein Client Informationen aus TEAMCENTER an, wird eine Anfrage an die Web-Tier gesendet, diese nimmt die Anfrage entgegen und gibt sie an die Enterprise-Tier weiter, wo sie verarbeitet wird. Die Enterprise-Tier holt die benötigten Informationen aus der Ressourcen-Tier und erstellt eine Antwort für den Client.

Anfragen für Dateien werden über ein eigenes Datei Verwaltungssystem, dem *File Management System* (FMS), direkt an den Datenspeicher gesendet. So wird ein schneller und sicherer Dateitransfer sichergestellt. Das FMS bietet Funktionen für das Speichern, Verteilen, Cachen und eine Zugriffsteuerung auf Dateien. Zum FMS gehören der *Volume Server* für die Datenablage, ein *File Server Cache* (FSC) und ein *File Client Cache* (FCC). Die Caching Funktionalitäten ermöglichen es, Daten näher am Verbraucher zu halten und somit die Verfügbarkeit zu verbessen. Gleichzeitig werden alle Dateien weiterhin zentral in *Volumes* und in der Datenbank verwaltet. Das FMS kann auf unterschiedlichste Weise betrieben werden. Eine einfache Betriebsart ist in Abbildung 16 abgebildet. Zur Perfomancesteigerung können die Komponenten beliebig erweitert werden.

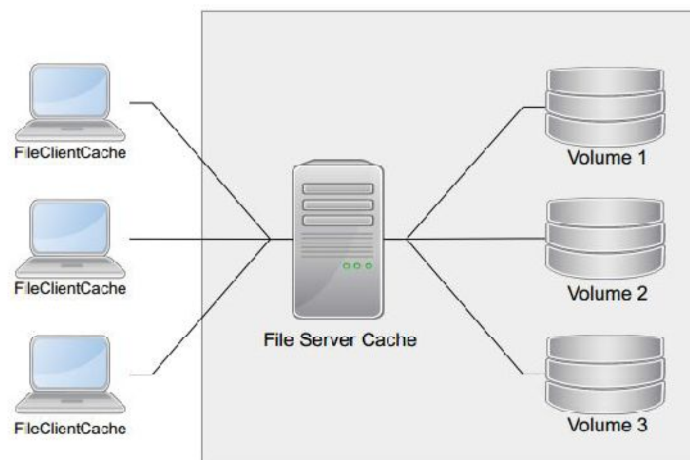

Abbildung 16: File Management System

Alle Schichten/Tiers können auf einem gemeinsamen Host (Gastsystem) betrieben werden. Um eine Verteilung der Last auf mehrere Hosts und somit eine Skalierung der Anwendung zu ermöglichen, können die Schichten auf verschiedene Hosts verteilt werden. Zum Zweck der Lastverteilung ist der Betrieb von mehreren Applikation Servern und Servermanagern möglich (siehe Abbildung 17).

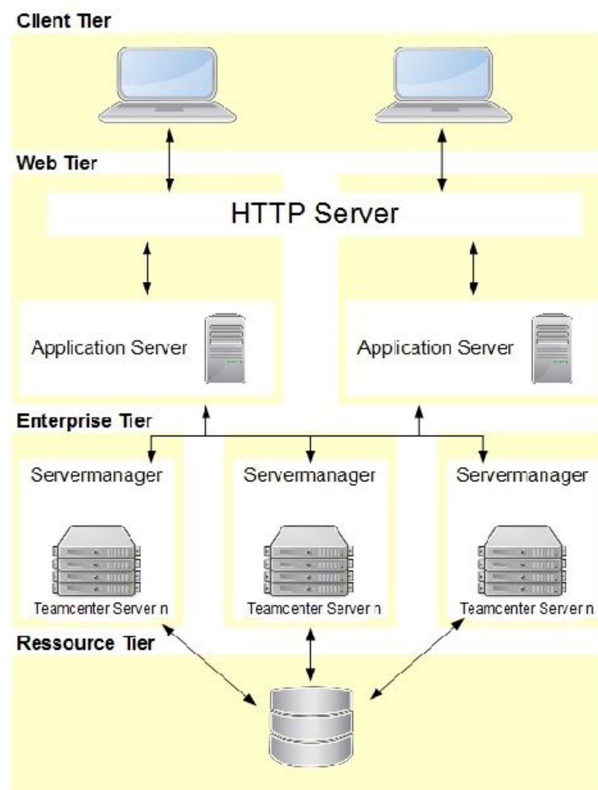

Abbildung 17: Mögliches Deployment für Lastverteilung

Zusätzlich zu der *4-Tier Architektur* bietet TEAMCENTER die Möglichkeit eines 2-Tier Betriebs. Bei der *2-Tier Architektur* werden die einzelnen Schichten zusammengezogen. Die Clientschicht beherbergt in diesem Fall nicht nur die Enduseranwendung, sondern auch die Geschäftslogik. Für das dauerhafte Speichern von Metadaten und Dateien ist weiterhin die Ressourcen Schicht zuständig. Für die Kommunikation wird CORBA[3] verwendet.

3.2. Teamcenter Datenmodell

TEAMCENTER hat ein eigenes Datenmodell, das die Repräsentation von Geschäftsobjekten und Geschäftsprozessen ermöglicht. Das Datenmodell von TEAMCENTER kann mit dem *Business Modeler IDE* (BMIDE) erweitert und angepasst werden. Das TEAMCENTER Datenmodell besteht aus mehreren Komponenten die im Folgenden vorgestellt werden.

- *Klassen* - Sind die persistente Repräsentation von Objekten, in Form von Tabellen in einer Datenbank. Jeder Zeile in einer Tabelle entspricht einer Instanz einer Klasse

[3] http://www.corba.org

- *Attribute* - Beschreiben die Eigenschaften einer Klasse. Jede Spalte in einer Tabelle entspricht einem Attribut des Klasse. Attribute werden entweder von der Elternklasse geerbt oder sind direkt in der Klasse definiert.
- *Geschäftsobjekte* - Repräsentieren die Geschäftsdaten welche in TEAMCENTER verwaltet und verarbeitet werden. Geschäftsobjekte definieren welche Werte in der Datenbank gespeichert werden. Das Verhalten von Geschäftsobjekten kann u.a. mit Wertelisten, Benennungsregeln und ähnlichem konfiguriert werden. Es gibt persistent Geschäftsobjekte diese werden in der Datenbank gespeichert. Zusätzlich gibt es Geschäftsobjekte, welche zur Laufzeit der Anwendung berechnet werden.
- *Eigenschaften* - Repräsentieren die Werte von Geschäftsobjekten.

An der Spitze der Geschäftsobjekte steht das *POM_object*, *POM* steht für *Persistent Object Manager*. Der POM definiert die Architektur (Schema) für die Verwaltung der Teamcenterdaten in der Datenbank und in laufenden Teamcentersitzungen. Geschäftsobjekte repräsentieren die unterschiedlichen Arten von Objekte, wie Items, Datasets, Forms, Folders usw.

Ein Objekt ist eine Datenstruktur und ist im Grunde eine Instanz einer Klasse. Die Klasse definiert den Type eines Objekts, die Attribute und Methoden die auf das Objekt angewendet werden können. Ein Folder, ähnlich einem Ordner im Dateisystem, ist ein Beispiel für eine Klasse in TEAMCENTER. Er hat einen Namen, eine Beschreibung und eine Liste von Einträgen, als Attribut. Wenn ein Folder instanziiert wird, wird ein neues Folder-Objekt erzeugt und in der Datenbank gespeichert.

TEAMCENTER unterstützt unterschiedliche Typen von Attributen, dazu gehören, integer, float, boolean, string, date, tag und Arrays dieser Typen. Ein tag ist ein eindeutiger Identifier für ein verknüpftes Objekt. Attribute in Teamcenter können zusätzliche Charakteristika haben (Unique, Protected, NULL allowed, Upper bound, Lower Bound, Default Value)

Die Abbildung18 zeigt einen Auszug mit einigen der wichtigen Klassen in TEAMCENTER.

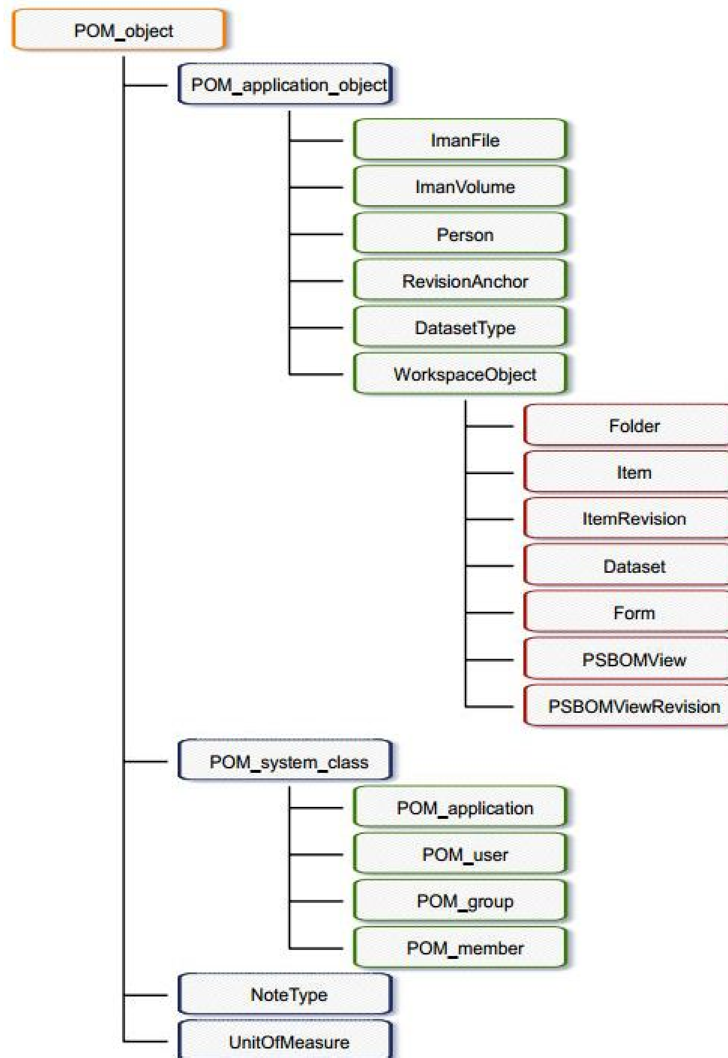

Abbildung 18: Auszug aus der Teamcenter Klassen Hierarchie

3.2.1. Grundlegende Geschäftsobjekte

In diesem Abschnitt, werden einige der gängigsten Geschäftsobjekte kurz beschrieben.

- *Item*: In TEAMCENTER sind Items die fundamentalen Objekte für die Verwaltung von Informationen. Typischerweise werden Artikel, Teile, Dokumente, Ausrüstung und ähnliches mit Items abgebildet.
- *ItemRevision*: Revisionen repräsentieren die unterschiedlichen Änderungsstände eines Items. Wann neue Revisionen erzeugt werden, ist von den Geschäftsprozessen des jeweiligen Unternehmens abhängig.
- *Relation*: Organisationen erzeugen oft Objekte mit Informationen, die nur in einem bestimmten Zusammenhang mit anderen Objekten sinnvoll sind. Um die Objekte verstehen zu können, müssen sie in einer aussagekräftigen Weise mit anderen

Objekten verknüpft werden. TEAMCENTER verwendet zu diesem Zweck *Relationen*. Bei der Erstellung von *Relationen* müssen je nach *Relation* gewisse Regeln eingehalten werden. Eine Liste der Regeln kann in Siemens PLM Software [2014b] nachgeschlagen werden. Zu den wichtisten Relationen gehören unter anderem die *IMAN_revision*, dieser verknüpft Revisionen mit dem zugehörigen Item. Oder die *IMAN_specification*, diese verknüpft beschreibende bzw. definierende Daten wie CAD Modelle mit Items oder Revisionen. Es gibt noch eine Menge weiterer *Relationen* die spezielle Verknüpfungen zwischen Objekten repräsentieren, für eine vollständige Liste wird auf die Dokumentation der jeweiligen Teamcenterinstallation verwiesen.

- *Dataset*: In Datasets werden meist Informationen gespeichert, die in anderen Anwendungen erstellt wurden. Typische Beispiele sind Dokumente oder CAD Modelle. Zu einem Dataset können mehrere *Versionen* existieren die unterschiedliche Änderungsstände repräsentieren. Betrachtet man das das *Dataset* im Teamcenter Datenmodell, besteht es aus einer Menge von *Dataset*-Objekten. Jedes dieser *Dataset*-Objekte referenziert ein oder mehrere *ImanFile*-Objekte. Die *ImanFile*-Objekte beinhalten die eigentliche Information. Zusätzlich existiert ein *RevisionAnchor*-Objekt, welches die Liste der Datasetversionen beinhaltet und die aktuellst Datasetversion kennt.

3.3. System Administration

Um Objekte vor unbefugtem Zugriff zu schützen, haben die Instanzen der *POM_ application_object* Klassen und deren Subklassen Eigentümer und spezielle Zugriffsrechte in Form einer *Protection*. Der Schutz (*Protection*) ergibt sich aus den Einträgen der *Access Control List* (ACL). Ein Eintrag besteht aus einer Referenz zu einem Benutzer und/oder Gruppe und den Zugriffsberechtigungen für bestimmte Objekte oder Objekttypen.

In Abbildung 19 sind die, für die Systemadministration relevanten, Abhängigkeiten zwischen den Model Objekten von Teamcenter dargestellt.

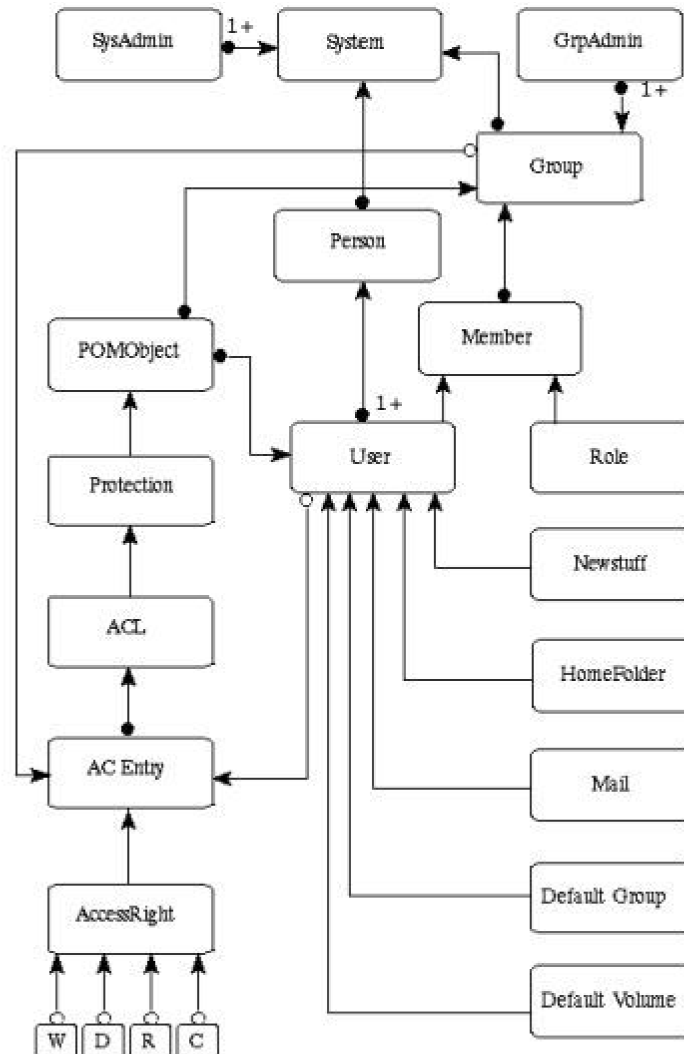

Abbildung 19: System administration model Siemens PLM Software [2014b]

3.4. Produktstrukturen

Produkte bestehen in der Regel aus mehr als einem Einzelteil. PLM Systeme müssen aus diesem Grund eine Möglichkeit bieten um Produktstrukturen zu verwalten. Teamcenter bietet für diese Aufgabe das Strukturmanagement, welches die Erstellung, Anzeige oder Bearbeitung von Produktstrukturen ermöglicht. Produkte die aus einer Menge von Teilen bestehen werden auch als Baugruppen bezeichnet. Im Gegensatz dazu sind Einzelteile Teile die keine Struktur besitzen. Baugruppen entstehen meist durch eine Gruppierung von Teilen, um diese komfortabel weiterverwenden zu können. Eine Baugruppe kann sowohl Einzelteile als auch andere Baugruppen beinhalten.

In Teamcenter werden sowohl Baugruppen als auch Einzelteile mit Hilfe von Items bzw. Item Revisionen verwaltet. Ist ein Item/ItemRevision eine Baugruppe, so beinhaltet sie ein Objekt vom Typ BOMView/BOMViewRevision. In der BOMView ist

gespeichert, aus welchen Teilen sich die Baugruppe zusammensetzt. TEAMCENTER bietet eine große Anzahl von Konfigurationsmöglichkeiten für Stücklisten, dazu gehören die *Revisionskonfiguration*, *Vorkommniskonfiguration* oder auch die *Variantenkonfiguration*. Alle Konfigurationsmöglichkeiten zu beschreiben würde den Umfang dieser Arbeit übersteigen. Da die *Revisionskonf*iguration ein essentieller Teil des Strukturmanagements in TEAMCENTER ist, wird diese kurz beschrieben.

In einer Stücklistenansicht ist gespeichert, aus welchen Teilen eine Baugruppe besteht. Ein Teil (Item) in TEAMCENTER kann jedoch aus mehreren Änderungsständen (Revisionen) bestehen. Öffnet man eine Baugruppe im Strukturmanagement, errechnet TEAMCENTER, auf Grund einer *Revisionsregel*, welche Revisionen als Teile der Baugruppe konfiguriert werden. Eine Revisionsregel besteht wiederum aus einer Menge von Regeln. Diese werden angewendet, um eine geeignete Revisionskonfiguration zu errechnen.

3.5. Automatisierung mit Workflows

Teamcenter bietet mit Workflows die Möglichkeit interaktive oder automatische Geschäftsprozesse zu erstellen. Ein Workflow besteht dabei aus einer Abfolge von Aufgaben. Im Zuge eines Workflows werden die Informationen oder Dokumente entsprechend den definierten Regeln an die jeweiligen Teilnehmer zur Durchführung der Aufgabe weitergegeben. In welcher Reihenfolge und durch welchen Teilnehmer die Aufgaben durchgeführt werdem müssen, kann über den Workflow definiert werden. Typische Aufgaben für Workflows sind die Durchführung von Prüfungen, die Statusvergabe oder ein Datenexport.

Der Einsatz von Workflows ermöglicht eine Standardisierung der Geschäftsprozesse und fördert die Effizienz indem unnötige Arbeitsschritte eliminiert werden können.

3.6. Lizenzierung

In TEAMCENTER wird das sogenannte *Named User Licensing* verwendet, dabei wird jeder Benutzer im System genau einer verfügbaren Lizenz zugeordnet. Die Anzahl der aktiven Lizenzen kann dabei niemals die Anzahl der verfügbaren Lizenzen übersteigen. Eine aktive Lizenz ist Voraussetzung für eine Anmeldung am System. Für die Verteilung der Lizenzen wird der Siemens PLM Software Common Licensing Server Daemon (*ugslmd*) verwendet.

3.7. Teamcenter Services

In Siemens PLM Software [2014a] werden die Teamcenterservices als Grundlage für hoch performante, erweiterbare, WAN freundliche Teamcenteranwendungen dargestellt. Mit den Teamcenterservices können, unabhängig von Programmiersprache und Betriebssystemen, lose gekoppelte Integrationen mit TEAMCENTER realisiert werden. Die Funk-

tionen der Business Logic Server sind per API als WebServices veröffentlicht. Es können sowohl interaktive Clientanwendungen für Endbenutzer, als auch nicht interaktive System-zu-System Integrationen realisiert werden. Vorhanden sind Clientbibliotheken für Java, C++ und .NET. Zusätzlich sind dem WS-1 Standard genügende Web Services Description Language - Dokumente (WSDL) verfügbar, wodurch auch andere als die genannten Sprachen verwendet werden können.

Die Services basieren auf der Idee der *Service-oriented Architecture*. Zu den wichtigsten Zielen, denen die Teamcenterservices genügen sollen, gehören nach Siemens PLM Software [2014a]:

- *Contract based interfaces*: Die Services-Interfaces sind von der darunterliegenden Geschäftslogik entkoppelt. Die Interfaces entsprechen einem Vertrag zwischen Client und Service. Solange der Client eine Anfrage an einen Service stellt, wird der Service diese Anfrage mit einer dem Vertrag entsprechenden Antwort beantworten. Der Client kennt dabei nicht die internen Abläufe der Services. So werden Client und Serverimplementierung entkoppelt und können sich unabhängig voneinander entwickeln.
- *Coarse-grained interfaces*: Alle Services repräsentieren eine vollständige Transaktion. Stellt eine Client eine Anfrage, führt der Service die Operation durch ohne eine weitere Interaktion mit dem Client zu benötigen. Am Ende der Operation sendet der Service eine vollständige Antwort an den Client.
- *Single entry point to business logic*: Sobald ein Service veröffentlicht wurde, ist er für alle Clients verfügbar. Die Serverlogik benötigt keine Informationen über den Client. Somit können alle Clients den neuen Service uneingeschränkt nutzen.

Die verfügbaren Clientbibliotheken beinhalten eine Kommunikationsverwaltung und eine Datenmodelverwaltung (*Data Model Manager*). Zusammen genommen wird eine vollständige Request/Response-Pipeline geboten, inklusive einer clientseitigen Speicherung von Objekten aus den Teamcenterservice-Antworten (*Response*). So wird es Entwicklern ermöglicht, sich auf die Verwendung der zur Verfügung stehenden Funktionen zu konzentrieren, ohne eine grundlegende Infrastruktur entwickeln zu müssen. Sind die Clientbibliotheken ungenügend oder aus anderen Gründen nicht einsetzbar, können die Anwendung unter Verwendung der WSDL entwickelt werden. In diesem Fall müssen Entwickler eigenständig für die Verwaltung der Kommunikation (Service-Stubs, HTTP Cookies) und des Client Datenmodels sorgen.

Um eine Durchgängigkeit bei der Verwendung der Services zu erreichen wurden alle Teamcenterservices nach den gleichen Mustern und Charakteristika entwickelt. So wurde eine weitgehend einheitliche Nachrichtenstrukur bezüglich des Input und Output realisiert. Im Allgemeinen sind Serviceinputs *set-based*, das bedeutet die Eingabestrukturen entsprechen Arrays oder Vektoren von Strukturen. So können mit einem einzigen Serviceaufruf mehrere Objekte bearbeitet werden. Außerdem wurden die Services so gestal-

tet, dass diese möglichst breit anwendbar sind. Ein Teamcenterservice kann in der Regel auf eine Vielzahl von Geschäftsobjekten angewandt werden. Obwohl *set-orientation* und *broad-applicability* der Standardansatz für die Entwicklung von Services ist gibt es auch Services die spezialisierte Operation für ausgewählte Aufgaben realisieren. Bei diesen steht die Anwendbarkeit für Entwickler im Vordergrund.

3.7.1. Voraussetzungen

Siemens PLM Software [2014a] beinhaltet Voraussetzungen für die Teamcenterservices, die wichtigen werden in diesem Abschnitt beschrieben. Die Teamcenterservices sind ein grundlegender Bestandteil von TEAMCENTER 2007 und späteren Versionen und somit nur für diese und nachfolgenden Versionen verfügbar. Abgesehen von den C++-Bibliotheken, welche nur in 4-Tier-Betrieb unterstützt werden, können die Bibliotheken für JAVA und .NET in allen Betriebsvarianten (2-Tier und 4-Tier) genutzt werden. Es wird sowohl der Betrieb in einer J2EE-Web Schicht, als auch in einer .NET Webschicht unterstützt.

Für die Implementierung der Teamcenterservices ist folgendes notwendig.

- Zugriff auf einen Teamcenterserver mit mindestens TEAMCENTER 2007
- Für Java Entwicklungen, das *Java Development Kit* ab Version 7
- Für C++ Entwicklungen, das Visual Studio 2010 Service Pack 1
- Teamcenter-Clientbibliotheken

3.7.2. Teamcenter Connection

Für die Durchführung eines Serviceaufrufs, muss eine Verbindung mit TEAMCENTER erstellt werden. Mit Hilfe dieser Verbindung können im Zuge eines Serviceaufrufs, je nach Aufgabe, Services-Stubs instanziiert werden. Für die Herstellung einer Verbindung zu TEAMCENTER wird ein *CredentialManager* und ein *Connection*-Objekt benötigt.

Das Teamcenterservice Framework definiert ein *CredentialManager*-Interface, welches im Zuge der Cliententwicklung implementiert werden muss. In der *CredentialManager*-Implementierung könnten Benutzer und Authentifizierungsinformationen gespeichert werden. Je nach Anforderung könnte die Implementierung die Zugangsdaten auch direkt von einem Benutzer über eine Eingabemaske oder über ein anderes Authentifizierungssystem abfragen.

Verbindungen zu TEAMCENTER können aufgrund von langen Inaktivitätszeiten oder Netzwerkfehlern geschlossen werden. In diesen Fällen wird ein Serviceaufruf mit einer *InvalidUser*-Exception beantwortet und der Client muss sich erneut authentifizieren. Anstatt die Clientanwendung einen Fehler werfen zu lassen, verwendet das Service Framework den *CredentialManager* und stellt automatisch eine neue Verbindung her. Sobald

die neue Verbindung erstellt wurde, werden alle Service Aufrufe die eine *InvalidUser*-Exception verursacht haben erneut ausgeführt.

Ein *Connection*-Objekt verwaltet eine Verbindung für einen Benutzer zu einem Server. Ein Client kann mehrere Verbindungen aufbauen, in dem für jede Verbindung eine *Connection* erstellt wird. Die Kommunikation mit einem Server wird erst durchgeführt, wenn ein Serviceaufruf ausgeführt wird.

In Teamcenter existiert für jeden angemeldeten Client ein dezidierter Teamcenter-Serverprozess. Je nach Anwendungsfall kann es notwendig sein, mehrere Anwendungen mit einem Benutzer zu betreiben und den Serverprozess zwischen diesen Anwendungen zu teilen. Zu diesem Zweck wird bei der Anmeldung an einem Teamcenterserver eine Kombination von *Host*, *Benutzername* und *Discriminator* verwendet. Wird von einem Client bei der Anmeldung an einem Teamcenterserver eine bereits existierende Kombination angegeben, wird ein bestehender Teamcenter-Serverprozess zugewiesen.

3.7.3. Service Invocation

Ein typischer Serviceaufruf eines Teamcenterservices beinhaltet folgende Schritte:

1. Befüllen einer Struktur, welche die Geschäftsobjekte oder Operation beschreibt
2. Sammeln der Strukturen in einem der jeweiligen Technologie entsprechenden Set, wie einem Vektor oder einem Array
3. Instanziieren eines entsprechenden Service-Stub
4. Aufrufen der Serviceoperation
5. Auf die Antwort der Operation warten
6. Auf die Antwort entsprechend reagieren

Fast alle angebotenen Services akzeptieren mehrere Inputobjekte, die mit einer Serviceanfrage von der Geschäftslogik (Business Logic) verarbeitet werden können. Die Möglichkeit viele Objekte mit einem einzelnen Aufruf bearbeiten zu können, ist besonders bei Netzwerkverbindungen mit hoher Latenz wichtig. Die Teamcenterservices sind so beschaffen, dass sie in sich geschlossene Aktionen darstellen.

3.7.4. Fehlerhandling

InvalidUser-Exceptions werden automatisch vom *CredentialManager*-Interface abgearbeitet. Im Zuge eines Serviceaufrufs können jedoch auch andere Fehler auftreten. Um auf diese Fehler nicht bei jedem Serviceaufruf zu prüfen und diese anschließend abhandeln zu müssen, definiert das Teamcenterservice Framework ein *ExceptionHandler*-Interface. Dieses Interface kann von Clientanwendungen implementiert werden und dem *Connection*-Objekt mitübergeben werden. Das ermöglicht jedem Client eine eigene Implementierung des Fehlerhandlings.

Im Gegensatz zu den oben beschriebenen Fehlern (*Full Service Errors*), bei denen ein vollständiger Serviceaufruf fehlschlägt, enthält das Teamcenterservice Framework auch einen Mechanismus für teilweise Fehler, sogenannte *PartialErrors*. Der Großteil aller Teamcenterservices sind *set-based*, somit kann es bei der Ausführung vorkommen, dass eine Operation bei einigen Objekten fehlschlägt, bei den Restlichen jedoch erfolgreich ist. Um die Clientanwendung über die Fehler die bei einer Teilmenge der Objekte aufgetreten sind zu informieren, meldet der Server eine Struktur von *PartialErrors* als Teil der Antwort zurück. Eine Clientanwendung hat verschiedene Möglichkeiten *PartialErrors* eines Serviceaufrufs abzufragen:

- Abfragen der *PartialErrors* direkt aus dem *ServiceData*-Objekt
- Registrieren eines *PartialErrorListeners* im Model Manager, es können beliebig viele Implementierungen des *PartialErrorListeners* gleichzeitig registriert werden.
- Bei Aufrufen die kein *ServiceData*-Objekt zurückliefern, sind die *PartialErrors* ein Teil der eigentlichen Serviceantwort.

3.7.5. Client Data Model

Das *ClientDataModel* von Teamcenter oder kurz CDM, besteht aus mehreren Komponenten zur Repräsentation von Objekten, die aus Teamcenter geladen wurden. Die wichtigsten Klassen sind in der com.teamcenter.soa.client.model Bibliothek enthalten und werden in Folge kurz beschrieben.

Das *ModelObject* ist die Grundklasse für alle Geschäftsobjekte aus dem Teamcenter Meta-Model. Ein *ModelObject* kann entweder direkt in einer Antwortstruktur vorkommen, oder in einem *ServiceData*-Objekt.

Jede, von einem Serviceaufruf zurückgemeldete, Instanz eines *ModelObject* beinhaltet eine Menge von Eigenschaften. Welche Eigenschaften für welche Geschäftsobjekte vom Server zurückgemeldet werden, wird über die *Object Property Policy* definiert. Die Clientanwendung kann beliebig viele Policies definieren und entscheiden welche Policy zu einem bestimmten Zeitpunkt gültig ist. Da die Policy definiert wie viele Eigenschaften für einen bestimmten Objekttyp zurückgemeldet werden und somit vom Teamcenter Serverprozess geladen und versendet werden, hat sie einen entscheidenden Einfluss auf die Performance eines Serviceaufrufs. Versucht eine Clientanwendung auf eine Eigenschaft eines *ModelObject* zuzugreifen, die nicht geladen wurde, wird eine *NotLoadedException* geworfen. Das Teamcenterservice Framework versucht nicht automatisch Eigenschaften vom Server nachzuladen.

Die *ServiceData* Klasse ist eine Datenstruktur die in Antworten auf Serviceaufrufe verwendet wird. Sie beinhaltet eine Menge von *ModelObjects*, die nach der durchgeführten Aktion des Serviceaufrufs geordnet sind. Zu den möglichen Aktionen gehören *Created*, *Updated*, *Deleted*, oder *Plain*. *Plain* enthält Objekte die vom Serviceaufruf zurückgelie-

fert wurden, aber nicht in der Datenbank geändert wurden. Es existieren Methoden, mit denen Clientanwendungen auf die Objekte dieser vier Listen zugreifen können. Zusätzlich beinhaltet die *ServiceData*-Struktur eine Liste aller aufgetretenen *PartialErrors*.

Der *ModelManager* wird vom Teamcenterservice Framework verwendet um *ModelObjects* zu verarbeiten die von einem Serviceaufruf zurück gemeldet werden. Mit dem *ModelManager* können verschiedene *ModelEventListener* registriert werden. Diese reagieren auf Events, die in Verbindung mit *ModelObjects* stehen. Eine Clientanwendung kann beliebig viele Listeners registrieren, diese werden vom *ModelManager* aufgerufen falls entsprechende Informationen vom Server zurückgemeldet werden. Auf den *ModelManager* kann über das *Connection*-Objekt zugegriffen werden.

Das *ClientDataModel* ist ein Datenspeicher und enthält alle *ModelObjects*, die von einem Serviceaufruf retourniert wurden. Mittels des *Connection*-Objekt kann auf diesen Speicher zugegriffen werden. Der Speicher ist kumulativ, jedes Mal wenn ein *ModelObject* von einem Serviceaufruf retourniert wird, wird der Speicher erweitert. Ist ein Objekt bereits im Speicher enthalten und wird erneut von einem Serviceaufruf zurückgemeldet, so wird das entsprechende Objekt im Speicher aktualisiert. Objekte werden nur aus dem Speicher entfernt wenn explizite Aufrufe für die Entfernung des Objekts ausgeführt werden oder wenn Objekte vom Server mit einem *Delete*-Event markiert wurden.

Es werden zwei Arten des *ClientDataModels* unterstützt. Zum einen, ein generisches Model (*loose coupled* oder *loose bindings*), bei diesem Model werden alle Geschäftsobjekte durch das *ModelObject* repräsentiert. Zum Anderen ein typsicheres (*strongly typed* oder *type-safe*) Model. Hier entspricht die Klassenhierachie des Modeltyps genau dem Datenmodell in Teamcenter. Die *ModelObject* Klasse ist in diesem Fall die Grundklasse des Geschäftsobjekts und das Client Framework instanziiert eine dem Geschäftsobjekt entsprechende Klasse, zusätzlich stehen hier *Strongly Typed Accessors* (Getter und Setter) für alle Objekte und ihre Eigenschaften zur Verfügung. Ist der entsprechende Type nicht bekannt, wird der nächst bekannte Elterntyp instanziiert. Wurde in einer Teamcenterinstallation das Datenmodell um kundenspezifische Geschäftsobjekte erweitert, müssen diese extra im CDM registriert werden, um vom *strongly typed* Datenmodel instanziiert werden zu können.

Das *ClientMetaModel* beinhaltet eine clientseitige Version der TEAMCENTER Geschäftsmodelldefinition (*Meta-Model*). Über die *Connection* Klasse kann auf die *Meta-Model*-Informationen zugegriffen werden. Auf diesem Weg können *Type* Objekte abgefragt werden. Eine *Type* Klasse beinhaltet Informationen wie Name, Konstanten, Eigenschaften und Beschreibungen für den jeweiligen Objekttyp. Das Clientframework lädt automatisch die Informationen vom Server, falls diese benötigt werden und noch nicht geladen wurden.

3.7.6. Object Property Policy

Die Geschäftslogik eines Teamcenterservices definiert, welche Geschäftsobjekte im Zuge eines Serviceaufrufs zurückgemeldet werden. Die *Object Property Policy* definiert welche Eigenschaften für ein Geschäftsobjekt zurückgegeben werden. Die aktuell gültige Policy gilt immer für alle Serviceaufrufe die mit einer bestimmen Teamcenter-*Connection* durchgeführt werden. Die Clientanwendung definiert, welche *Policy* bei welchen Operationen verwendet werden soll. Wird von der Clientanwendung keine *Policy* definiert, wird automatisch die Default-*Policy* verwendet.

Eine *Object Property Policy* ist eine Liste von TEAMCENTER Klassen/Geschäftsobjekten und den zugehörigen Eigenschaften. Eigenschaften die für eine Elternklasse definiert werden, werden automatisch vererbt. Das Teamcenterservice Framework bietet eine Reihe von Optionen die für eine *Object Property Policy* gesetzt werden können. Mit diesen Optionen kann das Verhalten genauer gesteuert werden. Für eine Liste dieser Eigenschaften wird auf die offizielle Dokumentation verwiesen.

Die Anzahl der im Zuge eines Serviceaufrufs zurückgemeldeten Eigenschaften, hat einen erheblichen Einfluss auf die Performance eines Aufrufs. Daher ist es von entscheidender Bedeutung für die Cliententwicklung, je nach Aufgabe die richtige *Policy* zu erstellen und zu verwenden.

3.7.7. Gliederung der Teamcenterservices

Die Menge der Services ist sehr umfangreich und wächst mit der Veröffentlichung neuer Teamcenterversionen stetig. Um einen besseren Überblick über die verschiedenen Services zu haben, wurden diese in eine Hierarchie von Funktionalen Gruppen, Bibliotheken, Services und Operationen eingeteilt. Die Services können in sechs Funktionalen Gruppen eingeteilt werden. Es gibt noch eine siebente Gruppe, die Gruppe der Distributed Systems welche Verbindungen und Funktionen über und zwischen mehreren Teamcenterinstallationen (Mulitsite) beinhaltet. Diese Services sind jedoch von geringer Bedeutung für diese Arbeit und werden aus diesem Grund nicht angeführt.

- *Applications* - Beinhaltet Bibliotheken für spezielle Aufgaben oder für spezielle Lösungen
- *Application Support* - Beinhaltet Bibliotheken für Aufgaben die von Benutzerapplikationen regelmäßig benötigt werden. Darunter fallen unter anderem Abfragen, Suchen, Workflows, aber auch Reports.
- *Application Integration* - Beinhaltet Bibliotheken für die Verbindung zu externen Anwendungen wie CAD oder CAM Tools
- *System Administration* - Beinhaltet Bibliotheken für regelmäßige Service und Administrationsaufgaben

- *System Definition* - Beinhaltet Bibliotheken, die verwendet werden, um Teamcenterinstallationen an Anforderungen der Kunden anzupassen.
- *Platform* - Beinhaltet Bibliotheken für grundlegende Aufgabe von Teamcenter wie unter anderem das Speichern, Verwalten und Abfragen von Informationen in und aus Teamcenter. Dieser Bereich stellt den Kern der Teamcenter Funktionen dar.

4. Technologien und Frameworks

Um ein besseres Verständnis für die anschließenden Kapitel in dieser Arbeit zu schaffen, werden in diesem Kapitel das APACHE CAMEL Framework, das SPRING Framework, das Java Message Service (JMS) und WebService Technologien kurz beschrieben und erläutert.

4.1. Apache Camel

In Ibsen and Anstey [2010] wird APACHE CAMEL als Framework zur Erstellung von Integrationslösungen basierend auf den EIPs beschrieben. Das APACHE CAMEL Projekt wurde Anfang 2007 gestartet und ist als Open Source Software unter der Apache 2.0 Lizenz veröffentlicht. Ziel des APACHE CAMEL Integrations Frameworks ist es, Integrationsaufgaben zu vereinfachen. Dazu werden viele der in Kapitel 2.2.3 beschriebenen EIPs implementiert. EIPs ermöglichen eine allgemeine Beschreibung von Integrationslösungen, ohne auf Details einer Programmierung eingehen zu müssen.

Das APACHE CAMEL Framework bietet umfangreiche Funktionen. In Ibsen and Anstey [2010] werden einige der Grundideen beschrieben. Dazu gehören:

- *Routing and mediation engine* - ermöglicht eine Nachrichtenweiterleitung und Übersetzung
- *Enterprise integration patterns (EIPs)* - ermöglichen die Konzentration auf die Lösung von Geschäftsproblemen.
- *Domain-specific language (DSL)* - ermöglicht eine komfortable Definition von Routen in unterschiedlichen Sprachen.
- *Extensive component library* - eine umfangreiche Sammlung an Komponenten
- *Modular and plugable architecture* - nur benötigte Teile des Frameworks müssen verwendet werden
- *Plain Old Java Object (POJO) model* - ermöglicht die Verwendung von einfachen POJOs
- *Automatic type converters* - eingebaute Typumwandlung
- *Test kit* - umfangreiche Testmodule

APACHE CAMEL verwendet ein Message Model um Nachrichten zwischen Systemen auszutauschen. Es werden zwei Arten von Objekten für die Modellierung von Nachrichten in Camel verwendet. Einerseits existiert die *org.apache.camel.Message*, andererseits der *org.apache.camel.Exchange*.

Eine *Message* besteht aus einer Menge von *Headern*, *Attachments*, einem *Body* und einer *Fault Flag*. *Header* sind Paare von Namen und Wert. Typische Beispiele für *Header* sind Kodierungsinformationen oder Authentifizierungsinformationen. Der Name ist vom Typ *String* und muss einzigartig sein. Der Wert kann ein beliebiges Objekt von Typ

java.lang.Object sein. Zusätzlich kann eine Nachricht in Camel *Attachments* beinhalten, diese werden typischerweise für WebServices oder Emailkomponenten verwendet. Der *Body* ist vom Typ *java.lang.Object* und beinhaltetet meist den Hauptteil der zu übertragenden Informationen. Einige Protokolle und Spezifikationen unterscheiden zwischen Ausgabe und Fault Message. Beide sind gültige Ergebnisse einer Operation. Letztere zeigt jedoch den nicht erfolgreichen Ausgang einer Operation. Das *Fault Flag* wird verwendet um anzuzeigen ob eine Operation erfolgreich war oder nicht.

Ein *Exchange* ist ein Container für Nachrichten und wird verwendet um Nachrichten während des Routings auszutauschen. Zwischen den einzelnen Bearbeitungsschritten in einer Route wird jeweils ein *Exchange* weitergegeben. Abbildung 20 zeigt die Bestandteile eines *Exchanges*. Jeder *Exchange* besteht aus zwei Nachrichten vom Typ *org.apache.camel.Message*, einer Eingangsnachricht und einer Ausgangsnachricht. Die Eingangsnachricht beinhaltet die eingehenden Informationen, die Ausgangsnachricht ist zu Beginn der Bearbeitung leer. Am Ende eines Bearbeitungsschritts wird die Ausgangsnachricht des *Exchanges* automatisch zur Eingangsnachricht des nächsten *Exchanges*. Ist die Ausgangsnachricht am Ende eines Bearbeitungsschrittes leer, wird automatisch die Eingangsnachricht des ursprünglichen *Exchanges*, zur Eingangsnachricht des nachfolgenden *Exchanges*. Zusätzlich zu den Nachrichten besitzt ein *Exchange* noch weitere Informationen, eine wichtige Information ist das *Message Exchange Pattern* (*MEPs*) (siehe Abschnitt 2.2.3). Mit dem MEP wird die Art der durchzuführenden Interaktion definiert. Das MEP kann *InOnly* oder als *InOut* definiert werden und definiert, ob Nachrichten nur an den Nachfolger übergeben werden (one way) oder ob die Ergebnisse des Nachfolgers wieder an den Aufrufenden zurückgemeldet sollen (request-reply). Weiters beinhaltet ein *Exchange* eine *Exchange ID* mit der eine eindeutige Identifizierung möglich ist, *Exceptions* welche gesetzt werden, wenn Fehler auftreten und eine Menge von *Properties*. *Properties* sind vergleichbar mit Message Header, sind jedoch für den gesamten *Exchanges* gültig.

Der *CamelContext* ist die Laufzeitkomponente von Camel. Durch den *CamelContext* werden Routen verwaltet und die einzelnen Komponenten zusammen gehalten. Zusätzlich bietet der *CamelContext* eine Palette an Services um eine Camelanwendung verwalten zu können.

Routen sind ein Kernelement von Camel. Sie definieren die einzelnen Bearbeitungsschritte als Abfolge von Prozessoren bzw. Endpunkten. Eine Route kann als Graph betrachtet werden, die Knoten sind Prozessoren und die Kanten verbinden den Output eines Prozessors mit dem Input des nachfolgenden Prozessors.

Definiert werden Routen mit Hilfe von *Domain-Specific Languages* (DSL). Camel bietet verschiedene DSL für unterschiedliche Programmier-/Skriptsprachen, darunter sind Java, Groovy, Scala und auch eine DSL für Spring ist vorhanden. Bei der Definition einer Route wird mit Hilfe einer DSL eine Liste von Operationen bzw. Kommandos erstellt.

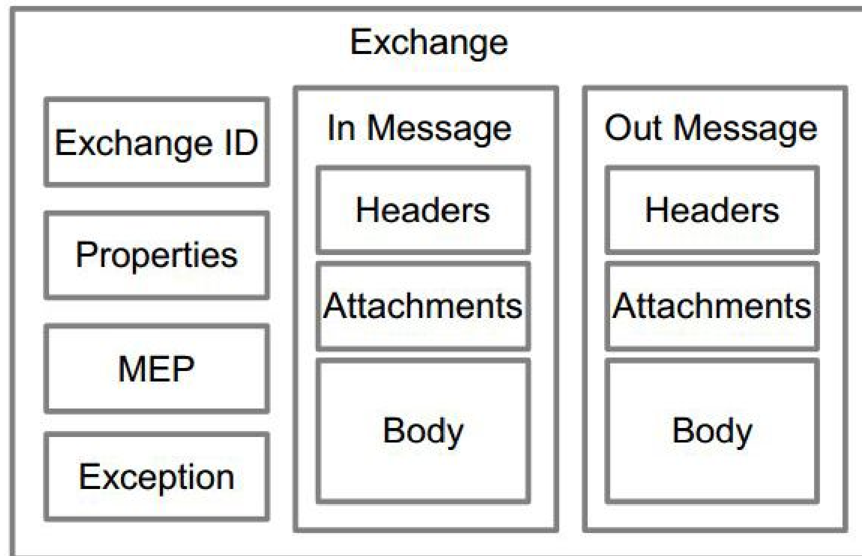

Abbildung 20: Exchange in Apache Camel

```
Operation1().Operation2()...OperationN();
```

Prozessoren (Processor) sind Teile einer Route und ermöglichen die Bearbeitung eingehender Nachrichten. Vereinfacht dargestellt sind Prozessoren für die Bearbeitung zwischen Endpunkten zuständig. Camel bietet eine große Auswahl an bestehenden Prozessoren, viele davon sind Implementierungen von EIPs, dazu gehören unter anderem Filter, Router oder Translator. Zusätzlich zu den bestehenden Prozessoren können neue Prozessoren entwickelt werden.

In Integrationsprojekten ist es notwendig direkt mit den zu verknüpfenden/externen Technologien oder Systeme zu kommunizieren. Camel abstrahiert und kapselt diese Aufgabe in sogenannte *Komponenten* (Components). Das ermöglicht den Entwicklern sich auf die Geschäftslogik der jeweiligen Lösung zu konzetrieren ohne auf die Details der Kommunikation achten zu müssen. Aktuell gibt es über 80 Komponenten für unterschiedlichste Aufgaben und für die Kopplung verschiedenster Technologien und Anwendungen. Jede Komponente hat einen Namen, der sie in einer Route eindeutig identifiziert und mit dem sie in einer Route aktiviert werden kann. Im weiteren Sinn ist eine *Komponente* eine Fabrik für *Endpunkte* und übernimmt die Anlage und Verwaltung der in Routen spezifizierten *Endpunkten*. Ein Beispiel ist die Komponente für den Java Message Service (JMS). Wird in einer Route ein JMS-Endpunkt spezifiziert, erstellt die *Komponente* einen *Endpunkt* für die jeweils definierte JMS-Verbindung. Der erstellte *Endpunkt* übernimmt anschließend den eigentlichen Datenaustausch mit dem jeweiligen JMS-Broker.

Über *Endpunkte* können Nachrichten empfangen oder gesendet werden. Konfiguriert werden *Endpunkte* durch die Verwendung eines *Uniform Resource Identifier* (URI). Ein URI besteht aus 3 Teilen, URI = *Scheme* : *ContextPath* ? *Optionen* (z.B.: *fi-*

le:data/infolder?delay=1000). Das Scheme definiert welche Komponente für die Erstellung des Endpunkts zuständig ist, im genannten Beispiel definiert das Scheme *file* die Verwendung einer *FileComponent*. Basierend auf dem URI erzeugt die *FileComponent* einen *FileEndpoint*. In diesem Beispiel definiert *data/infolder* den Startordner für die *FileComponent*. Mit Optionen können *Endpunkte* noch genauer spezifiziert werden. Welche Optionen zur Verfügung stehen, hängt vom Endpunkt ab und ist in der jeweiligen Dokumentation beschrieben. In diesem Fall definiert *delay=1000*, dass der Ordner jede Sekunde auf neue Dateien geprüft wird.

Consumer oder *Producer* sind fähig, Nachrichten an Endpunkte zu senden oder zu empfangen. Wenn eine Nachricht an einen Endpunkt gesendet werden soll, wird von einem *Producer* ein Exchange erzeugt und mit den notwendigen Daten für den Endpunkt befüllt. Im oben genannten Beispiel des *FileEndpoint*, wird ein *FileProducer* erzeugt. Dieser schreibt den Körper der Nachrichten in eine Datei. Dieses Konzept ist sehr wichtig in Camel, denn es versteckt die Komplexität eines Transports. Es muss nur eine Nachricht an einen *Endpunkt* geroutet werden, die Arbeit wird anschließend vom *Producer* übernommen.

Ein *Consumer* ist ein Dienst der Nachrichten empfängt, in einen *Exchange* packt und an einen *Prozessor* sendet. Der *Prozessor* initiiert in der Folge das Routing des *Exchanges* durch die Routing Engine. Es gibt zwei verschiedene Arten von *Consumer*. Einen *ereignisgesteuerten Consumer*, der einen Nachrichtenkanal überwacht. Sobald eine Nachricht eintrifft, erwacht der Consumer und übernimmt die Nachricht für die weitere Bearbeitung. In der Welt der EIPs wird dieser *Consumer* auch *Asyncronous Reciever* genannt. Die zweite Art ist ein *abfragender Consumer*. Dieser versucht in regelmäßigen Zeitabständen aktiv Nachrichten aus einer Quelle abzufragen. Im Zusammenhang mit den EIPs wird von einem *Synchronous Reciever* gesprochen. Der Name begründet sich darauf, dass eine solcher Consumer weitere Nachrichten erst abfragt wenn die aktuelle Nachricht fertig bearbeitet wurde.

Abbildung 21: Arten von Consumer

4.1.1. Eine einfache Applikation

Um die grundlegenden Ideen und Konzepte von Apache Camel besser verstehen zu können wird eine einfache Apache Camel Anwendung untersucht. Für die Erstellung

dieses Beispiels wird die Java DSL von Camel verwendet. In diesem Beispiel wird von einem Wetterdienst eine Wetterinformation abgefragt und in eine Datei geschrieben.

Listing 1: MyFirstCamelApp

```
1  public class MyFirstCamelApp {
2      public static void main(String[] args) throws Exception{
3          CamelContext context = new DefaultCamelContext();
4          //anlegen einer neuen Route
5          context.addRoutes(new RouteBuilder() {
6              public void configure(){
7                  from("http:api.openweathermap.org/data/2.5/weather?q=Vienna")
8                      .to("file:target/messages/others");
9              }
10         });
11         context.start();        //Camel starten
12         Thread.sleep(10000);    //10 Sekunden warten
13         context.stop();       //Camel stoppen
14     }
15 }
```

Apache Camel kann auf verschiedene Arten gestartet und betrieben werden. In komplexeren Anwendungsfällen wird Camel über Spring oder direkt in einem WebContainer betrieben. In diesem Beispiel wird Camel in einer einfachen Java Anwendung gestartet. Zuerst wird in Zeile 3 in Listing 1 ein *CamelContext* erstellt, der *CamelContext* stellt die Laufzeitumgebung für Camel bereit. Anschließend wird im *CamelContext* mit Hilfe eines *Routebuilders* eine neue Route erstellt. Die Routen selbst werden mit Hilfe einer DSL definiert, das ermöglicht die Konzentration auf die Erstellung und Definition der Routen. Der *CamelContext* instanziiert später alle benötigten Komponenten, Endpunkte, Konsumenten, Produzenten, usw. automatisch. Jede *Route* beginnt mit dem Schlüsselwort *from()*, zusätzlich wird innerhalb von runden Klammern der URI für den gewünschten Endpunkt angegeben. Alle weiteren Schritte in der Route werden mit „.“ und weiteren Schlüsselwörtern (z.b.: *to()*) definiert. Mit „;“ wird die Definition der Route beendet.

In diesem Beispiel wird eine HTTP-Komponente und eine File-Komponente verwendet. Camel erzeugt für jede der angegebenen Komponenten einen Endpunkt, der Endpunkt wiederum instanziiert einen Producer oder Consumer sobald die Route aufgerufen wird. Da die HTTP-Komponente in dem *from* Teil der Route verwendet wird (siehe Zeile 7 in Listing 1), wird von dem Endpunkt ein Consumer erzeugt. Dieser holt die Wetterinformationen von Wien, indem eine HTTP-Anfrage an die entsprechende Adresse versendet wird, die Antwort auf die Anfrage wird in einem Exchange verpackt und an den nächsten Bearbeitungsschritt der Route übergeben. In diesem Fall an den File-Endpunkt. Da die File-Komponente in einem *to* Teil der Route verwendet wurde (siehe Zeile 8 in Listing 1), wird vom Endpunkt bei jedem Durchlauf der Route ein *File-Producer* erzeugt, dieser speichert die Wetterinformation aus dem Exchange in eine Datei in den angegebenen Ordner (*target/messages/others*).

Um die Routen auszuführen, wird mit *context.start(),* Camel gestartet und anschließend kurz gewartet. So kann die Route durchlaufen werden, abschließend wird Camel wieder gestoppt.

4.1.2. Expressions und Predicates

Expressions und *Predicates* werden an vielen Stellen in Apache Camel verwendet und sind vielseitig einsetzbar. *Expressions* und *Predicates* können komfortabel in Routen definiert werden und ermöglichen das Erzeugen von dynamischen Inhalten, oder die Evaluierung von Bedingungen.

Eine *Expression* (org.apache.camel.Expression) wird zur Laufzeit berechnet, indem der aktuelle *Exchange* ausgewertet wird. Eigene *Expressions* können erstellt werden indem das Interface *Expression* implementiert wird. Alternativ kann auch die Klasse *ExpressionAdapter* erweitert werden.

Ein *Predicate* (org.apache.camel.Predicate) ist eine spezielle Form einer *Expression* die einen Rückgabewert vom Typ Boolean hat. Eigene *Predicates* können durch die Implementierung des Interfaces *Predicate* erstellt werden. *Predicates* werden oft genutzt um Entscheidungen in einem Router oder Filter zu treffen.

Camel bietet verschiedene Möglichkeiten *Expressions* oder *Predicates* in Routen zu verwenden, dazu gehört die *Simple Language*[4] oder verschiedene Scripting Sprachen[5].

4.1.3. Type Converter

Apache Camel bietet mit den *TypeConvertern* eine Möglichkeit zur Typumwandlung zwischen bekannten Datentypen. So können verschiedene Komponenten von Camel zusammenarbeiten, ohne den gleichen Datentyp zu verwenden. Wird eine Typeumwandlung benötigt, durchsucht Camel die *TypeConverterRegistry* nach einer geeigneten Methode für die Umwandlung. Die *TypeConverterRegistry* wird beim Start von Camel angelegt, indem alle in der Datei META−INF/services/org/apache/camel/TypeConverter definierten Klassen nach geeigneten Methoden durchsucht werden. Alle Methoden die mit einer @Converter Annotation versehen sind, werden als *TypeConverter* erkannt und zu der Registry hinzugefügt.

4.1.4. Zusammenfassung

Apache Camel vereinfacht die Integrationsaufgaben durch den konsequenten Einsatz von EIPs. Durch den Einsatz einer DSL bietet Camel eine komfortable und flexible Möglichkeit um Routen zu definieren. Der Betrieb von Camel ist in unterschiedlichsten Konfigurationen möglich, Standalone, eingebettet in anderen Applikationen oder in einem Container. Camel ist ein vielseitiges Framework und ermöglicht Entwicklern zu jederzeit die Kontrolle zu behalten. Durch die breite Palette an verfügbaren Komponenten und die Möglichkeit neue Komponenten zu entwickeln, ist eine Verbindung zu einer Vielzahl von Anwendungen und Systemen möglich. Für einen tieferen Einblick in die

[4]http://camel.apache.org/simple.html
[5]http://camel.apache.org/scripting-languages.html

Mechaniken und Funktionen von Camel wird auf Ibsen and Anstey [2010] und Cranton and Korab [2013] verwiesen. Zusätzlich ist auf der Webpräsenz von Apache Camel[6] eine detailierte Dokumentation inklusive Anwendungsbeispielen zu finden.

4.2. Spring

SPRING[7] ist ein Open Source Framework und wurde erstmal offiziell 2004 in der Version 1.0 unter der Apache 2.0 Lizenz veröffentlicht. Der Zweck des SPRING Frameworks ist die Entwicklung von Java Enterprise Anwendungen zu vereinfachen. In Walls [2011] werden die Folgenden Kernstrategien von SPRING beschrieben.

- *Leichtgewichtige und minimal invasive Entwicklung mit Plain Old Java Objects (POJOs)* - Ursprünglich forderten viele Frameworks, wie EJB, bei der Entwicklung einer Bean die Implementierung vieler Methoden. Der Großteil der Methoden erzeugt keinen Mehrwert und wurde nur implementiert, weil das jeweilige Framework diese erforderte. SPRING vermeidet solche Erweiterungen des Anwendungscodes so weit wie möglich. Meist gibt es in Klassen die von SPRING genutzt werden keinen Hinweis darauf, dass sie in SPRING verwendet werden. Falls Eingriffe in Klassen vorgenommen werden, sind das Annotations, sonst bleibt die Klasse ein einfaches POJO. Das erleichtert die Verwaltung, Pflege und Wiederverwendung des Codes
- *Lose Kopplung durch Dependency Injection und Schnittstellenorientierung* - Traditionell ist jedes Objekt für die Verwaltung seiner Abhängigkeiten (Dependencies) zuständig. Mit *Dependency Injection* (DI) werden Abhängigkeiten zur Laufzeit den Objekten zugeordnet. *Dependency Injection* (DI) ist ein Begriff von Martin Fowler und ist eine treffendere Bezeichnung des Prinzips der *Inversion of Control* (IoC). Ho and Harrop [2012] zeigt wie IoC bzw. DI auf zwei Kernelemente von Java aufbauen, einerseits die Java Beans und andereseits Interfaces. DI wurde über die Jahre zu einer Best Practice in der Programmierung und 2009 in einem formalen Java Specification Request (JSR-330) definiert. Der JSR330 wurde später in Java Enterprise Edition 6 (JEE6) umgesetzt.
- *Beschreibende Programmierung durch Aspekte und Common Conventions* - *Aspect Oriented Programming* (AOP) gliedert Funktionen, die an vielen Stellen einer Anwendung immer wieder benötigt werden, in wiederverwendbare Komponenten. Zum Beispiel wird eine Logging Komponente an vielen unterschiedlichen Stellen einer Anwendung benötigt. Anstatt sie jedes Mal neu zu implementieren, wird eine zentrale Komponente verwendet. Dadurch wird der Verwaltungsaufwand erheblich reduziert. In der Literatur gibt es viele Bücher die sich mit AOP beschäftigen, dazu gehört AspectJ in Action von Laddad [2003] oder das AspectJ Cookbook von Miles [2004]. Für genauere Informationen zu AOP wird auf diese Bücher verwiesen.

[6]http://camel.apache.org/
[7]HTTPS://SPRING.IO/

- *Reduktion von Boilderplate Code durch Aspekte und Vorlagen* - SPRING versucht wiederkehrenden Code in wiederverwendbare Vorlagen zu kapseln. Ein Beispiel dafür ist eine JDBC Operation, für fast jede JDBC Operation muss beinahe der exakt selbe Code geschrieben werden, mit Hilfe des JDBC-Templates von SPRING wird das vereinfacht.

Hinter SPRING verbirgt sich aber noch wesentlich mehr, um die Java Entwicklung einfacher zu gestalten. Wheeler [2013] beschreibt sechs grobe funktionale Bereiche aus denen das Framework besteht:

- *Core Container* - beinhaltet die grundlegenden Funktionen auf die alle weiteren Module aufgebaut sind. Dazu gehört der DI-Container der die Entkopplung der Anlage, Konfiguration und Verwaltung von Beans ermöglicht.
- *Web* - Das Web Modul bietet Unterstützung für allgemeine Aufgaben der Web Infrastruktur, einem Model-View-Controller (MVC) Framework und integriert sich mit vielen Web-Entwicklungsframeworks und Technologien wie Struts, Velocity, FreeMarker und vielen weiteren.
- *Instrumentation* - Bietet Performance und Nutzungsinformation über den SPRING Container und ermöglicht eine Kontrolle über Container zur Laufzeit.
- *Aspect-Oriented Programming* - Das Framework unterstützt zwei Ansätze für AOP, einerseits SPRING AOP und andererseites AspectJ und ermöglicht so die Entkopplung von bereichsübergreifenden Funktionen, wie Sicherheit oder Logging.
- *Data Access/Integration* - Datenzugriff und Integration bieten Unterstützung für Java Database Connectivity API (JDBC), object-relational mapping (ORM), Objekt/XML Mapping (OXM), Java Messaging Service (JMS) und Transaktionssupport.
- *Test* - Dieses Modul bietet Unterstützung für JUnit und TestNG Frameworks

Zusätzlich zu den Kernfunktionen können bei Bedarf weitere Module von SPRING genutzt werden. SPRING ist sehr umfangreich, für den Einstieg empfiehlt es sich, mit den Kernmodulen zu beginnen und anschließend bei Bedarf zusätzlich Module zu lernen. Es gibt neben einer Online Dokumentation eine Menge von guten Büchern wie SPRING IN ACTION (siehe Walls [2011]) oder SPRING IN PRACTICE (siehe Wheeler [2013]), die beim Einstieg helfen können.

4.3. Java Message Service

Der JAVA MESSAGE SERVICE oder kurz JMS ist definiert als JSR 914[8]. JMS ist eine Message Oriended Middleware und erlaubt Anwendungen Nachrichten zu erzeugen, zu lesen, zu senden und zu empfangen. JMS hat zum Ziel Anwendungen lose zu koppeln.

[8] https://jcp.org/aboutJava/communityprocess/final/jsr914/index.html

Dazu wird eine asynchrone Kommunikation eingesetzt und ein Weg geboten, Nachrichten verlässlich zu verteilen.

In der JMS-Spezifikation werden folgende Bestandteile einer JMS-Anwendung definiert

- *JMS Client* - Das sind die Java Programme die Nachrichten versenden und empfangen können
- *Non-JMS Clients* - Das sind die Clients, welche die native API der Clientimplementierung anstatt JMS verwenden
- *Nachrichten* - Das sind die Entitäten, die verwendet werden um Informationen zwischen Clients auszutauschen.
- *JMS Provider* - Das ist das Messaging System, das JMS implementiert
- *Administrative Objekte* - Objekte für die Konfiguration, typischerweise JMS Ziele oder Connection Factories.

Im Weiteren werden durch den JSR 914, zwei Nachrichtenstile definiert

- *Point-to-Point* - Bei diesem Stil werden Warteschlangen eingesetzt. Ein Client kann eine Nachricht an eine Warteschlange senden. Eine Warteschlange kann beliebige Arten von Nachrichten beinhalten. Clients können Nachrichten aus einer Warteschlange entnehmen.
- *Publish and Subscribe* - Ein Client kann Nachrichten in einem *Topic* veröffentlichen und Nachrichten eines Topics abonnieren. Ein Topic kann als Art Broker verstanden werden, er sammelt Nachrichten und leitet sie an die Abonnenten weiter.

Aktuell gibt es eine breite Palette an JMS Providern, dazu gehört die Referenz Implementierung Open MQ[9], oder bekannte Implementierungen wie ActiveMQ[10] oder RabbitMQ[11].

4.4. Webservices

Ein WebService ist eine Funktion, auf die über Netzwerkprotokolle zugegriffen werden kann. Heutzutage sind WebServices fester Bestandteil in vielen Bereichen des alltäglichen Lebens, zum Beispiel beim Aufrufen einer dynamischen Webseite. Über ein standardisiertes Interface, in diesem Fall dem HTTP Protokoll, kann eine Webseite angefordert werden. Anschließend wird die HTML Webseite von einem Service erstellt und als standardisierte HTTP Antwort an den Benutzer zurückgegeben. Mit welcher Technologie die HTML Webseite erstellt wird, ist für den Aufruf weder ersichtlich, noch relevant. Es zählt einzig, dass als Antwort auf den standardisierten Aufruf die HTML Webseite zurückgegeben wird.

In Kalin [2013] werden folgende Eigenschaften, die WebServices ausmachen, beschrieben.

[9] https://mq.java.net/
[10] http://activemq.apache.org/
[11] http://www.rabbitmq.com/

- *Offene Infrastruktur*: Web Services werden mit Hilfe von standardisierten, Anbieter unabhängigen Protokollen und Sprachen, wie HTTP, XML oder JSON betrieben.
- *Plattform und Sprach Transparenz*: Web Services und Clients können interagieren, auch wenn diese in unterschiedlichen Programmiersprachen umgesetzt wurden. Web Services können auf unterschiedlichen Hardware Plattformen und Betriebssystemen betrieben und konsumiert werden. Web Services bieten so eine gute Möglichkeit unterschiedliche Software Systeme miteinander zu integrieren und bieten dabei den Entwicklern die Möglichkeit mit der jeweils bevorzugten Sprache zu arbeiten.
- *Modulares Design*: Web Services sollen als Module mit klar abgegrenzten Funktionen designed werden. So können neue Services durch eine Komposition bestehender Services erzeugt werden.

4.4.1. SOAP

SOAP ist ein standardisiertes Protokoll zum Verteilen von Nachrichten zwischen verschiedenen Anwendungen. SOAP basiert auf der XML Spezifikation und nutzt XML Standards wie XML Schema und XML Namespace für die Definition und Funktion. Obwohl SOAP mit verschiedenen Transportprotokollen funktioniert, wird meist HTTP eingesetzt. SOAP ermöglicht Clientanwendungen unabhängig von Plattform und Sprachen zu kommunizieren. Es existieren Implementierungen für unterschiedliche Sprachen wie Java, Perl, C#, JavaScript, C, C++, Python und viele mehr.

In Cerami [2002] werden drei wesentliche Bestandteile der SOAP Spezifikation[12] beschrieben.

- *SOAP Envelope* - Der SOAP XML Envelope definiert wie Informationen abgekapselt werden, um sie zwischen Anwendungen transferieren zu können.
- *Data Encoding Rules* - SOAP inkludiert eine Menge von Konventionen für die Kodierung von Informationen. Die meisten dieser Konventionen basieren auf den W3C XML Schema Spezifikationen
- *RPC Conventions* - SOAP kann in Einweg oder in Zweiweg (Request/Response) Nachrichtensystemen verwendet werden. SOAP definiert eine Konvention für die Abbildung von entfernten Anfragen und der zugehörigen Antworten

Jede Einweg-Nachricht, Anfrage oder Antwort wird durch eine *SOAP Message* in Form eines XML Dokuments repräsentiert. Jede SOAP Nachricht hat ein verpflichtendes *Envelope* Element. Diese beinhaltet ein optionales *Header* Element und ein verpflichtendes *Body* Element. Jedes dieser Elemente ist mit einer Menge von Regeln verbunden.

[12] http://www.w3.org/TR/soap/

4.4.2. REST

Das World Wide Web ist Teil unseres alltäglichen Lebens, aber warum ist das Web so erfolgreich? Roy Fielding stellt in seiner Dissertation *„Architectural Styles and the Design of Network-based Software Architecture“* Fielding [2000] diese Frage. In seiner Arbeit identifiziert er Prinzipien, wie eine Architektur aufgebaut sein muss, um so erfolgreich sein zu können wie das World Wide Web. Diese Architekturprinzipien sind unter dem Namen *REpresentational State Transfer* oder kurz REST bekannt und umfassen

- *Addressable resources* - Jede Ressource muss eindeutig identifizierbar sein. Für die Adressierung wird ein Uniform Resource Identifier (URI) verwendet.
- *A uniform, constrained interface* - Das Prinzip besagt, dass bei der Kommunikation ein einheitliches Protokoll verwendet wird und alle Services mit den Möglichkeiten des Protokolls realisiert werden. Viele Technologien wie SOAP und WS-* nutzen HTTP nur als Transportprotokoll. HTTP bietet aber wesentlich mehr Möglichkeiten. REST bedingt nicht den Einsatz von HTTP als Protokoll, jedoch ist es sicherlich die am weitesten verbreitete Variante. Im Fall von REST mit HTTP besagt das *uniform, constrained interface*, dass ausschließlich die von HTTP definierten Operationsmethoden verwendet werden dürfen. Im Fall von HTTP sind das GET, PUT, DELETE, POST, HEAD und OPTIONS. Mit diesem Set von Methoden müssen alle Funktionen realisiert werden.
- *Representation-oriented* - Das Prinzip definiert, dass ein Objekt in verschiedenen Formaten unter einem URI verfügbar sein soll. So können unterschiedliche Plattformen die gleiche Ressource verwenden, bekommen jedoch die jeweils benötigte Representation der Ressource wie z.B.: XML, JSON oder YAML.
- *Communicate statelessly* - Statuslose Applikationen können einfacher skaliert werden.
- *Hypermedia As The Engine Of Application State (HATEOAS)* - Im Gegensatz zu SOAP, wo Client und Server auf Basis einer fixen Schnittstelle interagieren, interagiert ein REST-Client nur mit Hilfe der Hypermediainformationen, die dynamisch vom Applikationsserver bereitgestellt werden. Dabei greift der REST-Client bei der ersten Kommunikation auf einen festen URI zu. Alle weiteren verfügbaren Aktionen werden vom Server in Form von Links an den Client zurückgemeldet. Der Client bewegt sich anschließend mit Hilfe dieser Links durch die Applikation. Die verfügbaren Interaktionen werden vom Server im Hypermedia definiert.

Durch die Anwendung der Prinzipien auf die Welt der WebServices entsteht eine flexibler Weg WebServices zu entwickeln.

5. Entwicklung einer Integrationlösung für Teamcenter

In dem Hauptteil dieser Arbeit wird eine Lösung zur Integration von Teamcenter mit anderen Anwendungen vorgestellt. Dazu wird im ersten Schritt ein geeignetes Integrationsframework ausgewählt. Anschließend werden die nötigen Erweiterungen, um Teamcenter mit dem Framework integrieren zu können, entwickelt und beschrieben.

Eine zentrale Anforderung an die Integrationslösung ist, einen möglichst hohen Wiederverwendungswert zu erzielen. Die Integrationslösung soll mit möglichst geringem Programmieraufwand in beliebigen Integrationsprojekten für Teamcenter einsetzbar sein. Da sich in der Realität Anforderungen an Integrationsprojekte nur in sehr seltenen Fällen exakt wiederholen, wird ein Schwerpunkt auf Anpassbarkeit und Konfigurierbarkeit der Lösung gelegt. Abschließend werden Integrationsszenarien in Form von Fallbeispielen behandelt. Es wird jeweils zuerst die Problemstellung erörtert, dann eine Integrationslösung mit Hilfe der Enterprise Integration Pattern beschrieben und abschließend die Lösung mit Hilfe des gewählten Frameworks umgesetzt.

5.1. Auswahl eines Frameworks

Für die Integration von Teamcenter mit anderen Anwendungen, wurde Apache Camel gewählt. Folgende Gründe führten zu dieser Entscheidung.

- Veröffentlicht unter der Apache Lizenz - kann somit beliebig verwendet, erweitert und angepasst werden
- leichtgewichtiges Framework
- unterschiedlichste Betriebsmöglichkeiten - Standalone, eingebettet in Anwendungen, oder in einem Container
- Hohe Verfügbarkeit von Komponenten - Camel bietet eine große Auswahl an Komponenten
- Komponenten können erweitert werden und neu entwickelt werden
- Große Community
- Sehr gute Integration mit dem Spring-Framework für die Verwendung eines IoC Containers
- Domain Specific Language - Camel bietet eine DSL für unterschiedliche Sprachen, wie Java, XML, Groovy, Scala
- Top Level Projekt bei Apache - eine Weiterentwicklung und Weiterführung des Projekts ist gesichert
- Mediation Engine ausreichend - Teamcenter wird mit anderen Anwendungen integriert und nicht eine Menge von Anwendungen untereinander.

Apache Camel bietet somit eine hervorragende Grundlage um Integrationslösungen zu schaffen. Camel bietet eine Vielzahl von unterschiedlichen Komponenten um Verbin-

dungen mit den gängigsten Anwendungen und Systemen aufzubauen. Zusätzlich ist es sehr gut mit SPRING integriert und bietet eine eigene DSL für Spring. Auf diesem Weg können Routen und Integrationslösungen, ohne weitere Programmierung und Kompilierung, über XML Dokumente konfiguriert werden. Durch die Erstellung eigener Komponenten können für standardisierbare Aufgaben wiederverwendbare Lösungen entwickelt werden. Das ist speziell für die Verbindung zu TEAMCENTER notwendig. Da aktuell keine Teamcenter-Komponente für Camel exisitert, diese jedoch ein essentieller Teil jedes Integrationsprojekts mit TEAMCENTER ist, wird eine solche Komponente entwickelt.

Da FUSE ESB auf APACHE CAMEL basiert, können die Camel-Komponenten bei Bedarf auch im Rahmen eines umfangreichen ESB betrieben werden. Die Teamcenterkomponente für Camel kann so in unterschiedlichsten Typen von Integrationslösungen (siehe Abschnitt 2) verwendet werden.

5.2. Apache Camel Komponente für Teamcenter

Wie schon im letzten Kapitel angedeutet, ist aktuell keine APACHE CAMEL-Komponente für TEAMCENTER verfügbar. Da ohne eine solche Komponente keine Verbindung zwischen TEAMCENTER und APACHE CAMEL aufgebaut werden kann, muss diese entwickelt werden.

Die Grundidee der Teamcenterkomponente ist das verpacken der TEAMCENTER Service API in eine API, die komfortabel in Routen verwendet werden kann. Sobald eine Komponente für Teamcenter existiert, kann ein Teamcenter-Endpunkt in einer Camel Route definiert werden. Wird diese Route aufgerufen, erzeugt der Endpunkt einen entsprechenden Producer. Der Producer prüft alle nötigen Eingabeinformationen aus dem Camel Exchange und führt den gewünschten Service in TEAMCENTER aus. Das Ergebnis wird schlussendlich wieder in den Exchange geschrieben und in der Route weitergeleitet. Die Teamcenterkomponente stellt somit eine Art Fassade für die Teamcenterservices dar, vergleichbar mit dem gleichnamigen Designmuster in Gamma et al. [2004]. Zusätzlich muss die Komponete sich um den Verbindungsaufbau und Authentifizierung zu Teamcenter kümmern.

Bevor mit der Entwicklung der Komponente begonnen wird, werden die benötigten Funktionen beschrieben.

5.2.1. Anforderungen

- *Erzeugen und Verwalten von Verbindungen zu Teamcenter* - TEAMCENTER ist ein zugangsbeschränktes System, bevor eine Interaktion durchgeführt werden kann, muss sich jeder Benutzer mit Name und Passwort authentifizieren. Jeder Benutzer hat in TEAMCENTER spezielle Berechtigungen, unterschiedliche Aufgaben benötigen spezielle Berechtigungen. Aus diesem Grund muss eine Möglichkeit gebo-

ten werden, mit der Benutzerinformationen über die Teamcenterkomponente an TEAMCENTER übergeben werden können. Aus Sicherheits- und Performancegründen schließt TEAMCENTER inaktive Verbindungen nach einer bestimmten Zeit automatisch. Die Komponente muss die Verbindungsdaten zwischenspeichern und im Falle einer geschlossenen Verbindung bei Bedarf automatisch eine Neue erstellen.

- *Aufrufen von Teamcenterservices* - Teamcenter bietet eine breite Palette an Services. Viele der Services dienen speziellen Aufgaben und sind nur in ausgewählten Anwendungsfällen relevant. Für die Durchführung der grundlegenden Integrationsaufgaben wird nur ein ausgewählter Teil der angebotenen Teamcenter-Services benötigt. Die folgenden Funktionen werden typischerweise benötigt und sollen von der Teamcenterkomponenten unterstützt werden.
  - *Abfragen/Erstellen von Produktinformationen in Teamcenter*: Die Komponente muss Möglichkeiten bieten, Informationen zu Objekten aus Teamcenter abzufragen, zu ändern oder neu zu erstellen. Beim Abfragen von Informationen aus Teamcenter muss definiert werden können, welche Informationen zu einem bestimmten Objekt abgefragt werden sollen.
  - *Revisionieren von Produkten*: Das Revisionieren ist ein grundlegender Mechanismus zur Verwaltung unterschiedlicher Entwicklungsstände in Teamcenter. Dieser Mechanismus muss auch von der Teamcenterkomponente unterstützt werden.
  - *Abfragen/Erstellen von Stücklisteninformationen in Teamcenter*: Die Komponente muss Möglichkeiten bieten, Stücklisteninformationen aus Teamcenter abzufragen, zu ändern oder neu zu erstellen. Teamcenter bietet sehr umfangreiche Funktionen zur Verwaltung von Stücklisten, die Implementierung all dieser Funktionen würde den Umfang der Arbeit übersteigen. Aus diesem Grund sollen nur grundlegende Stücklistenfunktionen implementiert werden. Dazu gehören das Abfragen vom Stücklisten, das Erstellen von neuen Stücklistenobjekte, das Ändern bestehender Strukturen und das Ändern von Informationen zu einzelnen Stücklistenzeilen.
  - *Abfragen/Erstellen von Dateien in Teamcenter*: Von Teamcenter verwaltete Dateien sollen aus Teamcenter geladen werden können und neue Dateien sollen nach Teamcenter importiert werden können. Teamcenter bietet die Möglichkeit mehrere Versionen von einer Datei zu verwalten. Werden Änderungen an einer Datei vorgenommen muss die Komponente eine neue Version anlegen.
  - *Reservieren von Objekten in Teamcenter*: Um den gleichzeitigen Zugriff mehrerer Benutzer auf den gleichen Datensatz zu verhindern, gibt es in Teamcenter die Möglichkeit Datensätze exklusiv zu sperren. Die Teamcenter-Komponente muss Möglichkeiten bieten Datensätze zu sperren und die Sperre wieder aufzuheben.

- *Fehlerhandling*: Alle Fehler, die bei der Durchführung eines Teamcenter Services auftreten, müssen von der Komponente gemeldet werden.

5.2.2. Erstellung einer Projektstruktur

Der einfachste Weg mit der Entwicklung einer neue Apache Camel Komponente zu starten, ist mit Maven[13] automatisch eine Projektstruktur anlegen zu lassen. Es gibt einen Maven-Archetype für Camel-Komponenten, welcher die nötige Projektstruktur erstellen kann. Ist Maven eingerichtet und auf der Kommandozeile verfügbar, kann mit folgendem Befehl ein neues Projekt angelegt werden.

```
mvn archetype:create -DarchetypeGroupId=org.apache.camel.archetypes -
    DarchetypeArtifactId=camel-archetype-component -DarchetypeVersion=2.5.0
     -DgroupId=teamcenter.component -DartifactId=MyComponent
```

Auf der folgenden Abbildung ist die erzeugte Projektstruktur ersichtlich. Maven legt zusätzlich eine lauffähige HelloWorld-Komponente an. Mit dem Befehl *mvn test* kann ein Komponententest durchführen werden.

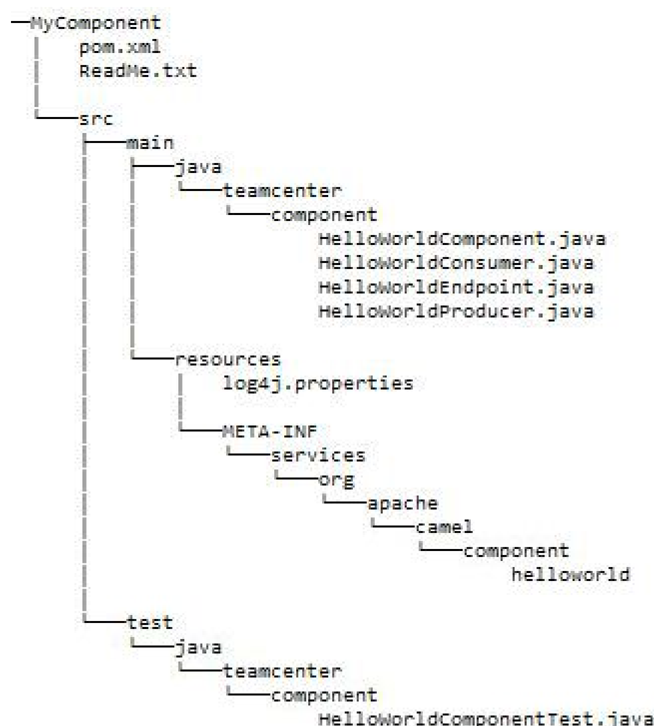

Abbildung 22: MyComponent Projektstruktur

Jede Camel-Komponente besteht aus vier grundlegenden Klassen. Erstens die *Component* Klasse, diese wird einmalig instanziiert, sobald die Komponente in einer Camel

[13] https://maven.apache.org/

Route verwendet wird. Die Component Klasse instanziiert für jedes Vorkommnis der Komponente in einer Camel Route einen entsprechenden Endpunkt aus der *Endpoint* Klasse. Außerdem gibt es noch eine Klasse für *Consumer* und *Producer*, je nach Verwendung der Komponente in einer Camel Route wird ein Consumer oder ein Producer erstellt. Camel erkennt anhand der Schlüsselwörter *from* und *to* ob ein Consumer oder Producer benötigt wird. Bei jedem Durchlauf einer Route wird von dem jeweiligen Endpunkt ein Producer bzw. ein Consumer erstellt. Soll eine Komponente ausschließlich als Consumer oder Producer verwendet werden, empfiehlt es sich trotzdem, sowohl eine Klasse für einen Producer als auch für einen Consumer anzulegen, jedoch bei der Instanziierung der nicht unterstützten Klasse eine Exception mit einer entsprechenden Fehlermeldung zu werfen.

Um die neue Komponente in Camel Routen verwenden zu können, muss noch ein eindeutiger Name zugewiesen werden. Auf der Webpräsenz von Apache Camel steht eine Liste aller offiziell verfügbaren Komponenten[14] zur Verfügung. Bei der Wahl eines Namens für die Komponente muss darauf geachtet werden, dass dieser noch nicht verwendet wird. Da die Bezeichnung *tc* nicht in Verwendung ist, wird *tc* als Name für die neue Komponente gewählt. Definiert wird der Name der Komponente durch eine Datei in der Projektstruktur. Die Datei beinhaltet den Namen der Klasse für die Komponente. Die Datei muss sich in dem Verzeichnis META−INF/services/org/apache/camel/component/tc befinden und hat in diesem Fall den folgenden Inhalt:

```
class=at.acam.camel.component.tc.TcComponent
```

Für die neue Teamcenter-Komponente wird eine neue Klasse mit dem Namen *TcComponent* erzeugt, diese erbt von der Klasse *DefaultComponent*. Damit Camel automatisch Endpunkte erzeugen kann, definiert die Überklasse eine Methode *createEndpoint()*, diese kann in der neuen Komponente überschrieben werden und so das Verhalten bei der Anlage von Endpunkten beeinflusst werden. An dieser Stelle könnten Eingabeparameter geprüft werden, außerdem bietet sie, wie später gezeigt wird, den idealen Ort für die Zuweisung oder Erstellung der Verbindung zu Teamcenter.

Das Grundgerüst für eine Camel Komponente ist jetzt vorhanden, im nächsten Kapitel wird dieses Gerüst befüllt um eine Kommunikation mit Teamcenter zu ermöglichen.

5.2.3. TcComponent

Die Hauptaufgabe der Teamcenter-Komponente ist es, einen in der Camel Route definierten Service der Teamcenter API aufzurufen und das Ergebnis an die Camel Route zurück zu geben. Für diese Aufgabe sind mehrere Informationen notwendig, die sich dynamisch von Anwendungsfall zu Anwendungsfall ändern können. Um die Informationen anwendungsfallspezifisch anzugeben, müssen diese innerhalb der Camel Route definiert

[14] http://camel.apache.org/components.html

werden. Folgende Informationen werden benötigt um einen Service in Teamcenter aufzurufen und müssen in der Route gesetzt werden können:

- Der Teamcenterhost auf dem der Service ausgeführt werden soll
- Benutzerinformationen für die Authentifizierung
- Inputinformationen für den jeweiligen Service

Aus einer Camel Route können Information entweder über die Endpunkt URI oder über den Exchange an eine Komponente übergeben werden. Die Informationen müssen von der Komponente verarbeitet werden, eine Verbindung zu Teamcenter aufgebaut werden, ein Teamcenter-Serviceaufruf durchgeführt werden und abschließend die Antwort an die Camel Route zurück übergeben werden.

Um Camel anzuweisen eine Komponente zu verwenden, muss ein entsprechender Endpunkt über eine URI in einer Route definiert werden. Die Syntax einer URI für die Teamcenter-Komponente ist folgendermaßen definiert:

```
tc:TeamcenterServerAdresse[?options]
```

Die Teamcenter Server Adresse hat folgende Syntax *hostname:port/app-name* und wird durch die Webschicht der Teamcenter Anwendung festgelegt. Diese Information wird benötigt um zu wissen, zu welcher Teamcenterinstallation eine Verbindung aufgebaut werden soll.

hostname	Netzwerkaddresse der Teamcenter Webschicht
port	Portnummer der Webschicht
app-name	Anwendungsname mit dem die Teamcenter Webschichtprozesse betrieben werden

Tabelle 2: Teamcenter Server Adresse

Um sich mit Teamcenter verbinden zu können, werden zusätzlich zur Server Adresse, Authentifizierungsinformationen in Form von Benutzername und Passwort benötigt, optional kann eine Gruppe und Rolle angegeben werden. Wird keine Gruppe oder Rolle angegeben, wird eine Verbindung mit der jeweiligen Defaultgruppe und Rolle hergestellt. Auch diese Informationen können über die URI definiert werden, sie werden als URI Parameter nach der Teamcenter Server Adresse definiert. In Tabelle 3 sind die erlaubten URI Paramater aufgelistet.

Name	Mandatory	Beschreibung
tcUser	Ja	Name für die Authentifizierung in Teamcenter
tcPass	Ja	Passwort für die Authentifizierung in Teamcenter
tcGroup	Nein	Gruppe für die Authentifizierung in Teamcenter
tcRole	Nein	Rolle für die Authentifizierung in Teamcenter
tcSessionKey	Nein	Identifier für die Speicherung und Wiederverwendung von Verbindungen zu Teamcenter
tcMethod	Nein	Servicemethode die von der Teamcenterkomponente ausgeführt werden soll

Tabelle 3: URI Parameter für TcComponent

Um den Teamcenterserver nicht unnötig zu belasten, sollten Verbindungen zu Teamcenter erst aufgebaut werden, wenn diese tatsächlich benötigt werden. Benötigt wird eine Verbindung jedes Mal, wenn eine Route mit einer Teamcenter-Komponente durchlaufen wird. Da der Verbindungsaufbau eine gewisse Zeit beansprucht, macht es Sinn bestehende Verbindungen wenn möglich wiederzuverwenden. Beim erstmaligen Aufbau einer Verbindung wird in der Enterprise-Schicht ein Teamcenterserverprozess erstellt und der Verbindung zugewiesen, je nach Verfügbarkeit der Teamcenteranwendungen kann das nur wenige Sekunden, aber auch Minuten dauern. Der Aufbau der Verbindung hat somit einen erheblichen Einfluss auf die Performance. Die Frage, wann eine Verbindung wiederverwendet werden kann, ist nicht eindeutig zu beantworten und hängt meist von der durchzuführenden Aufgabe ab. Aus diesem Grund wird die Entscheidung, Verbindungen wiederzuverwenden, dem Ersteller der Camel Route überlassen. Grundsätzlich kann eine Verbindung nur wiederverwendet werden, wenn der Benutzer und der Teamcenterserverhost gleich bleibt. Es ist möglich, dass Benutzer dynamisch in einer Route definiert werden, somit könnte sich der Benutzer bei jedem Durchlauf der Route ändern.

Um die Wiederverwendung von Verbindungen aus der Camel Route steuern zu können, wird die URI Option *tcSessionKey* definiert. Wird die Option *tcSessionKey* verwendet, wird die Verbindung gespeichert. Wird später in einer Route bei einem Teamcenterendpunkt die gleiche Kombination von Benutzer und TcSessionKey verwendet, wird auch die Verbindung wiederverwendet. Die Teamcenterkomponente benötigt somit eine Möglichkeit zur Speicherung von Verbindungen.

Wird in einer Camel Route ein bestimmter Endpunkt definiert, erstellt Camel automatisch eine Instanz der zugehörigen Komponentenklasse. Für jede Komponente wird nur eine Instanz erzeugt, diese ist für die gesamte Laufzeit der Anwendung verfügbar und erzeugt für jedes Vorkommnis in einer Route einen Endpunkt. Der Endpunkt wiederum erzeugt bei jedem Durchlauf der Route eine Instanz des Producers oder Consumers. Da es nur eine Instanz der Komponente in einem *CamelContext* gibt, diese über die gesamte Laufzeit verfügbar ist und alle Endpunkte über diese Instanz erzeugt werden, bietet die Komponente den idealen Ort für die Erstellung und Speicherung der Verbindungen.

Abbildung 23 zeigt ein vereinfachtes Klassendiagramm der Teamcenterkomponente.

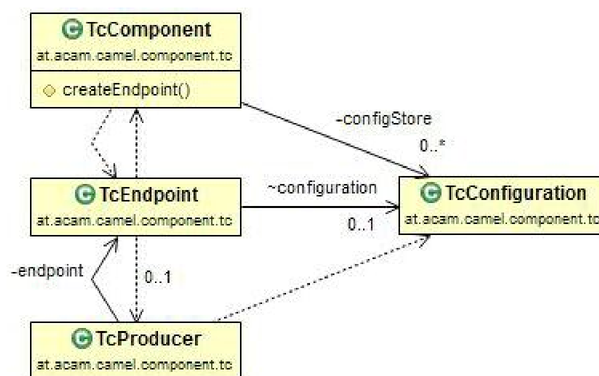

Abbildung 23: Vereinfachtes Klassendiagram der Teamcenter Komponente für Camel

Für die Erstellung einer Verbindung zu Teamcenter muss ein Objekt der Klasse com.teamcenter.soa.client.Connection instanziiert werden (siehe Zeile 14 in Listing 2), zusätzlich muss eine Implementierung des com.teamcenter.soa.client.CredentialManager Interfaces (siehe Zeile 5 in Listing 2) und des com.teamcenter.soa.client.ExceptionHandler Interfaces (siehe Zeile 20 in 2) erstellt werden und dem Connection Objekt übergeben werden. Die *CredentialManager* Implementierung kümmert sich um die Bereitstellung der Authentifizierungsinformation. Sobald eine Authentifizierungsanfrage vom Server gestellt wird, werden Methoden des *CredentialManagers* verwendet um die Authentifizierungsinformation abzufragen. Die Implementierung des *ExceptionHandler* übernimmt die Abarbeitung von Fehlern, die außerhalb der Geschäftslogik der Teamcenterservice Operationen auftreten können. Um die Erstellung einer Verbindung zu Teamcenter zu vereinfachen, wird diese in eine eigene Klasse TcConfiguration ausgelagert. Wird eine TcConfiguration erzeugt, werden alle benötigten Objekte für eine Verbindung zu Teamcenter instanziiert. Zusätzlich können dort auch Konstanten für eine genauere Spezifizierung der Verbindungsdetails gesetzt werden. Bei der Anlage einer neuen TcConfiguration müssen alle verbindungsspezifischen Informationen mit übergeben werden (siehe Zeile 1 in Listing 2). Die Instanzen von TcConfiguration werden unter einer Kombination von Benutzername und TcSessionKey gespeichert und können somit von mehreren Endpunkten verwendet werden. Die Speicherung erfolgt über eine Variable direkt in der Klasse TcComponent. Bei der Erstellung eines neuen TcEndpoint, wird von der TcComponent bei Bedarf eine neue TcConfiguration erstellt, gespeichert und dem Endpunkt zugewiesen. Existiert bereits eine geeignete TcConfigration wird diese an den Endpunkt übergeben.

5.2.4. TcEndpoint

Für jedes Vorkommnis einer URI in einer Camel Route wird von der jeweiligen Komponente ein zugehöriger Endpunkt erzeugt. Dazu wird von Camel automatisch die Methode

Listing 2: TcConfiguration - Konstruktor

```
public TcConfiguration(String host, String tcUser, String tcPass, String tcGroup, String tcRole){
    // Set discriminator so we always connect to the same server
    discriminator = UUID.randomUUID().toString();
    // The CredentialManager is used by the SOA Framework to get the user's credentials when challenged by
         the server.
    credentialManager = new TcCredentialManager();
    credentialManager.setUserPassword(tcUser, tcPass, discriminator);
    if (tcGroup!=null&&tcRole!=null){
        credentialManager.setGroupRole(tcGroup, tcRole);
    }
    String protocol=SoaConstants.HTTP;
    // only http will be supported
    host = "http://"+host;
    // Create the Connection object, no contact is made with the server until a service request is made
    connection = new Connection(host, credentialManager, SoaConstants.REST, protocol);
    logger.debug("OPT_USE_COMPRESSION - "+connection.getOption("OPT_USE_COMPRESSION"));
    logger.info("New connection for {}@{} created",tcUser,host);
    // Add an ExceptionHandler to the Connection, this will handle any
    // InternalServerException, communication errors, XML marshaling errors
    // .etc
    connection.setExceptionHandler(new TcExceptionHandler());
    // While the above ExceptionHandler is required, other Listeners are optional.
    // Add a Change and Delete Listener, this will be notified when ever a
    // a service returns model objects that have been updated or deleted.
    connection.getModelManager().addModelEventListener(new TcModelEventListener());
}
```

Listing 3: TcEndpoint.createConsumer()

```
public Consumer createConsumer(Processor processor) throws Exception {
    throw new UnsupportedOperationException(
            "Consumer are not supported for this endpoint:" + getEndpointUri());
}
```

createEndpoint() der jeweiligen Komponente aufgerufen. Der Endpunkt existiert für die gesamte Laufzeit der Anwendung. Die Hauptaufgabe des TcEndpoint ist die Erstellung von Producer oder Consumer. Bei jedem Durchlauf einer Route wird eine neue Instanz des Producers bzw. Consumer angelegt. Camel übernimmt diese Aufgabe automatisch, indem die Methode *createProducer()* bzw. *createConsumer()* in der Endpunktklasse aufgerufen wird. Der TcEndpoint unterstützt nur die Anlage von Producer. Wird trotzdem verucht einen Consumer zu erstellen, wirft die Methode *createConsumer()* eine Exception (siehe Listing 3). Welcher Producer erzeugt wird, wird anhand der URI Option *tcMethod* wird in der TcEndpoint Klasse entschieden (siehe Listing 4). Um die verschiedenen Funktionen der Teamcenterkomponente zu gruppieren und übersichtlicher zu gestalten, wurden mehrere Producer implementiert. Die Tabellen 4 und 5 beinhalten alle erlaubten Werte für den URI Option tcMethod. Zu jedem Wert wird beschrieben welche Funktionen ausgeführt werden und welcher Producer für diese Funktion zuständig ist. Die zugehörigen Inputparameter sind im Anhang in den Tabellen 7, 8, 9 und 10 aufgelistet.

tcMethod	Beschreibung
createObjects	Der TcCOProducer ermöglicht das Erzeugen von Geschäftsobjekten in Teamcenter. So können Datasets oder Items und ItemRevisionen erstellt werden. Bedingt die Erstellung eines neuen Geschäftsobjekts die Anlage weiterer abhängiger Objekte, wird das automatisch vom Service übernommen. So wird zum Beispiel bei der Anlage eines neuen Items automatisch eine ItemMasterForm und eine ItemRevision erzeugt. Alle Header die im Zusammenhang mit dem TcCOProducer stehen, beginnen mit tc.CO.
getFiles, putFiles	Der *TcFileProducer* ermöglicht den Down- oder Upload von Dateien aus/nach Teamcenter. Der *TcFileProducer* verwendet die Funktionen des *File-Client-Cache*. Voraussetzung dafür ist eine verfügbare Installation des FCC, zusätzlich muss der FCC für die Kommunikation mit dem entsprechenden Teamcenterserver konfiguriert sein. Die Teamcenter Client Services suchen das FCC Installationsverzeichnis über die Umgebungsvariable FMS_HOME. Die entsprechende Konfigurationsdatei ist unter FMS_HOME\fcc.xml zu finden. Alle Header, die im Zusammenhang mit dem *TcFileProducer* stehen, beginnen mit tc.*file*.
getItemAndRelatedObjects	Der *TcGIAROProducer* verwendet die Funktionen des GetItemAndRelatedObjects-Services, um Objekte und deren Verknüpfungen aus Teamcenter zu laden. Es können Items, ItemRevisionen, Datasets und verknüpfte Objekte geladen werden. Welche Objekte geladen werden sollen, muss über die Eingabeparameter angegeben werden. Alle Header, die im Zusammenhang mit dem *TcGIAROProducer* stehen, beginnen mit tc.*GIARO*.
loadObjects	Der *TcLOProducer* ermöglicht das Laden von beliebigen Objekten anhand deren Uid. Die Uids werden aus einem Header ausgelesen und an die Methode *TcFacade.loadObjects()* übergeben. Alle Header, die im Zusammenhang mit dem *TcLOProducer* stehen, beginnen mit tc.*LO*.
query	Der *TcQUERYProducer* ermöglicht das Suchen in der Teamcenterdatenbank. Als Eingabeparameter wird der Name der Teamcenter Suche und eine Liste von Kriterien und Werten benötigt. Alle Header, die im Zusammenhang mit dem *TcQUERYProducer* stehen, beginnen mit tc.*QUERY*.
setProperties	Der *TcSPProducer* ermöglicht das Ändern von beliebigen Objekten anhand deren Uid. Die Uids und die zu aktualisierenden Eigenschaften werden als Input benötigt. Alle Header, die im Zusammenhang mit dem *TcSPProducer* stehen, beginnen mit tc.*SP*.
createWorkflowInstance	Der *TcWFProducer* kann genutzt werden um Workflowprozess in Teamcenter zu erzeugen und auszulösen. Über Header können verschiedene Eigenschaften des Workflows festgelegt werden. Zusätzlich können Attachments für den Workflow definiert werden. Alle Header, die im Zusammenhang mit dem *TcWFProducer* stehen, beginnen mit tc.*WF*.

Tabelle 4: Erlaubte Werte für URI Option tcMethod - Teil 1

tcMethod	Beschreibung
revise	Der *TcREVProducer* bietet Funktionen zur Erzeugung von neuen Revisionen in Teamcenter. Über Header kann definiert werden von welcher ItemRevision eine neue Revision angelegt werden soll. Zusätzlich können auch Informationen bezüglich *Deep Copy Rules* definiert werden. Alle Header, die im Zusammenhang mit dem *TcREVProducer* stehen, beginnen mit tc.*REV*.
processBOM	Der *TcBOMProducer* ermöglicht Operationen im Zusammenhang mit Stücklisten (Bill of Material oder kurz BOM). Für die Ansicht von Stücklisten müssen diese zuerst in einen fiktiven Arbeitsbereich geladen werden. Dieser Bereich wird als Stücklistenfenster bezeichnet und definiert in welcher Konfiguration eine Stückliste dargestellt wird. Anschließend können Funktionen für die Erweiterung von Stücklisten ausgeführt werden. Um ein Stücklistenfenster von dem TcBOMProducer erstellen zu lassen, wird der Header *tc.BOM.window.create=true* verwendet. Zusätzlich muss angegeben werden welche Stückliste geöffnet werden soll. Über die Header *tc.BOM.window.bomview*, *tc.BOM.window.item* oder *tc.BOM.window.itemrev* kann die Uid des entsprechenden Objekts angegeben werden. Optional kann zusätzlich die Uid einer Revisionsregel im Header *tc.BOM.window.revrule* angegeben werden. Das Stücklistenfenster wird automatisch am Ende des Services geschlossen. Soll es für spätere Operationen geöffnet bleiben, kann das mit dem Header *tc.BOM.window.close.after=false* definiert werden. In dem geöffneten Stücklistenfester können Funktionen, wie zum Beispiel die Erweiterung einer oder aller Ebenen der Stückliste, durchgeführt werden. Diese Operation kann mit dem Header *tc.BOM.expand* angestoßen werden. Erlaubte Werte sind „one“ oder „all“. Soll nicht die oberste Stücklistenzeile erweitert werden, muss eine spezielle Stücklistenzeile mit ihrer Uid im Header *tc.BOM.expand.parent* angegeben werden. Wird ein Stücklistenfenster nicht mehr benötigt, sollte es geschlossen werden. Das kann explizit durch den Header *tc.BOM.expand.close* mit der Uid des Stücklistenfensters durchgeführt werden. Alle Header, die im Zusammenhang mit dem *TcBOMProducer* stehen, beginnen mit tc.*BOM*.
createOrUpdateRelative Structure	Der TcBOMProducer ermöglicht auch die Erstellung und Bearbeitung von Stücklisten. Im Gegensatz zur Erweiterung von Stücklisten wird für die Erstellung einer Stückliste kein Stücklistenfenster benötigt. Über die Header *tc.BOM.COURS.parent* und *tc.BOM.COURS.childs* können die jeweiligen Eltern- und Kind-Elemente der Stückliste über deren Uids definiert werden. Zusätzlich kann über weitere Header die Manipulation und Anlage der Stücklisten genauer gesteuert werden. Für eine vollständige Liste der unterstützten Funktionen wird auf den Anhang verwiesen. Zu beachten ist, dass bestehende Stücklisten nur geändert werden können, wenn das zugehörige *BOMView*-Objekt zuvor ausgecheckt wurde, ansonsten wird ein entsprechender Fehler in der *ServiceData* Struktur der Ergebnisses zurückgemeldet. Alle Header, die im Zusammenhang mit dem *TcBOMProducer* stehen, beginnen mit tc.*BOM*.

Tabelle 5: Erlaubte Werte für URI Option tcMethod - Teil 2

Listing 4: TcEndpoint.createProducer()

```
1  public Producer createProducer() throws Exception {
2      if (tcMethod!=null)
3      {
4          switch (tcMethod)
5          {
6              case ("processBom"):
7                  Logger.info("Start TcBOMProducer");
8                  return new TcBOMProducer(this, configuration);
9
10             case ("createOrUpdateRelativeStructure"):
11                 Logger.info("Start TcBOMProducer");
12                 return new TcBOMProducer(this, configuration);
13
14             case ("checkin"):
15                 Logger.info("Start TcProducer");
16                 return new TcProducer(this, configuration);
17
18             case ("checkout"):
19                 Logger.info("Start TcProducer");
20                 return new TcProducer(this, configuration);
21
22             case ("createWorkflowInstance"):
23                 Logger.info("Start TcWFProducer");
24                 return new TcWFProducer(this, configuration);
25
26             case ("createObjects"):
27                 Logger.info("Start TcCOProducer");
28                 return new TcCOProducer(this, configuration);
29
30             case ("getFiles"):
31                 Logger.info("Start TcFileProducer");
32                 return new TcFileProducer(this, configuration);
33
34             case ("putFiles"):
35                 Logger.info("Start TcFileProducer");
36                 return new TcFileProducer(this, configuration);
37
38             case ("getItemAndRelatedObjects"):
39                 Logger.info("Start TcGIAROProducer");
40                 return new TcGIAROProducer(this, configuration);
41
42             case ("loadObjects"):
43                 Logger.info("Start TcLOProducer");
44                 return new TcLOProducer(this, configuration);
45
46             case ("query"):
47                 Logger.info("Start TcQueryProducer");
48                 return new TcQUERYProducer(this, configuration);
49
50             case ("revise"):
51                 Logger.info("Start TcREVProducer");
52                 return new TcREVProducer(this, configuration);
53
54             case ("setProperties"):
55                 Logger.info("Start TcSPProducer");
56                 return new TcSPProducer(this, configuration);
57
58             default :
59                 throw new Exception("Method "+tcMethod+" is not supported");
60         }
61     }
62     else
63     {
64         Logger.info("Start TcProducer");
65         return new TcProducer(this, configuration);
66     }
67 }
```

5.2.5. TcProducer

Die Producer Klasse wird verwendet um die Eingabeinformationen aus den Camel Exchange zu sammeln, diese zu prüfen und in eine geeignete Inputstruktur für den jeweiligen Serviceaufruf zu übersetzten, den benötigten Teamcenter Service aufzurufen und die Ergebnisse an die Camel Route zurück zu geben. Um die Implementierung des Producers übersichtlicher zu gestalten, wurden mehrere Producer erstellt und die Funktionen aufgeteilt. Das vereinfachte Klassendiagramm in Abbildung 24 zeigt die verfügbaren Tc-Producer. Es gibt eine Superklasse TcProducer, diese enthält grundlegende Funktionen die bei jedem Serviceaufruf benötigt werden. Dazu gehören *Policy-Services*, *Metamodel-Services* und *Reservation-Services*. Jeder weitere Producer ist von dem TcProducer abgeleitet und ist für spezielle Services aus einem funktionellen Bereich verantwortlich.

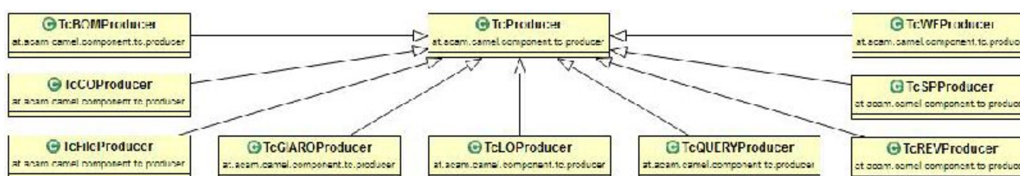

Abbildung 24: Auszug aus Klassendiagramm für TcProducer

Welcher *TcProducer* erzeugt wird, wird im Teamcenter-Endpunkt entschieden. Der erzeugte Producer durchsucht den eingehenden Exchange nach den benötigten Inputparametern die für die jeweiligen Teamcenter Service benötigt werden. Die Tabelle 7, 8, 9 und 10 im Anhang beinhalten die erlaubten bzw. benötigten Inputparameter. Zusätzlich ermöglichen alle Producer die Durchführung der in Tabelle 6 beschriebenen Funktionen. Der *TcProducer* kann auch ohne den URI Parameter tcMethod verwendet werden. In diesem Fall sind jedoch nur die Inputparameter aus Tabelle 6 erlaubt.

Input Header		Output Header		Beschreibung
tc.policy.name	String	tc.policy.name. result	boolean	Setzten einer Property Policy anhand des Namen
tc.policy. typenames	String	tc.policy. typenames.result	boolean	Setzten einer neuen Policy für den angegebenen Typ mit allen Eigenschaften
tc.meta.types	String[]	tc.meta.types. result	List< com.teamcenter.soa. client.model.Type>	Abfragen von Metamodel Informationen für eine Menge von Typenamen
tc.meta.types. createinput	String[]	tc.meta.types. createinput.result		Abfragen der nötigen Eingabeinformationen für die Anlage eines Objekts von einem bestimmten Typ
tc.meta.types. saveasinput	String[]	tc.meta.types. saveasinput. result		Abfragen der nötigen Eingabeinformationen für das „Speichern Unter“ eines Objekts von einem bestimmten Typ
tc.RES.ci.uids	String[]	tc.RES.ci.result	com.teamcenter.soa. client.model. ServiceData	Freigeben von exklusiv zur Bearbeitung gesperrten Objekten
tc.RES.co.uids	String[]	tc.RES.co.result		Exklusives reservieren von Objekten für die Bearbeitung durch einen Benutzer
tc.RES.co. comment	String	-	-	Checkout Kommentar, nur in Zusammenhang mit tc.RES.co.uids relevant
tc.RES.co. changeid	String	-	-	Checkout ChangeId, nur in Zusammenhang mit tc.RES.co.uids relevant

Tabelle 6: Allgemeine Optionen für TcComponent

5.2.6. TcFacade

> Die Facadenklasse definiert eine abstrakte Schnittstelle, welche die Benutzung des Subsystems vereinfacht (vgl. Seite 212 in Gamma et al. [2004])

Die TcFacade ist eine Implementierung des Teamcenterservices und stellt ausgewählte Methoden der Teamcenter API für die Teamcenterkomponente zur Verfügung. Die Hauptaufgabe der *TcFacade* ist die Durchführung der Teamcenterservice-Aufrufe und die Rückgabe der Serviceantwort.

Alle Methoden der *TcFacade* sind statisch und benötigen als Übergabeparameter ein Teamcenter Connection-Objekt (*com.teamcenter.soa.client.Connection*) und optional die jeweilige Inputstruktur. Jede Methode der *TcFacade* ist so wie der zugehörige Service der Teamcenter API benannt. Die Teamcenter API bietet eine Vielzahl von Services für unterschiedliche Aufgaben an. Die meisten Services führen sehr spezielle Aufgaben aus und machen nur in bestimmten Anwendungsfällen Sinn. Um eine gewisse Funktionalität in der Teamcenterkomponente anbieten zu können, die Komponente jedoch nicht mit überflüssigen Services zu überladen, gilt es, eine gezielte Menge von Services zu implementieren.

Folgende Methoden wurden implementiert und werden im Anschluss kurz beschrieben. Grundsätzlich wird in der *TcFacade* kein Fehlerhandling durchgeführt. Fehler werden von dem jeweiligen Service in der Antwort inkludiert und müssen später durch parsen der Antwort erkannt und entsprechend abgehandelt werden.

getItemAndRelatedObjects Mit Hilfe der *getItemAndRelatedObjects*-Methode können gezielt Objekte aus Teamcenter geladen werden. Input der Methode ist ein Connection-Objekt und eine Inputstruktur com.teamcenter.services.loose.core._2008_06. DataManagement.GetItemAndRelatedObjectsInfo. Das Connection-Objekt wird verwendet um einen Service-Stub zu erzeugen (siehe Zeile 2 in Listing 5). Ein Service-Stub wird verwendet um den Teamcenterserviceaufruf durchzuführen (siehe Zeile 5 und Zeile 15). Die Inputstruktur repräsentiert eine Kombination von Item-Ids, Revision-Ids, Uids, Datasettypen und Relationen und definiert welche Objekte gesucht werden und welche Verknüpfungen zu diesen Objekten gesucht sind. Der Teamcenterservice-Aufruf in Zeile 5 Listing 5, ruft den Service com.teamcenter.services.loose.core._2008_06.DataManagement .getItemAndRelatedObjects auf.

Ist der Teamcenter-*ClientModelCache* aktiviert (siehe Zeile 8 Listing 5), müssen alle Objekte im Ergebnis aktualisiert werden, nur dann ist sicher gestellt, dass eine aktuelle Repräsentation des Objektes aus Teamcenter zurückgegeben wird. Der Grund für dieses Verhalten ist im Model-Cache des Teamcenterclients zu finden. Dieser speichert alle Objekte, die über Serviceaufrufe aus Teamcenter geladen wurden. Wird ein Objekt später erneut aus Teamcenter geladen, wird es mit dem bereits vorhanden

Objekt im Cache zusammengeführt und das Objekt im Cache wird mit den neu geladenen Eigenschaften aktualisiert. Eigenschaften die im zweiten Serviceaufruf jedoch nicht zurückgegeben werden, bleiben unverändert und können veraltete Informationen enthalten. Das *ServiceData*-Objekt, beinhaltet eine Liste aller Objekte aus dem Serviceaufruf. Diese Objekte werden mit Hilfe des Teamcenterservices com.teamcenter.services.loose.core._2007_01.DataManagement.refreshObjects aktualisiert (siehe Zeile 15 Listing 5). Zuletzt wird das Ergebnis des *getItemAndRelatedObjects*-Services zurückgegeben.

Listing 5: TcFacade.getItemAndRelatedObjects

```
1  public static GetItemAndRelatedObjectsResponse getItemAndRelatedObjects(Connection connection,
       GetItemAndRelatedObjectsInfo[] input)
2  {
3          DataManagementService dmService = DataManagementService.getService(connection);
4          log.debug("Send ServiceRequest for GetItemAndRelatedObjectsResponse");
5          GetItemAndRelatedObjectsResponse response = dmService.getItemAndRelatedObjects(input);
6          log.debug("{} Plain objects returned by getItemAndRelatedObjects Service Call",response.serviceData.
               sizeOfPlainObjects());
7          //Refresh the model object in the response to ensure the object is up to date
8          if (connection.getOption("OPT_CACHE_MODEL_OBJECTS")=="true"){
9                  log.debug("OPT_CACHE_MODEL_OBJECTS=true - refresh objects");
10                 if(response.serviceData.sizeOfPlainObjects()>0){
11                         ModelObject [] pObjects = new ModelObject[response.serviceData.sizeOfPlainObjects()];
12                         for (int i=0; i<response.serviceData.sizeOfPlainObjects(); i++){
13                                         pObjects[i] = response.serviceData.getPlainObject(i);
14                                 }
15                                 dmService.refreshObjects(pObjects);
16                         }
17                 }
18                 return response;
19 }
```

getProperties Mit dieser Methode können Eigenschaften zu einem *ModelObject* explizit aus TEAMCENTER geladen werden. Besonders nützlich ist das, wenn spezielle Eigenschaften benötigt werden die nicht in der aktuellen *ObjectPropertyPolicy* definiert werden.

Input der Methode ist ein *Connection*-Objekt, ein Array von *ModelObject* mit den Objekten zu denen weitere Eigenschaften geladen werden sollen und ein Array von Strings mit den gewünschten Eigenschaften.

In Zeile 3 in Listing 6 wird mit dem *Connection* Objekt ein Service-Stub erzeugt, mit dem Service-Stub wird im Anschluss der com.teamcenter.services.loose.core._2006_03.DataManagement.getProperties Service ausgeführt. Der Rückgabewert ist eine com.teamcenter.soa.client.model.ServiceData Struktur.

Listing 6: TcFacade.getProperties

```
1  public static ServiceData getProperties(Connection connection, ModelObject[] objects, String[] attributes )
2          {
3                  DataManagementService dmService = DataManagementService.getService(connection);
4                  return dmService.getProperties(objects, attributes);
5          }
```

setProperties Mit dieser Methode können Eigenschaften von Objekten in TEAMCENTER geändert werden. Input der Methode ist ein *Connection*-Objekt und eine Inputstruk-

tur vom Typ com.teamcenter.services.loose.core._2010_09.DataManagement.PropInfo . Die Inputstruktur beinhaltet eine Liste von Uids der zu ändernden Objekte, eine Liste der zu ändernden Eigenschaften und die neuen Eigenschaftswerte. Zuerst wird mit Hilfe der *Connection* ein Service-Stub erzeugt (siehe Zeile 2 in Listing 7). Mit diesem Service-Stub wird anschließend der Serviceaufruf com.teamcenter.services.loose.core._2010_09 .DataManagement.setProperties durchgeführt (siehe Zeile 4 in Listing 7). Die Antwort der Serviceaufrufs wird abschließend zurückgegeben.

Listing 7: TcFacade.setProperties

```
1 public static SetPropertyResponse setProperties(Connection connection, PropInfo[] propInfo, String[] options
      ){
2         DataManagementService dmService = DataManagementService.getService(connection);
3         log.debug("Send ServiceRequest for setProperties");
4         return dmService.setProperties(propInfo, options);
5 }
```

loadObjects Die Methode ermöglicht das Abfragen von Elementen deren Uid bekannt ist. Der Service ist einfacher aufgebaut als der *GetItemAndRelatedObjects*-Service, liefert jedoch nur das gefragte Objekt ohne dessen Verknüpfungen. Auch bei diesem Service ist es nötig alle Objekte, die bereits im Client-Model-Cache geladen wurden, zu aktualisieren (siehe 5.2.6). Da jedoch im Vergleich zu *GetItemsAndRelatedObjects* die Uids aller gesuchten Objekte bekannt sind, kann vorab geprüft werden, welche Objekte sich bereits im Model-Cache befinden (siehe Zeile 5 in Listing 8). Nur diese Objekte werden in Zeile 12 aktualisiert. Für die Durchführung des Serviceaufrufs in TEAMCENTER, wird wieder mit dem *Connection*-Objekt ein Service-Stub erzeugt (siehe Zeile 9 in Listing 8). Abschließend wird der com.teamcenter.services.loose.core._2007_09.DataManagement .loadObjects Service aufgerufen und dessen Antwort zurückgegeben.

Listing 8: TcFacade.loadObjects

```
1  public static ServiceData loadObjects(Connection connection, String [] uids){
2          //objects that are already existing in the client data model, needs to be refreshed
3          List<ModelObject> refreshList = new ArrayList<ModelObject>();
4          for (String uid : uids){
5                  if(connection.getClientDataModel().containsObject(uid)){
6                          refreshList.add(connection.getClientDataModel().getObject(uid));
7                  }
8          }
9          DataManagementService dmService = DataManagementService.getService(connection);
10         if (refreshList.size()>0){
11                 ModelObject[] mObjects = new ModelObject[refreshList.size()];
12                 dmService.refreshObjects(refreshList.toArray(mObjects));
13         }
14         return dmService.loadObjects(uids);
15 }
```

setPolicy Die *setPolicy*-Methode setzt für eine angegebene *Connection* eine *PropertyPolicy*. Die *PropertyPolicy* definiert, welche Eigenschaften für welche Objekte in der Antwort eines Serviceaufrufes von Teamcenter mit zurückgegeben werden (siehe 3.7.6). Zuerst wird in Zeile 2 in Listing 9 mit der *Connection* ein Session-Service-Stub erzeugt.

Anschließend wird in Zeile 4 mit dem Service-Stub der com.teamcenter.services.loose.core._2007_01.Session.setObjectPropertyPolicy Service aufgerufen und versucht die gewünschte *Policy* zu setzten. Definiert wird die *Policy* über den policyName, dieser muss einer gleichnamigen Policy-Datei im *TcData*-Verzeichnis des jeweiligen Teamcenterserver entsprechen. Tritt ein Fehler auf wird eine com.teamcenter.schemas.soa._2006_03.exceptions.ServiceException geworfen.

Listing 9: TcFacade.setPolicy

```
1 public static boolean setPolicy ( Connection connection , String policyName){
2         SessionService sessionService = SessionService.getService(connection);
3         try {
4                 return sessionService.setObjectPropertyPolicy(policyName);
5         } catch (ServiceException e) {
6                 e.printStackTrace();
7         }
8         return false;
9 }
```

createObjects Die *createObjects*-Methode erzeugt Objekte anhand der übergebenen Eingabestruktur vom Typ com.teamcenter.services.loose.core._2008_06.DataManagement.CreateIn. Diese muss alle benötigten Eingaben enthalten, ansonsten meldet der com.teamcenter.services.loose.core._2008_06.DataManagement.createObjects Service einen Fehler. Die Antwort des Serviceaufrufs wird abschließend zurückgegeben (siehe Zeile 3 in Listing 10).

Listing 10: TcFacade.createObjects

```
1 public static CreateResponse createObjects(Connection connection , CreateIn [] input) throws ServiceException
      {
2         DataManagementService dmService = DataManagementService.getService(connection);
3         return dmService.createObjects(input);
4 }
```

createInstance Die *createInstance*-Methode erzeugt Workflowprozesse in TEAMCENTER. Mit Hilfe der *Connection* wird ein com.teamcenter.services.loose.workflow.WorkflowService Stub erstellt, mit diesem Service-Stub wird der com.teamcenter.services.loose.workflow._2008_06.Workflow.createInstance Service ausgeführt (siehe Zeile 3 in Listing 11). Das Ergebnis ist eine com.teamcenter.services.loose.workflow._2008_06.Workflow.InstanceInfo Struktur.

Listing 11: TcFacade.createInstance

```
1 public static InstanceInfo createInstance(Connection connection , boolean startImmediately , String observerKey
      , String name, String subject , String description , ContextData contextData) {
2         WorkflowService wService = WorkflowService.getService(connection);
3         return wService.createInstance(startImmediately , observerKey , name, subject , description , contextData
                );
4     }
```

revise Mit dem com.teamcenter.services.loose.core._2008_06.DataManagement.revise2 Service können neue Revisionen in TEAMCENTER angelegt werden. Die com.teamcenter.services.loose.core._2008_06.DataManagement.ReviseInfo Struktur beinhaltet alle benötigten Informationen für die Anlage der neuen Revision. Der Rückgabewert ist vom Typ com.teamcenter.services.loose.core._2008_06.DataManagement.ReviseResponse2 . Etwaige Fehler werden als Teil des Antwort zurückgeliefert.

Für die Durchführung des *revise2* Services wird ein com.teamcenter.services.loose.core .DataManagementService Stub benötigt (siehe Zeile 4 in Listing 12).

Listing 12: TcFacade.revise

```
1 public static ReviseResponse2 revise(Connection connection, ReviseInfo[] info)
2     {
3         DataManagementService dmService = DataManagementService.getService(connection);
4         return dmService.revise2(info);
5     }
```

createOrUpdateRelativeStructure Der com.teamcenter.services.loose.cad._2009_04 .StructureManagement.createOrUpdateRelativeStructure Service kann mit Hilfe eines com.teamcenter.services.loose.cad.StructureManagementService Stubs ausgeführt werden (siehe Zeile 4 in 13) und ermöglicht die Anlage und Manipulation von Stücklisten in Teamcenter. Ergebnis des Services ist eine com.teamcenter.services.loose.cad._2009_04 .StructureManagement.CreateOrUpdateRelativeStructureResponse.

Listing 13: TcFacade.createOrUpdateRelativeStructure

```
1     public static CreateOrUpdateRelativeStructureResponse createOrUpdateRelativeStructure(Connection
          connection, CreateOrUpdateRelativeStructureInfo[] inputs, CreateOrUpdateRelativeStructurePref pref)
2     {
3         StructureManagementService smService = StructureManagementService.getService(connection);
4         return smService.createOrUpdateRelativeStructure(inputs, pref);
5     }
```

createBOMWindows Mit Hilfe des com.teamcenter.services.loose.cad._2007_01.Structure Management.createBOMWindowsService kann ein neues Stücklistenfenster geöffnet werden. Für die Ausführung des Services (siehe Zeile 4 in Listing 14), wird ein com.teamcenter .services.loose.cad.StructureManagementService Stub benötigt. Das Ergebnis ist eine com.teamcenter.services.loose.cad._2007_01.StructureManagement.CreateBOMWindows Response.

Listing 14: TcFacade.createBOMWindows

```
1     public static CreateBOMWindowsResponse createBOMWindows(Connection connection, CreateBOMWindowsInfo[]
          info)
2     {
3         StructureManagementService smService = StructureManagementService.getService(connection);
4         return smService.createBOMWindows(info);
5     }
```

closeBOMWindows Geöffnete Stücklistenfenster können mit dem Service com.teamcenter .services.loose.cad._2007_01.StructureManagement.closeBOMWindows wieder geschlossen werden. Für die Ausführung (siehe Zeile 4 in Listing 15) wird ein com.teamcenter .services.loose.cad.StructureManagementService Stub benötigt. Das Ergebnis des Aufrufs ist eine com.teamcenter.services.loose.cad._2007_01.StructureManagement.Close BOMWindowsResponse Struktur.

Listing 15: TcFacade.closeBOMWindows

```
1   public static CloseBOMWindowsResponse closeBOMWindows(Connection connection, ModelObject[] bomWindows)
2   {
3       StructureManagementService smService = StructureManagementService.getService(connection);
4       return smService.closeBOMWindows(bomWindows);
5   }
```

expandPSAllLevels In einem geöffnetem Stücklistenfenster können mit Hilfe des com. teamcenter.services.loose.cad._2008_06.StructureManagement.expandPSAllLevels Service können alle Zeilen einer Stückliste erweitert werden. Benötigt wird dieser Aufruf wenn der Inhalt einer Stückliste gefragt ist. Für die Ausführung (siehe Zeile 4 in Listing 16) wird ein com.teamcenter.services.loose.cad.StructureManagementService Stub benötigt. Die Rückgabestruktur vom Typ com.teamcenter.services.loose.cad._2008_06. StructureManagement.ExpandPSAllLevelsResponse2 beinhaltet anschließend die gesamte Struktur unterhalb der expandierten Stücklistezeile.

Listing 16: TcFacade.expandPSAllLevels

```
1   public static ExpandPSAllLevelsResponse2 expandPSAllLevels(Connection connection, ExpandPSAllLevelsInfo
        info, ExpandPSAllLevelsPref pref)
2   {
3       StructureManagementService smService = StructureManagementService.getService(connection);
4       return smService.expandPSAllLevels(info, pref);
5   }
```

expandPSOneLevel In einem geöffnetem Stücklistenfenster kann mit Hilfe des com. teamcenter.services.loose.cad._2008_06.StructureManagement.expandPSOneLevel Servics eine Zeile einer Stückliste erweitert werden. Für die Ausführung (siehe Zeile 4 in Listing 17) wird ein com.teamcenter.services.loose.cad.StructureManagementService Stub benötigt. Die Rückgabestruktur ist vom Typ com.teamcenter.services.loose.cad. _2008_06.StructureManagement.ExpandPSOneLevelResponse2 und beinhaltet die Struktur für ein Level unterhalb der angegebenen Stücklistezeile.

Listing 17: TcFacade.expandPSOneLevel

```
1   public static ExpandPSOneLevelResponse2 expandPSOneLevel(Connection connection, ExpandPSOneLevelInfo info,
        ExpandPSOneLevelPref pref)
2   {
3       StructureManagementService smService = StructureManagementService.getService(connection);
4       return smService.expandPSOneLevel(info, pref);
5   }
```

getFiles Der com.teamcenter.soa.client.FileManagementUtility.getFiles Service ermöglicht den Download von Dateien aus Teamcenter durch den Einsatz des File-Client-Cache. Voraussetzung ist eine konfigurierte Installation des FCC. Der *getFiles*-Service lädt Dateien in den FCC, für eine weitere Bearbeitung müssen die in einem unabhängigen Schritt aus dem Cache kopiert werden. Für die Ausführung wird ein com.teamcenter.soa.client.FileManagementUtility Stub verwendet (siehe Zeile 4 in Listing 18). Die Rückgabe Struktur ist vom Typ com.teamcenter.soa.client.GetFileResponse und beinhaltet die Informationen über den Ort der heruntergeladenen Dateien.

Listing 18: TcFacade.getFiles

```
1   public static GetFileResponse getFiles(Connection connection, ModelObject[] imanFiles)
2   {
3       FileManagementUtility fileUtil = new FileManagementUtility(connection);
4       return fileUtil.getFiles(imanFiles);
5   }
```

getFileToLocation Der *getFileToLocation* Service arbeitet ähnlich wie der *getFile* Service. Jedoch kann zusätzlich ein Zielort für die zu ladenden Dateien angegeben werden. Für die Durchführung des Service muss zusätzlich zum Service-Stub eine Implementierung des com.teamcenter.fms.clientcache.proxy.IFileCacheProxyCB-Interfaces angegeben werden (siehe Zeile 4 in Listing 19).

Die Rückgabe Struktur ist vom Typ com.teamcenter.soa.client.GetFileResponse und beinhaltet die Informationen über den Ort der heruntergeladenen Dateien.

Listing 19: TcFacade.getFileToLocation

```
1   public static GetFileResponse getFileToLocation(Connection connection, ModelObject imanFile, String
        targetDir, String targetName) throws NotLoadedException
2   {
3       FileManagementUtility fileUtil = new FileManagementUtility(connection);
4       return fileUtil.getFileToLocation(imanFile, targetDir , new TcFileCacheProxyCB(), targetName);
5   }
```

putFiles Der com.teamcenter.soa.client.FileManagementUtility.putFiles Service ermöglicht das Hochladen von Dateien nach Teamcenter. Für die Ausführung (siehe Zeile 4 in Listing 20) wird ein com.teamcenter.soa.client.FileManagementUtility Stub benötigt. Ergebnis ist eine com.teamcenter.soa.client.model.ServiceData Struktur.

Listing 20: TcFacade.putFiles

```
1   public static ServiceData putFiles(Connection connection, GetDatasetWriteTicketsInputData[] inputs)
2   {
3       FileManagementUtility fileUtil = new FileManagementUtility(connection);
4       return fileUtil.putFiles(inputs);
5   }
```

executeSavedQueries Der com.teamcenter.services.loose.query._2008_06.SavedQuery.executeSavedQueries Service ermöglicht die Ausführung von Suchen in der Teamcenterdatenbank, auf diese Weise können Objekte nach gewissen Kriterien in der Datenbank

gesucht werden. Für den Aufruf (siehe Zeile 38 in Listing 21) wird ein com.teamcenter.services.loose.query.SavedQueryService Stub benötigt. Ergebnis ist eine com.teamcenter.services.loose.query._2007_09.SavedQuery.SavedQueriesResponse Struktur.

Listing 21: TcFacade.executeSavedQueries

```
1  public static SavedQueriesResponse executeSavedQueries(Connection connection, String queryName, QueryInput[]
       savedQueryInput) throws NotLoadedException, ServiceException
2  {
3      ObjectPropertyPolicy pol = new ObjectPropertyPolicy();
4      pol.addType("ImanQuery", new String[] {"query_clauses", "query_class","query_name","query_flag","
           query_desc"});
5      pol.addType("WorkspaceObject", new String[] {"object_name"});
6
7      // set policy for ImanQuery Object
8      SessionService sessionService = SessionService.getService(connection);
9      sessionService.setObjectPropertyPolicy(pol);
10
11     SavedQueryService qService = SavedQueryService.getService(connection);
12     GetSavedQueriesResponse savedQueries = null;
13
14     // load available query
15     savedQueries = qService.getSavedQueries();
16
17     ModelObject query = null;
18
19     if (savedQueries.queries.length == 0)
20     {
21         log.error("There are no saved queries in the system.");
22         return null;
23     }
24
25     // Find one specific
26     for (int i = 0; i < savedQueries.queries.length; i++)
27     {
28         log.trace(savedQueries.queries[i].name);
29         if (savedQueries.queries[i].name.equals(queryName))
30         {
31             query = savedQueries.queries[i].query;
32         }
33     }
34
35     if (query!=null)
36     {
37         savedQueryInput[0].query = query;
38         SavedQueriesResponse resp = qService.executeSavedQueries(savedQueryInput);
39         return resp;
40     }
41     else
42     {
43         log.error("No query with name "+queryName+" found");
44         return null;
45     }
46 }
```

5.3. Erweiterungen für den Einsatz der Teamcenterkomponente

Die Teamcenterkomponente alleine ermöglicht noch keinen Datenaustausch mit anderen Anwendungen. Jede Anwendung hat meist ein eigenständiges Datenmodel und eine eigene Semantik. Es ist nicht ungewöhnlich, dass ein Objekt in einem ERP System unterschiedliche Bezeichnungen und/oder eine unterschiedliche Struktur hat, als ein Objekt in einem PLM System. Zusätzlich werden Daten in verschiedenen Anwendungen unterschiedlich verwaltet, gespeichert und ausgetauscht. Während eine Anwendung relationale Datenbanken verwendet, kann eine Andere auf einfachen Dateien oder XML

Dokumenten aufbauen. Eine Herausforderung in der Enterprise Application Integration und bei der Erstellung von Integrationslösung ist das Überwinden dieser Unterschiede. Die Enterprise Integration Pattern bieten einen allgemeinen Lösungsansatz in Form des Message Translator (siehe Kapitel 2.3.1.5).

Der Message Translator ist vergleichbar mit dem Adaptermuster aus Gamma et al. [2004]. Das Adaptermuster lässt Klassen zusammenarbeiten, die wegen inkompatibler Schnittstellen ansonsten dazu nicht in der Lage wären (vgl s.171 Gamma et al. [2004]).

Eine weitere Anforderung ist das Routen, Prüfen und Zusammenstellen von Nachrichten. Informationen müssen regelmäßig bereinigt, aufgeteilt, oder zusammengeführt werden. Camel implementiert für solche Anforderungen verschiedene EIPs wie den *Router* und dessen Spezialisierungen unter anderem den *Aggregator* oder den *Splitter*. Wie auch an vielen anderen Stellen in Camel werden bei diesen EIPs *Expressions* und *Predicates* zur Steuerung verwendet. Expressions und Predicates können auf unterschiedlichste Arten erstellt werden (siehe Kapitel 4.1.2). Für viele Anwendungsfälle kann die *Simple Expression* Language[15] verwendet werden und eine *Expression* oder ein *Predicate* erstellt werden. Die *Simple* Language ermöglicht den Zugriff auf Objekte mit Hilfe der *Object-Graph Navigation Language* (OGNL). Für komplexere Anwendungsfälle können Sprachen wie *XPath*, oder *XQuery* verwendet werden. Jedoch kann die *Simple*-Language nicht direkt verwendet werden, um die Ergebnisse der Teamcenterkomponente zu evaluieren. Die *Simple*-Language setzt Getter und Setter Methoden für die einzelnen Attribute der Datenstrukturen voraus, diese sind jedoch nur für sehr wenige Attribute in den Klassen des Teamcenter-*ClientModels* vorhanden. Zwar ist es möglich neue Klassen zu entwickeln, die das *Expression* Interface bzw. das *Predicate* Interface implementieren und so das native Datenformat der Teamcenterkomponete weiter zu verwenden. Allerdings sind die bestehenden Expression Languages umfangreich an Funktionalitäten, diese Funktionen neu zu entwickeln wäre sehr aufwendig. Aus diesem Grund wird ein neues Datenformat eingeführt und dieses so gestaltet, dass die Expression Languages wie *Simple* oder *XPath* anwendbar sind.

Für die Anwendung von *Simple* müssen Getter und Setter für die einzelnen Attribute verfügbar sein, für *XPath* oder *XQuery* muss das Format in ein XML Dokument überführt werden können. Eine weit verbreitete Art, diese Anforderungen umzusetzten, ist mit Hilfe der Java Architecture for XML Binding (JAXB [16]). Dabei werden die einzelnen Klassen und Attribute mit Annotations versehen, mit deren Hilfe JAXB ein Java-Objekt in ein XML Dokument umwandeln kann.

Im Folgenden wird gezeigt, wie ein Datenformat auf Basis der Teamcenterstrukturen erzeugt werden kann und wie mit Hilfe der *Type Converter* von APACHE CAMEL eine Umwandlung des nativen Teamcenter Datenformats in ein Custom Datenformat durch-

[15] http://camel.apache.org/simple.html
[16] https://jaxb.java.net/

geführt werden kann.

5.3.1. Custom Teamcenter Datenformat

Für alle Datenstrukturen der Teamcenterservices liefert Siemens PLM Software eine zugehörige XML Schema Definition (XSD). Diese Schema Definitionen kann automatisch in Javaklassen mit zugehörigen JAXB Annotations übersetzt werden. Dafür bietet die JAXB einen Binding Compiler in Form des Tools XJC. Seit Version 6 des Java Development Kit ist JAXB ein fester Bestandteil, somit ist XJC verfügbar sobald das JDK ab Version 6 installiert ist. Ist in der Path Variable des Betriebssystems der Pfad zur JDK Installation eingetragen, kann eine Schemadatei mit folgendem Konsolenbefehl in Javaklassen umgewandelt werden:

```
xjc Pfad/zur/Schemadatei.xsd
```

Mit dem Befehl xjc −help können weitere Parameter des Tools angezeigt werden. Im Anhang zeigt Listing 34 eine mit xjc erstellte Klasse, xjc erstellt automatisch die nötigen Annotations für JAXB (u.a. in Zeile 27 in Listing 34).

Für alle Datenstrukturen und darin vorkommende Objekte, die von der Teamcenterkomponente zurückgegeben werden, werden auf diese Art neue Java Klassen erzeugt. Im nächsten Teil wird gezeigt, wie die Umwandlung in das neue Datenformat durchgeführt werden kann.

5.3.1.1. Default Type Converter für TcComponent Camel implementiert das *Message Translator* Muster auf verschiedene Art und Weise. Es könnte ein neuer Prozessor oder eine beliebige Java Bean verwendet werden um die Umwandlung durchzuführen. Auch ist es möglich Camel um ein neues Datenformat zu erweitern, indem man eine Klasse entwickelt und das Interface *DataFormat* implementiert. Es hat sich jedoch gezeigt, dass die *TypeConverter* von Apache Camel (siehe Abschnitt 4.1.3) besonders gut anwendbar sind. In diesem Abschnitt wird gezeigt wie ein *TypeConverter* erstellt wird.

Für die Erkennung von *TypeConverter* verwendet Camel den org.apache.camel.impl.converter.AnnotationTypeConverterLoader und die Datei *TypeConverter* im META-INF Ordner: META−INF/services/org/apache/camel/TypeConverter. Die *TypeConverter*-Datei wird verwendet damit Camel nicht alle Klassen nach Konvertern durchsuchen muss und enthält Eintrage zu den einzelnen *TypeConverter*-Klassen. Erst seit der Camel Version 2.8 ist ein *Fully Qualified Name* (FQN) für die Definition der Klassen erlaubt, davor dürfen in der *TypeConverter*-Datei nur Paketnamen verwendet werden. Es wird jedoch empfohlen FQN zu verwenden. So wird vermieden, gesamte Pakete durchsuchen zu müssen. Alle Klassen die in der *TypeConverter*-Datei angegeben sind, werden durchsucht und alle Methoden die mit der Annotation @Converter markiert sind, werden als

TypeConverter registriert.

Listing 22: META-INF/services/org/apache/camel/TypeConverter

```
1 at.acam.camel.tc.converter.DefaultTcTypeConverter
```

Da alle *TypeConverter* für die Teamcenterkomponente in einer Klasse zusammengefasst sind, ist in Listing 22 nur eine einzelne Klasse angegeben. Die Standard *TypeConverter* von Camel sind direkt in den jeweiligen Paketen angegeben und werden somit geladen, sobald das Jar-File zum Classpath hinzugefügt wird.

Die Klasse at.acam.camel.tc.converter.DefaultTcTypeConverter stellt die eigentliche Implementierung der *TypeConverter* für die Teamcenterkomponente dar. Im Listing 23 ist ein Auszug aus der *DefaultTcTypeConverter*-Klasse gelistet. Apache Camel empfiehlt statische Methoden zu verwenden, die threadsicher und eintrittsinvariant sind.

Listing 23: Auszug aus at.acam.camel.tc.converter.DefaultTcTypeConverter

```
1  @Converter
2  public class DefaultTcTypeConverter {
3          final static Logger log = LoggerFactory.getLogger(DefaultTcTypeConverter.class);
4          @Converter
5          public static at.acam.tc.model.GetItemAndRelatedObjectsResponse
               toAcamGetItemAndRelatedObjectsResponse (GetItemAndRelatedObjectsResponse response){
6                  at.acam.tc.model.GetItemAndRelatedObjectsResponse aResp = new at.acam.tc.model.
                       GetItemAndRelatedObjectsResponse();
7                  for ( GetItemAndRelatedObjectsItemOutput o : response.output){
8                          aResp.getOutput().add(toAcamGetItemAndRelatedObjectsItemOutput(o));
9                  }
10                 aResp.setServiceData(toAcamServiceData(response.serviceData));
11                 return aResp;
12 }
```

Die *TypeConverter* werden automatisch von Camel verwendet falls ein Objekt in Form eines bestimmten Typs angefragt wird. Ist das Objekt keine Instanz des gefragten Typs, durchsucht Camel die *TypeConverter*-Registry nach einer Methode für die Umwandlung. Beispiele für den Einsatz wird in Listing 26 in Zeile 20 und in den Fallbeispielen in Kapitel 6 gezeigt.

5.4. Die erste Teamcenter Integration

Jetzt sind alle grundlegenden Elemente für die Erstellung einer Integrationslösung mit Teamcenter vorhanden und können zusammengefügt werden. Dazu gehört das Messaging und Routing Framework Apache Camel, die neu entwickelte Teamcenterkomponente für Apache Camel und die *DefaultTcTypeConverter*. In diesem Kapitel wird gezeigt wie die einzelnen Teile zu einem neuen Projekt zusammengefügt werden und gestartet werden können. Für die Erstellung des Java-Projekts, wird in diesem Beispiel das IDE Eclipse und Apache Maven verwendet. Der Einsatz dieser Tools erleichtert die Erstellung des Projekts, ist aber keine Voraussetzung.

Der erste Schritt ist die Anlage eines neuen Java-Projekts und das Hinzufügen der benötigten Java Archive (jar) zum Classpath. Mit der Hilfe von Maven-Archetypes kann

Maven automatisch eine geeignete Projektstruktur erstellen. So erhält man einen guten Startpunkt für die Entwicklung einer Integrationslösung. Eine Liste aller offiziellen Maven-Archetypes kann unter https://maven-repository.com/archetypes gefunden werden. Camel spezifische Archetypes sind zusätzlich auf der Webpräsenz von Apache Camel unter http://camel.apache.org/camel-maven-archetypes.html zu finden. Listing 24 zeigt wie ein neues Projekt mit Hilfe von Maven angelegt werden kann. In diesem Fall wird der Archetype *camel-archetype-java* verwendet.

Listing 24: Projektanlage mit Maven

```
1  C:\>mvn archetype:generate -DgroupId=at.acam -DartifactId=MyFirstTeamcenterIntegration -DarchetypeGroupId=org
       .apache.camel.archetypes -DarchetypeArtifactId=camel-archetype-java -DinteractiveMode=false
2  [INFO] Scanning for projects...
3  [INFO]
4  [INFO] ------------------------------------------------------------------------
5  [INFO] Building Maven Stub Project (No POM) 1
6  [INFO] ------------------------------------------------------------------------
7  [INFO]
8  [INFO] >>> maven-archetype-plugin:2.2:generate (default-cli) > generate-sources @ standalone-pom >>>
9  [INFO]
10 [INFO] <<< maven-archetype-plugin:2.2:generate (default-cli) < generate-sources @ standalone-pom <<<
11 [INFO]
12 [INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom ---
13 [INFO] Generating project in Batch mode
14 [INFO] Archetype [org.apache.camel.archetypes:camel-archetype-java:2.15.0] found in catalog remote
15 [INFO] ----------------------------------------------------------------------------
16 [INFO] Using following parameters for creating project from Archetype: camel-archetype-java:2.15.0
17 [INFO] ----------------------------------------------------------------------------
18 [INFO] Parameter: groupId, Value: at.acam
19 [INFO] Parameter: artifactId, Value: MyFirstTeamcenterIntegration
20 [INFO] Parameter: version, Value: 1.0-SNAPSHOT
21 [INFO] Parameter: package, Value: at.acam
22 [INFO] Parameter: packageInPathFormat, Value: at/acam
23 [INFO] Parameter: maven-resources-plugin-version, Value: 2.6
24 [INFO] Parameter: groupId, Value: at.acam
25 [INFO] Parameter: maven-compiler-plugin-version, Value: 2.5.1
26 [INFO] Parameter: slf4j-version, Value: 1.7.10
27 [INFO] Parameter: version, Value: 1.0-SNAPSHOT
28 [INFO] Parameter: exec-maven-plugin-version, Value: 1.2.1
29 [INFO] Parameter: log4j-version, Value: 1.2.17
30 [INFO] Parameter: camel-version, Value: 2.15.0
31 [INFO] Parameter: package, Value: at.acam
32 [INFO] Parameter: artifactId, Value: MyFirstTeamcenterIntegration
33 [INFO] project created from Archetype in dir: C:\MyFirstTeamcenterIntegration
34 [INFO] ------------------------------------------------------------------------
35 [INFO] BUILD SUCCESS
36 [INFO] ------------------------------------------------------------------------
37 [INFO] Total time: 7.552 s
38 [INFO] Finished at: 2015-03-19T15:12:13+01:00
39 [INFO] Final Memory: 12M/231M
40 [INFO] ------------------------------------------------------------------------
```

Anschließend kann mit dem Befehl mvn eclipse: eclipse das erstellte Projekt in ein Eclipseprojekt umgewandelt werden und in Eclipse, als bestehendes Projekt, importiert werden.

Im nächsten Schritt müssen die benötigten Abhängigkeiten definiert werden. Mit Maven werden die Abhängigkeiten über das pom.xml gesteuert. Durch die Erzeugung des Projektes über den Camel Archetype sind einige der nötigen Abhängigkeiten, wie das camel-core Package und die Logging Komponenten sjf4j und log4j, bereits im pom.xml eingetragen. Zusätzlich werden noch die Packages mit der Teamcenterkomponente und dem *DefaultTcTypeConverter* benötigt. Das Listing 35 im Anhang zeigt wie ein entspre-

chendes pom.xml aussehen kann. Alternativ kann anstatt MAVEN, auch GRADLE[17] oder ein anderes Build-Management-Tool verwendet werden. Soll kein solches Tool zum Einsatz kommen, müssen alle Abhängigkeiten manuell zum Classpath hinzugefügt werden.

Der Maven Archetype erstellt zusätzlich eine einfache Camel Routendefinition (MyRouteBuilder.java) und eine Applikation Klasse (MainApp.java) um die Camel Anwendung zu starten.

MainApp

Die MainApp Klasse ermöglicht die Ausführung der Camel-Anwendung. Über die main - Methode kann die Anwendung wie jede Javaanwendung gestartet werden. Anschließend wird in Zeile 12 in Listing 25 ein neues org.apache.camel.main.Main Objekt erzeugt. Dieses initialisiert im Hintergrund die Camel Laufzeitumgebung, unter anderem wird ein *CamelContext* und eine Registry erzeugt. In Zeile 14 wird die automatisch von Maven erzeugte Camel-Route zu der Laufzeitumgebung dem *CamelContext* hinzugefügt. Abschließend wird in Zeile 15 die Camellaufzeitumgebung gestartet. Durch *enableHangupSupport* kann die Anwendung mit Ctrl+C jederzeit beendet werden.

Listing 25: MainApp.java

```
1  package at.acam;
2  import org.apache.camel.main.Main;
3  /**
4   * A Camel Application
5   */
6  public class MainApp {
7
8      /**
9       * A main() so we can easily run these routing rules in our IDE
10      */
11     public static void main(String... args) throws Exception {
12         Main main = new Main();
13         main.enableHangupSupport();
14         main.addRouteBuilder(new MyRouteBuilder());
15         main.run(args);
16     }
17 }
```

MyRouteBuilder

In der Klasse MyRouteBuilder.java wird die Definition der Routen für dieses Beispiel vorgenommen. Um Routen in einer Klasse definieren zu können, muss die Klasse org. apache.camel.builder.RouteBuilder abgeleitet werden und die Methode *configure()* implementiert werden. In dieser Methode können im Anschluss beliebig viele Routen definiert werden. Eine Route beginnt immer mit *from()*, gefolgt von einer beliebigen Anzahl von *to()*-Blöcken die mit Punkten („.“) verknüpft werden. Geschlossen wird eine Route mit einem Semikolon. Eine Route kann aus beliebig viele Schritten bestehen.

MAVEN erzeugt bei der Anlage des Projekts automatisch eine exemplarische Route in der Klasse *MyRouteBuilder.java*. Diese wurde durch eine sehr einfache Route mit der Teamcenterkomponente ersetzt (siehe Zeile 16 bis 22 in Listing 26).

[17]https://gradle.org

Ausgelöst wird die Route durch einen Consumer, der mit Hilfe einer URL im *from()* Teil definiert wird. In diesem Beispiel ist das ein Timer (siehe Zeile 16) der einmalig auslöst und einen leeren Exchange an seinen Nachfolger in der Route übergibt. Der Timer startet sobald die Camelanwendung gestartet wird. In Zeile 17 wird eine Id für die Route vergeben, so können Loggingmeldungen besser zugeordnet werden. Die Vergabe einer solchen Id ist optional, aber empfehlenswert. Wird keine Id angegeben, vergibt Camel automatisch eine fortlaufende Nummer. In Zeile 18 wird der Exchange mit Eingabeinformationen für den Teamcenterservice befüllt. In diesem Beispiel ist das der Header *tc.GIARO.item.id*. Er muss ein String Array sein und wird mit Hilfe einer *Expression* gesetzt. Der Wert des Headers enthält alle ItemIds für die Items, die aus TEAMCENTER geladen werden sollen. Hier wird das Item mit der ItemId „000001“ gesucht. In echten Integrationslösungen würde diese Nummer dynamisch befüllt werden und nicht statisch in der Route definiert werden. Nachdem der Header gesetzt wurde, wird der Exchange mit den Eingabeinformationen an den Teamcenterendpunkt weitergeleitet (siehe Zeile 19). Dieser erzeugt im Hintergrund einen geeigneten Producer und führt die angefragte Serviceaufgabe aus. Die Ergebnisse werden wieder in den Exchange geschrieben und in der Route an den nächsten Prozessor in der Zeile 20 weitergegeben. Das ist ein Prozessor mit dem Namen *convertBodyTo*, dieser wandelt den Körper der *In*-Nachricht des aktuelle Exchanges in ein gewünschtes Format um. Dazu sucht Camel im Hintergrund nach einem geeigneten *TypeConverter* in der *TypeConverterRegistry*. Da der *DefaultTcTypeConverter* Teil des Projektes ist, findet Camel den in Abschnitt 5.3.1.1 beschrieben *TypeConverter* für TEAMCENTER und führt die Umwandlung durch. Abschließend wird in Zeile 21 eine Log-Komponente verwendet um das Ergebnis auszugeben. Zu beachten ist, dass in der Log-Komponente mit ${body} eine *Simple Expression* angegeben ist. Camel erkennt, dass der Körper der Nachricht in Form eines Strings benötigt wird und führt wiederum automatisch die Umwandlung durch. Da die *GetItemAndRelatedObjectsResponse* Klasse JAXB Annotations hat, wird automatisch JAXB verwendet um das Objekt in ein XML umzuwandeln. Damit Camel JAXB verwenden kann, muss das Camel-JAXB Paket als Abhängigkeit zum Projekt hinzugefügt werden (siehe Zeile 37 in Listing 35).

Listing 26: MyRouteBuilder.java

```
1  package at.acam;
2
3  import org.apache.camel.builder.RouteBuilder;
4  import at.acam.tc.model.GetItemAndRelatedObjectsResponse;
5
6  /**
7   * A Camel Java DSL Router
8   */
9  public class MyRouteBuilder extends RouteBuilder {
10
11     /**
12      * Let's configure the Camel routing rules using Java code...
13      */
14     public void configure() {
15
16         from("timer://einTimer?repeatCount=1")
```

```
17                  .routeId("MyFirstTeamcenterIntegrationRoute")
18                  .setHeader("tc.GIARO.item.ids", constant(new String [] {"000001"}))
19                  .to("tc://192.168.0.176:8080/tc?tcMethod=getItemAndRelatedObjects&tcUser=infodba&tcPass=
                        infodba")
20                  .convertBodyTo(GetItemAndRelatedObjectsResponse.class)
21                  .log("${body}")
22                  ;
23      }
24  }
```

Diese kurze Route zeigt wie einfach der Teamcenter-Komponente innerhalb einer Route Anweisungen gegeben werden können und anschließend die benötigten Teamcenterservices ausgeführt werden. Der Ablauf für den Einsatz der Teamcenterkomponente ist immer gleich, zuerst werden die Eingabeinformationen in den Exchange eingetragen, dann wird der Exchange an einen Teamcenterendpunkt übergeben und abschließend liefert der Endpunkt das Ergebnis im Körper oder in einem Header des Exchanges zurück. Welche Aufgaben die Teamcenterkomponete durch führen kann und welche Eingabeinformationen dafür nötig sind, ist im Abschnitt 5.2 beschrieben.

6. Fallbeispiele

Im abschließenden Kapitel dieser Arbeit werden Fallbeispiele mit typischen Integrationsaufgaben von Teamcenter beschrieben und eine Lösung mit Hilfe von Apache Camel und der Teamcenterkomponente präsentiert. Bei allen Beispielen wird zuerst ein Integrationsproblem definiert, anschließend wird ein Konzept für die Lösung mit Hilfe der *Enterprise Integrations Pattern* beschrieben und abschließend wird das Konzept mit Apache Camel umgesetzt.

Teamcenter verwendet das sogenannte *Named User Licensing* (siehe 3.6). Es besagt, dass für jede Interaktion mit Teamcenter ein eindeutiger Benutzer verwendet werden muss und jeder Benutzer genau einer Person zugeordnet sein muss. Ein Teamcenter Benutzer darf nicht von mehreren Personen verwendet werden. Bei Integrationslösungen muss sichergestellt werden, dass diese Lizenzbedingungen nicht verletzt werden. Es müssen bei jeder Anfrage Teamcenterbenutzerinformation mit übergeben werden und die Teamcenteranfrage mit diesen Authentifizierungsinformationen durchgeführt werden, oder über andere Mechaniken wie Webserver oder Authentifizierungsdienste sichergestellt werden, dass nur berechtigte Benutzer die Services verwenden können. Um die Komplexität der Beispiele zu reduzieren, werden die Benutzerdaten in den folgenden Beispielen statisch festgelegt.

6.1. Fallbeispiel 1 - Artikelinformationen abfragen

In dem ersten Fallbeispiel sollen Informationen über Teamcenter-Objekte mit anderen Systemen geteilt werden. Das ist ein typischer Anwendungsfall bei der Integration von Teamcenter mit einem ERP System. Zum Beispiel benötigt das ERP System für Bestellvorgänge von bestimmten Artikeln aktuelle Werkstoffinformationen, Abmessungen oder ähnliche Informationen, die typischerweise im Engineering und somit im PLM System festgelegt werden. In diesem Fallbeispiel sollen Teileinformationen über eine Webresource, einen Webservice, für das ERP System zugängig gemacht werden. Welche Attribute für welche Objekttypen benötigt werden, wurde vorab festgelegt. Eine entsprechende Object Property Policy Datei (siehe Abschnitt 3.7.6) mit dem Namen ObjectPolicy1.xml wurde erstellt und dem Teamcenterserver verfügbar gemacht. Das Ergebnis soll in Form eines XML Dokuments, als Antwort auf einen HTTP-Request an das ERP System zurückgegeben werden. Der HTTP-Request muss Item-Id und Revisions-Id des gesuchten Artikels als Query Parameter *item_id* und *rev_id* enthalten.

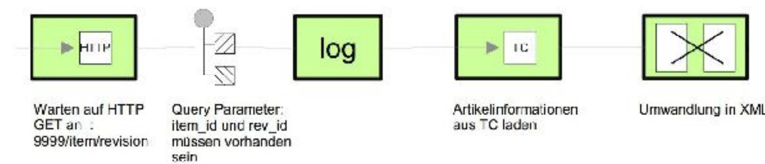

Abbildung 25: Fallbeispiel 1

Für die Umsetzung dieser Integrationslösung wird zuerst eine Webresource (Endpunkt) erstellt, diese nimmt HTTP-GET-Anfragen entgegen. Die Anfrage wird in eine Camel Exchange verpackt und an eine Log Komponente übergeben. Bei allen Integrationslösungen ist die Dokumentation von Events und Aktionen ein wichtiger Bestandteil und sollte keineswegs vernachlässigt werden. In diesem einfachen Anwendungsfall wird ein Eintrag mit der Anfrage in einem Logfile gespeichert. Anschließend werden die gewünschten Informationen mit Hilfe der Teamcenterkomponente geladen und in eine XML umgewandelt. Das Ergebnis wird als Antwort auf die Anfrage zurückgegeben.

Eine Camelroute für diese Integrationslösung wird in Listing 27 gezeigt.

Listing 27: Route für Fallbeispiel 1

```
1  from("jetty:http://0.0.0.0:9999/item/revision?httpMethodRestrict=GET").routeId("Fallbeispiel1")
2    .log("New request for ${header.item_id} ${header.rev_id}")
3    .setHeader("tc.GIARO.item.ids", simple("${header.item_id}"))
4    .setHeader("tc.GIARO.rev.id", simple("${header.rev_id}"))
5    .setHeader("tc.GIARO.rev.processing", constant("Ids"))
6    .setHeader("tc.policy.name", simple("ObjectPolicy1"))
7    .to("tc://192.168.0.176:8080/tc?tcMethod=getItemAndRelatedObjects&tcUser=acam&tcPass=acam")
8    .convertBodyTo(GetItemAndRelatedObjectsResponse.class);
```

Bei dieser Implementierung wird ein JETTY-Endpunkt als Consumer und somit als Startpunkt der Route verwendet. Dazu muss das Camel-Jetty Packet als Abhängigkeit zum Projekt hinzugefügt werden. Der Camel-Jetty-Endpunkt erstellt eine Webresource unter dem in der URI angegebenen Pfad, in diesem Fall *Localhost* mit dem Port *9999* unter */item/revision* (siehe Zeile 1 in Listing 27) und wartet auf HTTP-Requests. Die Endpunkt Option *httpMethodRestrict=GET* erlaubt für diesen Endpunkt nur HTTP-Anfragen vom Typ GET. Wird eine Anfrage gestellt, verpackt die JETTY-Komponente diese in einen neuen Exchange, dabei werden alle Query Parameter des HTTP-Requests als Header in den Exchange übertragen. Ebenso wird der Körper des Request in den Exchange übertragen und der Exchange an den nächsten Prozessor übergeben. Zu beachten ist, dass die JETTY-Komponente das Pattern des Exchanges automatisch auf *InOut* (siehe Abschnitt2.2.3.1) setzt. So wird der abschließende Exchange dieser Route, automatisch als Antwort auf die Anfrage zurückgegeben. Ein möglicher HTTP-Request könnte wie folgt aussehen:

http://*ServiceHost:9999/item/revision?item_id=000001&rev_id=00*

Um die gewünschten Informationen von der Teamcenterkomponente laden zu lassen, müssen entsprechende Inputparameter in Form von Headern im Exchange gesetzt wer-

den. In diesem Fallbeispiel passiert das in den Zeilen 3-6 in Listing 27, in Zeile 7 werden die gewünschten Informationen mit Hilfe der Teamcenterkomponente aus TEAMCENTER geladen. Die Authentifizierungsinformationen sind in diesem Fallbeispiel statisch in der Route definiert (Zeile 7 in Listing 27). Die Teamcenterkomponente retourniert das Ergebnis des Serviceaufrufs im Körper der In-Nachricht des aktuellen Exchanges. Abschließend wird in Zeile 8 in Listing 27 das Ergebnis mit Hilfe der TypeConverter von Camel in eine Klasse mit JAXB Annotations umgewandelt. Auf diese Weise, kann Camel bei der Ausgabe automatisch die Umwandlung in ein XML durchführen.

6.2. Fallbeispiel 2 - Artikelinformationen aktualisieren

In diesem Fallbeispiel müssen Änderungen von ERP-Objekten an TEAMCENTER weitergegeben und dort aktualisiert werden. Informationen zu einem Artikel/Objekt sind oft in mehreren Datenbanken redundant vorhanden. Änderungen die in einem Fremdsystem vorgenommen werden, müssen auch an TEAMCENTER weitergegeben werden um die Informationen aktuell zu halten. Erneut sollen die Informationen über einen HTTP-Request übergeben werden. Für die Aktualisierung eines Artikels in Teamcenter werden zusätzlich zu der Item-Id und Revision-Id auch Informationen über die zu aktualisierenden Attribute benötigt. Um dieses Beispiel einfach zu halten, wird davon ausgegangen, dass nur ein Artikel pro Aufruf aktualisiert wird. Jeder Aufruf enthält ein XML mit allen benötigten Informationen im Körper. Das XML muss dabei dem Schema in Listing 36 entsprechen, dieses Schema ist ausschließlich für dieses Fallbeispiel entwickelt worden. Normalerweise wird in Abhängigkeit zu dem Fremdsystem, in jedem Integrationsprojekt das Austauschformat definiert.

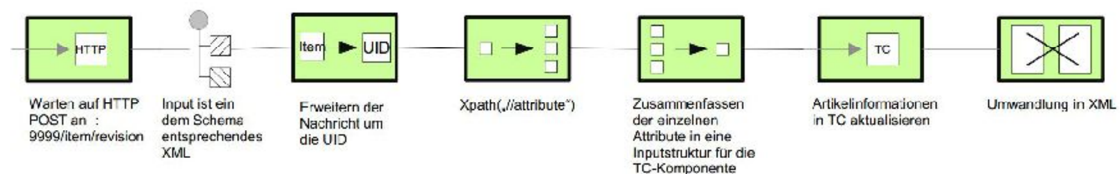

Abbildung 26: Fallbeispiel 2

In Abbildung 26 wird eine mögliche Integrationslösung für Fallbeispiel 2, mit Hilfe der EIPs beschrieben. Genauso wie in Fallbeispiel 1 wird eine Webresource (Endpunkt) erzeugt, diese wartet auf HTTP-POST-Anfragen. Trifft eine entsprechende Anfrage ein, wird sie in einen Camel Exchange verpackt und in der Route weitergegeben. Über einen *Content-Enricher* wird die Uid zu dem entsprechenden Artikel aus TEAMCENTER geladen und diese Information dem Exchange hinzugefügt. Die Uid wird später von der Teamcenterkomponente benötigt, um die Aktualisierung des Artikels durchführen zu können. Anschließend wird die Nachricht mit Hilfe eines Splitters aufgespalten und jedes gefundene Attribut zu einer Inputstruktur für die Artikelaktualisierung zusammen-

gefasst. Abschließend wird die Änderung in Teamcenter durchgeführt, das Ergebnis in ein XML umgewandelt und als Antwort auf den HTTP-Request wieder an den Aufrufenden zurückgegeben.

Aus Gründen der Übersichtlichkeit wurde diesmal auf explizite Angabe der Loggingvorgänge verzichtet. Wie schon in Fallbeispiel 1 besprochen, ist das Logging ein wesentlicher Bestandteil jeder Lösung. Grundsätzlich sollten alle Aktionen und Events geloggt werden.

Listing 28: Route für Fallbeispiel 2

```
1  Namespaces ns = new Namespaces("tns", "http://www.acam.at/Object");
2
3  from("jetty:http://0.0.0.0:9999/item/revision?httpMethodRestrict=POST").routeId("Fallbeispiel2")
4    .convertBodyTo(String.class)
5    .log("New POST request for ${body}")
6    .setHeader("tc.GIARO.item.ids", ns.xpath("tns:Object/@item_id", String.class))
7    .setHeader("tc.GIARO.rev.id", ns.xpath("tns:Object/@rev_id", String.class))
8    .setHeader("tc.GIARO.rev.processing", constant("Ids"))
9    .setHeader("attr", constant(new HashMap<String,String>()))
10   .split().xpath("//tns:attribute", ns)
11     .setHeader("tempAttrName", ns.xpath("//tns:attribute/@name", String.class))
12     .setHeader("tempAttrValue", ns.xpath("//tns:attribute/@value", String.class))
13     .process(new Processor() {
14       @Override
15       public void process(Exchange exchange) throws Exception {
16         exchange.getIn().getHeader("attr", (new HashMap<String,String>()).getClass()).put(
17                 exchange.getIn().getHeader("tempAttrName", String.class),
18                 exchange.getIn().getHeader("tempAttrValue", String.class));
19     }})
20     .end()
21   .enrich("tc://192.168.0.176:8080/tc?tcMethod=getItemAndRelatedObjects&tcUser=acam&tcPass=acam&tcSessionKey
         =1")
22   .convertBodyTo(GetItemAndRelatedObjectsResponse.class)
23   .to("xslt:removeNamespace.xslt")
24   .setHeader("uid", xpath("GetItemAndRelatedObjectsResponse/output/itemRevOutput/itemRevision/@uid", String.
         class))
25   .setHeader("tc.SP.propinfos", constant(new HashMap<String,HashMap<String,String>>()))
26   .process(new Processor() {
27     @Override
28     public void process(Exchange exchange) throws Exception {
29       exchange.getIn().getHeader("tc.SP.propinfos",(new HashMap<>()).getClass()).put(
30           exchange.getIn().getHeader("uid", String.class),
31           exchange.getIn().getHeader("attr", (new HashMap<String,String>()).getClass()));
32     }
33   })
34   .to("tc://192.168.0.176:8080/tc?tcMethod=setProperties&tcUser=acam&tcPass=acam&tcSessionKey=1")
35   .setBody(simple("${header.tc.SP.result}"))
36   .convertBodyTo(SetPropertyResponse.class)
37   .log("${body}");
```

Auch in diesem Beispiel wird ein JETTY Consumer verwendet. Dieser wartet darauf HTTP-POST-Anfragen am Port *9999* unter */item/revision* zu empfangen (siehe Zeile 3 in Listing 28). Von der HTTP-Anfrage wird erwartet, dass sie ein dem Schema in Listing 36 entsprechendes XML im Körper enthält. Für die Einfachheit in dieser Lösung wird das Schema des XML Dokuments nicht validiert. In produktiv eingesetzten Lösungen, ist eine solche Validierung zu empfehlen. In Zeile 4 wird die HTTP-Nachricht explizit in einen String umgewandelt, diese kann später einfacher bearbeitet werden. Die Umwandlung wird automatisch mit Hilfe der TypeConverter von Camel durchgeführt. In Zeile 6 und 7 werden mit Hilfe von XPATH die Item-Id und Revisions-Id aus dem empfangenen XML-Dokument ausgelesen. Zu beachten ist der im XML-Dokument definierte Namespace.

Dieser Namespace muss im XPATH-Ausdruck angegeben werden. In einer Camel Route wird das durch die Anlage eines Namespace Objekts durchgeführt (siehe Zeile 1). Der Namespace kann anschließend im XPATH verwendet werden. Mit Hilfe der Item-Id und Revisions-Id kann die Uid aus Teamcenter geladen werden. Diese wird später benötigt um eine Aktualisierung eines Objekts in Teamcenter durchführen zu können. Wie schon in Fallbeispiel 1 werden dazu die Header *tc.GIARO.item.ids* und *tc.GIARO.rev.id* mit den entsprechenden Werten befüllt.

Im XML sind noch die zu ändernden Attributwerte enthalten, diese müssen ausgelesen und verarbeitet werden. Mit Hilfe eines *Splitters* (Zeile 10) können mit einer XPATH-Expression alle in dem XML enthaltenen Attribute einzeln durchlaufen werden. Für jedes Attribut werden Name und Wert ausgelesen und in einem temporären Header (*tempAttrName*, *tempAttrVale*) gespeichert (Zeile 11 und 12). Diese Header sind nur für den Dauer des *Splitters* vorhanden und werden am Ende des Splitter (Zeile 20) gelöscht. Um keine neue Aggregationsstrategie für die Sicherung der Werte entwickeln zu müssen, wird bevor der *Splitter* ausgeführt wird, ein Header mit einer HashMap zum Speichern der Werte angelegt (Zeile 9). Dieser Header ist auch nach Beendigung des Splitters noch vorhanden. Bei jedem Durchlauf des *Splitters* werden die extrahierten Werte abschließend in die HashMap übertragen. Mit der Java DSL ist das aktuell nicht einfach möglich, jedoch gibt es verschiedene Wege Javacode innerhalb einer Route einzusetzen.

Ein Weg ist in der Route einen neuen Prozessor zu definieren (siehe Zeile 15). Für jeden Prozessor muss die Methode *process()* implementiert werden. Als Übergabewert wird ein Exchange angegeben, die Methode hat keinen Rückgabewert. Wird dieser Prozessor in der Route aufgerufen, startet Camel die Methode *process()* und übergibt automatisch den aktuellen Exchange. In der Methode kann anschließend ein beliebiger Java Code ausgeführt werden. In diesem Fall wird der Header mit der HashMap aus dem Exchange geladen und mit den temporären Werten befüllt. Nach dem Abschluss des *Splitters* enthält die Map alle Namen und Werte der in TEAMCENTER zu aktualisierenden Attribute. Jetzt ist das Einlesen des XML-Dokuments abgeschlossen.

Für den Aktualisierungsvorgang in TEAMCENTER wird die Uid des zugehörigen Objekts benötigt. Diese wird mit Hilfe der Teamcenterkomponente in Zeile 21 aus TEAMCENTER geladen. Verwendet wird in diesem Fall das *Enricher-Pattern*. Dieses ermöglicht die Anreicherung der Nachricht mit zusätzlichen Informationen, in diesem Beispiel der Uid. Dazu wird das Schlüsselwort *enrich* verwendet und ein Endpunkt per URI angegeben. In diesem Fall ein Teamcenterendpunkt. Der Endpunkt wird verwendet um die zusätzlichen Informationen aus TEAMCENTER zu laden. Dem *Enricher* kann optional eine *Aggregation-Strategie* angegeben werden. In dieser kann definiert werden, wie der Exchange, der dem Enricher übergeben wird und der Exchange der von dem Endpunkt erzeugt wird, zusammengeführt werden sollen. Wird keine Strategie angegeben, wird einfach der neu entstandene Exchange weiterverwendet und der Ursprüngliche verwor-

fen. Da der Teamcenterendpunkt nur die *In* Nachricht des ursprünglichen Exchanges verändert und der neue Exchange alle Informationen des alten Exchange enthält, muss keine Strategie verwendet werden. In diesem Fall könnte statt *enrich*, auch das Schlüsselwort *to* verwendet werden, jedoch wird durch den Enricher verdeutlicht welche Funktion umgesetzt werden soll.

Das Ergebnis des *getItemAndRelatedObjects*-Aufruf durch die Teamcenterkomponente wird in den Körper der *In*-Nachricht geschrieben. Aus dieser Ergebnisstruktur muss die Uid extrahiert werden. Für diese Aufgabe wird wieder XPath verwendet und auch diesmal muss dafür die Struktur zuerst in ein XML-Dokument umgewandelt werden (Zeile 22). Anschließend kann die Uid aus dem Ergebnis extrahiert werden. Dazu muss bekannt sein wie die Ergebnisstruktur aufgebaut ist. Die Uid ist ein Attribut des ItemRevision-Elements und kann mit dem XPath AusdruckGetItemAndRelatedObjectsResponse/output/itemRevOutput/itemRevision/@uid angesprochen werden. Der Aufbau der Ergebnisstrukturen kann in den Schemafiles der Teamcenter-Services nachgeschlagen werden. Die Uid wird in einem Header für die spätere Verwendung zwischengespeichert (siehe Zeile 24). Da die XPath Implementierung mit einigen Namespacedeklarationen nicht ohne weiteres zurecht kommt, werden in Zeile 23 alle Namespaces aus dem XML entfernt. Erreicht wird das durch den Einsatz einer XSLT-Komponente. Das entsprechende Stylesheet ist im Anhang im Listing 38 zu finden.

In Zeile 25 wird die Eingabestruktur für den *setProperties*-Service der Teamcenterkomponente erstellt. Da diese Struktur nicht ohne weiteres mit Hilfe der DSL befüllt werden kann, wird erneut ein Prozessor erstellt und Java Code für die Befüllung verwendet (Zeile 26 bis 33). Abschließend wird der Teamcenter-Service aufgerufen (Zeile 34), das Ergebnis in den Körper der Nachricht geschrieben (Zeile 35) und in eine Struktur mit entsprechenden XML Annotations umgewandelt (Zeile 36).

Da das Exchange-Pattern automatisch von der *Jetty*-Komponente auf *InOut* (siehe Abschnitt2.2.3.1) gesetzt wird, wird das Ergebnis der Route automatisch von der Jetty Komponete an den Aufrufer zurückübergeben. Da die Ergebenisstruktur XML Annotations besitzt, wird automatisch eine Umwandlung in ein XML Dokument durchgeführt.

Die Besonderheiten an diesem Fallbeispiel sind die Verwendung des *Splitter-Pattern* und des *Enricher-Pattern*. Außerdem der Einsatz von Prozessoren direkt in einer Route. Hier sollte darauf geachtet werden, keinen zu exzessiven Einsatz davon zu machen, da die Lesbarkeit der Route darunter leidet. Für einfache kurze Codestücke ist sie jedoch geeignet. Der Aufwand eine eigene Javabean zu erstellen oder eine Bestehende um eine entsprechende Methode zu erweitern und diese später zu warten, wäre relativ hoch. So wird die reduzierte Lesbarkeit der Route gerechtfertigt.

6.3. Fallbeispiel 3 - Erstellen von Objekten in Teamcenter

Dieses Fallbeispiel beschäftigt sich mit der Anlage von Objekten in TEAMCENTER. Bei der Anlage eines Artikels muss bekannt sein welchen Typ von Geschäftsobjekt dieser Artikel repräsentieren soll und welche Eigenschaften dieser Artikel haben soll. Die Anlage soll wiederum per HTTP-Request angestoßen werden. Die nötigen Informationen werden im Körper der Nachricht als XML übergeben. Das XML-Dokument muss dem Schema aus Fallbeispiel 2 entsprechen, siehe Listing 36.

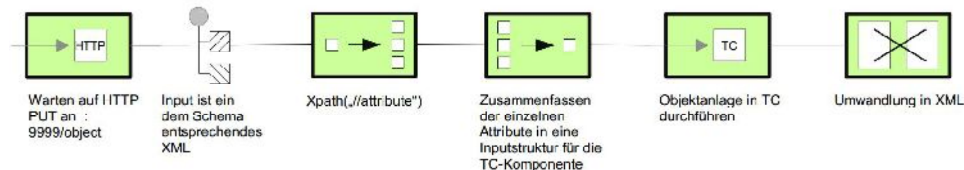

Abbildung 27: Fallbeispiel 3

Die Lösung dieses Fallbeispiels ähnelt dem Fallbeispiel 2, jedoch wird kein *Enricher* benötigt. Das Input Dokument wird über eine Webresource empfangen, in diesem Fall wartet der JETTY-Consumer auf eine HTTP-PUT-Request. Anschließend wird das XML Dokument aufgesplittet und die einzelnen Infomationen eingelesen. Abschließend werden die Inputinformationen in die für die Teamcenterkomponente benötigten Header übertragen und der *createObjects*-Service ausgeführt.

Listing 29: Route für Fallbeispiel 3

```
1  Namespaces ns = new Namespaces("tns", "http://www.acam.at/Object");
2  from("jetty:http://0.0.0.0:9999/object?httpMethodRestrict=PUT").routeId("Fallbeispiel3")
3          .convertBodyTo(String.class)
4          .log("New PUT request for ${body}")
5          // extract information from input message and create input structure for createObjects Service
6          .setHeader("tc.CO.typename", ns.xpath("tns:Object/@type", String.class))
7          // extract each attribute name and value from input xml and store it in a temporary variable
8          .split().xpath("//tns:attribute", ns).aggregationStrategy(new AggregationStrategy()
9            { // new inline AggregationsStrategy
10           @Override
11           public Exchange aggregate(Exchange oldExchange, Exchange newExchange) {
12
13             // the first time we aggregate we only have the new exchange, so we just return it
14             if (oldExchange == null)  return newExchange;
15             String  oldNames = oldExchange.getIn().getHeader("tc.CO.string.name", String.class);
16             String oldValues = oldExchange.getIn().getHeader("tc.CO.string.value", String.class);
17             String newNames = newExchange.getIn().getHeader("tc.CO.string.name", String.class);
18             String newValues = newExchange.getIn().getHeader("tc.CO.string.value", String.class);
19             if (oldNames.matches("^$")) {
20             // no old values available just copy the name and value to the oldExchange
21               oldExchange.getIn().setHeader("tc.CO.string.name", newNames);
22               oldExchange.getIn().setHeader("tc.CO.string.value", newValues);
23             }
24             else
25             {
26               // old values available append "," and the new name and value
27               oldExchange.getIn().setHeader("tc.CO.string.name", oldNames+","+newNames);
28               oldExchange.getIn().setHeader("tc.CO.string.value", oldValues+","+newValues);
29             }
30           return oldExchange;
31           }
32         })
33         // add attribute to name and value header
```

```
34            .setHeader("tc.CO.string.name", ns.xpath("//tns:attribute/@name", String.class))
35            .setHeader("tc.CO.string.value", ns.xpath("//tns:attribute/@value", String.class))
36          .end()
37          // execute setProperties service
38          .to("tc://192.168.0.176:8080/tc?tcMethod=createObjects&tcUser=acam&tcPass=acam&tcSessionKey=1")
39          .convertBodyTo(CreateResponse.class)
40            .log("${body}")
41            ;
```

Die Übernahme des XML-Dokuments wird von einem JETTY-Consumer übernommen. Es werden nur HTTP-PUT-Anfragen akzeptiert. Anschließend werden die Eingabeinformationen aus dem XML Dokument ausgelesen und in die entsprechenden Header für den *createObjects*-Service übertragen. Um die einzelnen Attribute einzulesen, kommt ein Splitter zum Einsatz (siehe Zeile 8), der das XML Dokument in die einzelnen Vorkommnisse der Elemente *attribute* aufteilt. Für jedes Vorkommnis wird ein neuer Exchange angelegt und die Route zwischen dem *Splitter* und dem Schlüsselwort *end()* durchlaufen. Um die Ergebnisse der einzelnen Durchläufe zusammenzuführen, wird eine *Aggregation*-Strategie verwendet. In diesem Beispiel wird die *Aggregation*-Strategie direkt innerhalb der Route definiert (Zeile 8 bis 32). Nach jedem Durchlauf des *Splitters* wird der alte Exchange mit dem neuen Exchange zusammengeführt. So wird aus den einzelnen Vorkommnissen der *Attribute*-Elemente, ein String mit Attributnamen und ein String mit Attributwerten erzeugt. Die Namen und Werte sind jeweils durch einen ',' getrennt. Die *Aggregation*-Strategie ermöglicht das komfortable Zusammenführen der einzelnen Ergebnisse des *Splitters*.

Abschließend wird der *createObject*-Service aufgerufen, das Ergebnis in ein XML Dokument umgewandelt und an den Aufrufer zurückgemeldet.

6.4. Fallbeispiel 4 - Load Balancing

Die Resourcen eines Teamcenterservers sind limitiert. Die Teamcenteranwendungen sind auf eine bestimmte Benutzerzahl und für eine bestimmte Belastung ausgelegt. Bisher haben alle Fallbeispiele, keine Rücksicht auf diese Tatsache genommen. Zwar wird in Fallbeispiel 2 und 3 die URI Option *tcSessionKey* verwendet und so verhindert, das bei jedem Aufruf ein neuer Teamcenterserverprozess erzeugt wird. Aber in manchen Anwendungsfällen ist das nicht ausreichend. Speziell wenn die Endpunkt URI dynamisch aufgrund von Eingabeinformationen erzeugt wird, kann nur schwer kontrolliert werden, wieviele Teamcenterserverprozesse erzeugt werden. Um zu verhindern, dass so wie im Fallbeispiel 1 eine große Anzahl von Anfragen den Teamcenterserver überlasten würden, wird im aktuellen Fallbeispiel gezeigt, wie die Anzahl der Serverprozesse besser kontrolliert werden kann. Die Aufgabe dieses Fallbeispiels ist die Erstellung eines Workflowprozesses, für diese Aufgabe sollen maximal 2 Teamcenterserverprozesse gleichzeitig verwendet werden. Über einen HTTP-PUT-Request soll die Anlage der Workflows angestoßen werden. Die nötigen Informationen für den Workflow werden als XML Dokument im Körper übertragen. Das XML entspricht dem Schema an Listing 37.

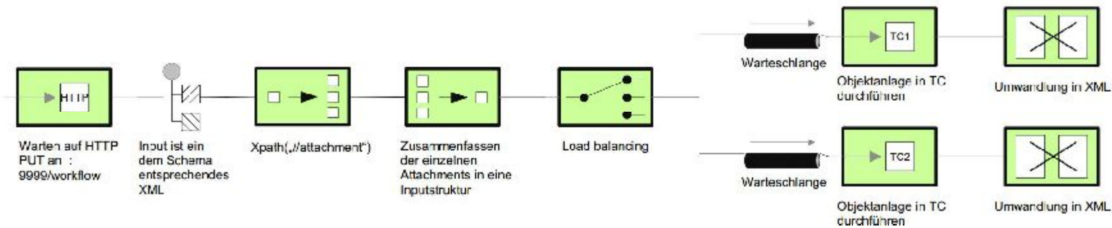

Abbildung 28: Fallbeispiel 4

Für die Entkopplung der Durchführung der Serviceaufrufe und der HTTP-Anfragen wird eine Warteschlage verwendet. Die Anfragen werden an eine Warteschlage übergeben und dort gespeichert. Der Teamcenterendpunkt nimmt eine Anfrage aus der Warteschlange und verarbeitet diese. Erst wenn die Bearbeitung abgeschlossen ist, wird die nächste Anfrage entnommen. So können eine Vielzahl von Anfragen mit einer Ressource ausgeführt werden. Um die Durchführung der Anfragen zu beschleunigen und so die Verfügbarkeit zu erhöhen, können weitere Endpunkte und Warteschlagen hinzugefügt werden. Mit Hilfe eines *Load Balancers*, einer speziellen Form eines *Message Routers*, werden die Anfragen an die einzelnen Warteschlagen und Endpunkte verteilt. Alternativ zu dem Load Balancer, könnten die Teamcenterendpunkte die Nachrichten direkt von einer gemeinsamen Warteschlange konsumieren. Auf diese Weise könnten die Ressourcen besser ausgelastet werden. Sobald eine Ressource verfügbar ist würde diese die nächste Nachricht aus der Warteschlange konsumieren. Mit dem Load Balancer kann jedoch gesteuert werden wie die Nachrichten aufgeteilt werden sollen.

Listing 30: Route für Fallbeispiel 4

```
1  Namespaces wf = new Namespaces("tns", "http://www.acam.at/TcWorkflow");
2  from("jetty:http://0.0.0.0:9999/workflow?httpMethodRestrict=PUT").routeId("Fallbeispiel4")
3    .convertBodyTo(String.class)
4    .log("New PUT request for ${body}")
5    // extract information from input message and create input structure for createWorkflowInstance Service
6    .setHeader("tc.WF.start", wf.xpath("tns:TcWorkflow/@startImmediately", String.class))
7    .setHeader("tc.WF.name", wf.xpath("tns:TcWorkflow/@name", String.class))
8    .setHeader("tc.WF.subject", wf.xpath("tns:TcWorkflow/@subject", String.class))
9    .setHeader("tc.WF.description", wf.xpath("tns:TcWorkflow/@description", String.class))
10   .setHeader("tc.WF.template", wf.xpath("tns:TcWorkflow/@template", String.class))
11   .split().xpath("//tns:attachment", wf).aggregationStrategy(new AggregationStrategy()
12   { // new inline AggregationsStrategy
13     @Override
14     public Exchange aggregate(Exchange oldExchange, Exchange newExchange) {
15       // the first time we aggregate we only have the new exchange, so we just return it
16       if (oldExchange == null) return newExchange;
17       String oldNames = oldExchange.getIn().getHeader("tc.WF.attachments", String.class);
18       String oldValues = oldExchange.getIn().getHeader("tc.WF.attachmenttypes", String.class);
19       String newNames = newExchange.getIn().getHeader("tc.WF.attachments", String.class);
20       String newValues = newExchange.getIn().getHeader("tc.WF.attachmenttypes", String.class);
21       if (oldNames.matches("^$"))
22       {
23         // no old values available just copy the name and value to the oldExchange
24         oldExchange.getIn().setHeader("tc.WF.attachments", newNames);
25         oldExchange.getIn().setHeader("tc.WF.attachmenttypes", newValues);
26       }
27       else
28       {
29         // old values available append "," and the new name and value
30         oldExchange.getIn().setHeader("tc.WF.attachments", oldNames+","+newNames);
31         oldExchange.getIn().setHeader("tc.WF.attachmenttypes", oldValues+","+newValues);
```

```
32          }
33          return oldExchange;
34        }
35        })
36        // add attribute to name and value header
37        .setHeader("tc.WF.attachments", wf.xpath("//tns:attachment/@name", String.class))
38        .setHeader("tc.WF.attachmenttypes", wf.xpath("//tns:attachment/@type", String.class))
39        .end()
40        .loadBalance().roundRobin()
41        .to("seda:tc1?timeout=60000", "seda:tc2?timeout=60000");
42
43  from("seda:tc1").routeId("Fallbeispiel4_tc1")
44    .to("tc://192.168.0.176:8080/tc?tcMethod=createWorkflowInstance&tcUser=acam&tcPass=acam&tcSessionKey=1")
45    .convertBodyTo(at.acam.tc.model.InstanceInfo.class);
46
47  from("seda:tc2").routeId("Fallbeispiel4_tc2")
48    .to("tc://192.168.0.176:8080/tc?tcMethod=createWorkflowInstance&tcUser=acam&tcPass=acam&tcSessionKey=2")
49    .convertBodyTo(at.acam.tc.model.InstanceInfo.class);
```

Die Übernahme eines XML Dokuments mit den nötigen Information zur Anlage eine Workflows wird so wie in den anderen Beispielen mit einem JETTY-Consumer durchgeführt. Anschließend werden die Informationen ähnlich wie im Fallbeispiel 3 eingelesen (siehe Zeile 1 bis 39 in Listing 30). In Zeile 40 wird das *Load-Balancer*-Pattern definiert, gefolgt von einer geeigneten *Load-Balancing-Policy*. Die Policy definiert nach welchen Regeln die Nachrichten aufgeteilt werden. Die *Round-Robin-Policy* verteilt die Nachrichten gleichmäßig an die nachfolgenden Endpunkte. In diesem Beispiel sind das zwei SEDA Endpunkte (Zeile 40).

Die Camel SEDA Komponente bietet asynchrones SEDA Verhalten. SEDA steht für *staged event-driven architecture*[18]. Nachrichten die an einen SEDA Endpunkt übergeben werden, werden von einer *Blocking-Queue* übernommen. Von dort konsumiert der SEDA-Endpunkt die Nachricht und bearbeitet sie in einem eigenen Thread. Je nach Einstellung, werden die Nachrichten der Reihe nach einzeln oder auch paralell bearbeitet und das Ergebnis an die aufrufende Route zurückgemeldet.

6.5. Fallbeispiel 5 - Persistenz mit JMS

In Unternehmensanwendungen ist Persistenz eine wichtige Anforderung. Treten unerwartete Fehler während der Bearbeitung einer Anfrage auf und führen zu einem Absturz der Anwendung, sind alle nicht persitierten Informationen verloren. Auch muss es möglich sein, eine Anwendung temporär zu beenden um Wartungen vornehmen zu können. Sobald die Anwendung wieder gestartet wird, soll die Bearbeitung an der richtigen Stelle fortgeführt werden. Es gibt verschiedene Möglichkeiten Informationen zu persitieren. In diesem Fallbeispiel wird gezeigt wie JMS für diese Anforderungen genutzt werden kann.

Aufgabe des Fallbeispiels ist Dateien aus TEAMCENTER zu laden und an einen angegebenen Zielort zu kopieren. Beim Aufruf des Services werden Informationen über die Uid der gesuchten *ImanFile* gemeinsam mit dem Zielort der Datei übergeben. Alle Anfragen müssen nach der Übernahme gespeichert werden, sodass die Anwendung jederzeit

[18] http://www.eecs.harvard.edu/~mdw/proj/seda/

gestoppt werden kann und nach einem erneuten Start die Anfragen weiter bearbeitet werden.

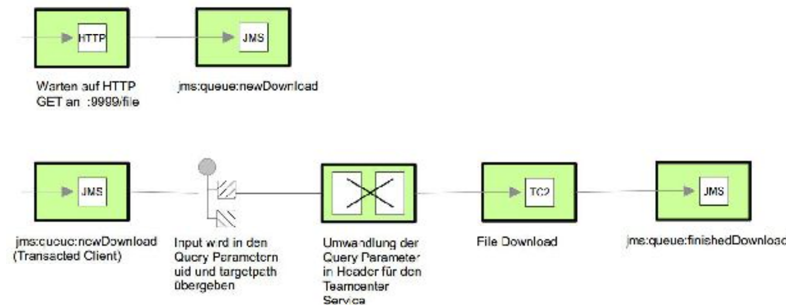

Abbildung 29: Fallbeispiel 5

Für die Umsetzung dieses Beispiels wird JMS verwendet. Alle eingehenden Anfragen zum Download einer Datei werden von einem Jetty-Consumer übernommen und an eine JMS-Warteschlage weitergeleitet. Die Anfragen werden von einem JMS-Client anschießend der Reihe nach entnommen. Besonders ist die Transaktionsfähigkeit des JMS-Clients hervorzuheben. Nur wenn die Anfrage vollständig bearbeitet wurde, wird die Nachricht endgültig aus der Queue entnommen. Bricht die Anwendung aus irgendeinem Grund während der Bearbeitung ab, betrachtet JMS die Nachricht als nicht entnommen und sie ist weiter in der Queue vorhanden.

Listing 31: Route für Fallbeispiel 5

```
1  CamelContext context = getContext();
2  ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://localhost:61616");
3  context.addComponent("jms", JmsComponent.jmsComponentAutoAcknowledge(connectionFactory));
4
5  from("jetty:http://0.0.0.0:9999/file?httpMethodRestrict=GET").routeId("Fallbeispiel5")
6    .inOnly("jms:queue:newFileDownload?requestTimeout=40000");
7
8  from("jms:queue:newFileDownload?transacted=true").routeId("Fallbeispiel5_1")
9    .process(new Processor() {
10     @Override
11     public void process(Exchange exchange) throws Exception {
12       String uid = exchange.getIn().getHeader("imanFile", String.class);
13       String dir = exchange.getIn().getHeader("targetDir", String.class);
14       String[] uids = new String[] {uid};
15       exchange.getIn().setHeader("tc.file.uids", uids);
16       exchange.getIn().setHeader("tc.file.target.dir", dir);
17     }
18   })
19   .to("tc://192.168.0.176:8080/tc?tcMethod=getFiles&tcUser=acam&tcPass=acam&tcSessionKey=1")
20   .convertBodyTo(at.acam.tc.model.GetFileResponse.class)
21   .to("jms:queue:finishedFileDownload");
```

Das Listing 31 zeigt die Umsetzung der in Abbildung 29 dargestellten Lösung. Die Umsetzung besteht aus zwei Routen. Eine Route für die Übernahme der Anfrage und eine Route für die Abarbeitung. Da die Abarbeitung der Anfrage längere Zeit dauern kann, ist sie von der Übernahme entkoppelt. Das wird durch das *InOnly* Exchange Pattern in Zeile 6 erreicht.

In Zeile 1 bis 3 in Listing 31 wird im *CamelContext* der JMS-Broker registriert, anschließend kann dieser mit „jms“ angesprochen werden.

Die zweite Route ist für den Download der Dateien aus TEAMCENTER zuständig. Über einen JMS-Consumer, der die Queue *newDownload* überwacht, werden Nachrichten aus der Queue entnommen und bearbeitet. Zu beachten ist die URI-Option *transacted=true* in Zeile 8. Diese veranlasst ein transaktionäres Verhalten der JMS-Komponente. Camel verwendet dazu automatisch den Transaction-Manager von SPRING. Erst wenn alle Bearbeitungsschritte abgeschlossen sind, ist auch die Transaktion abgeschlossen. Wird die Bearbeitung unterbrochen, ist die Transaktion nicht abgeschlossen und die Nachricht wird vom JMS-Broker weiterhin in der *newDownload*-Queue aufbewahrt. Aus dieser Queue kann die Nachricht später erneut konsumiert werden.

Der eigentliche File-Download wird in Zeile 19 mit der Teamcenterkomponente und der URI Option *tcMethod=getFiles* durchgeführt. Das Ergebnis wird in ein XML Dokument umgewandelt und abschließend wiederum in einer JMS Queue gespeichert. Tritt eine Exception auf wird versucht die Bearbeitung mehrfach zu wiederholen. Wird auch nach mehrfacher Wiederhohlung kein Erfolg erreicht, wird die Nachricht automatisch mit Hilfe der JMS Komponente an eine Error-Queue übergeben.

6.6. Fallbeispiel 6 - Exceptions and Error Handling

In allen Anwendungen ist das Fehlerhandling eine wichtige Komponente. Treten während der Ausführung eines Teamcenterservices Fehler auf, werden diese als Teil der Antwort des Services zurückgeliefert. In diesem Beispiel wird gezeigt, wie aufgetretene Fehler in einer Camel-Route verarbeitet werden können. In diesem Fallbeispiel soll eine Email mit der Fehlermeldung versendet werden und die originale Anfrage an eine spezielle JMS Queue übergeben werden falls ein Fehler während der Bearbeitung in TEAMCENTER auftritt.

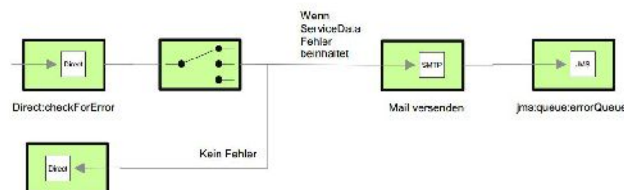

Abbildung 30: Fallbeispiel 6

In dieser Route wird ein *Direct*-Endpunkt von APACHE CAMEL als Producer verwendet. Die empfangene Nachricht wird auf Fehler in einem *ServiceData*-Objekt geprüft. Falls ein Fehler gefunden wird, wird eine Mail mit der Nachricht versendet und die ursprüngliche Nachricht an eine JMS Queue übergeben. Wird kein Fehler gefunden wird die Route beendet.

Listing 32: Route für Fallbeispiel 6

```
1  /**
2  * Exception Handling
3  * must be define before the routing logic
4  */
5  onException(Exception.class).maximumRedeliveries(0)
6    .handled(true)
7    .process(new Processor() {
8        public void process(Exchange exchange) throws Exception {
9          Exception exception = (Exception)exchange.getProperty(Exchange.EXCEPTION_CAUGHT);
10         exchange.getIn().setBody(exception.getMessage());
11         // Set header with subject
12         exchange.getIn().setHeader("Subject", "An error occured");
13     }})
14   // Send mail
15   .to("smtp://MailServerAdress?to=recipient@acam.at&from=sender@acam.at")
16   // Set original message
17   .useOriginalMessage()
18   .to("jms:queue:errorQueue");
19
20 from("direct:Fallbeispiel6").routeId("Fallbeispiel6")
21   .process(new Processor() {
22     @Override
23     public void process(Exchange exchange) throws Exception {
24       ServiceData sData = exchange.getIn().getBody(ServiceData.class);
25       if(sData.sizeOfPartialErrors()>0)
26       {
27         String errorMsg = "";
28         for (int i=0; i<sData.sizeOfPartialErrors(); i++)
29         {
30           // Each partial error has one or more associated error messages
31           for (ErrorValue errorValue : sData.getPartialError(i).getErrorValues())
32           {
33             errorMsg += "Code: "+errorValue.getCode()+" ErrorLevel:  "+errorValue.getLevel()+" Message: "+
                   errorValue.getMessage()+"\n";
34           }
35         }
36         // Throw a new exception with the error messages
37         throw new Exception(errorMsg);
38       }
39     }
40   });
```

Apache Camel bietet mit der *onException* Bedingung einen Mechanismus, um auftretende Ausnahmen in einer Route abfangen zu können. Die *onException*-Bedingung muss vor den Routen definiert werden. In Zeile 5 in Listing 32 wird eine *onException*-Bedingung für die Klasse *Exception* definiert. Sobald eine entsprechende *Exception* geworfen wird, wird in die *onException*-Bedingung gesprungen und diese abgearbeitet. In diesem Fallbeispiel wird im Falle einer *Exception* über eine SMTP-Komponente eine Mail mit der Fehlernachricht versendet (siehe Zeile 15) und abschließend die ursprüngliche Nachricht an die JMS-Queue *errorQueue* übergeben.

Im Zuge der Abarbeitung einer Serviceanfrage, befüllen die Teamcenterservices das *ServiceData*-Objekt mit allen Fehlern die während einer Anfrage aufgetreten sind. Wobei jeder aufgetretene *PartialError* aus einer oder mehreren Fehlermeldungen bestehen kann. In diesem Fallbeispiel ist das *ServiceData*-Objekt im Körper der Nachricht enthalten. Enthält das *ServiceData* Objekt einen oder mehrere *PartialErrors* (Zeile 25), werden diese und die zugehörigen Fehlermeldungen zu einer Nachricht zusammenfasst (siehe Zeile 27 bis 35). Mit dieser Fehlermeldung wird anschließend eine Exception geworfen, die von der *onException*-Bedingung abgefangen wird.

7. Fazit

Aktuell existiert eine Vielzahl an Anwendungen und Frameworks, die bei den Herausforderungen der Enterprise Application Integration unterstützen können. In Kapitel 2 werden einige Lösungen aufgelistet. Die meisten Frameworks bieten einen ähnlichen Umfang an Funktionen und so gut wie alle Lösungen implementieren die Enterprise Integration Pattern. Die Auswahl der Lösung ist von der Verfügbarkeit einzelner Komponenten und vom Umfang der benötigten Lösung abhängig.

Der Einsatz von Apache Camel hat sich für die Anforderung der Integration von Teamcenter als geeignet erwiesen. Das kann anhand mehrerer Punkte begründet werden. Zum Einen ermöglichen die angebotenen DSLs mit verschiedenen Programmiersprachen zu arbeiten. So können die Vorteile der jeweiligen Sprache genutzt werden. Zum Anderen bietet Apache Camel eine große Menge an bestehenden Komponenten. So können Lösungen für viele typische Anforderungen schnell und komfortabel erstellt werden. Ist eine Lösung mit Hilfe der EIPs beschrieben, so wie in den Fallbeispielen in Kapitel 6, kann diese einfach mit Hilfe der DSLs in eine Camel-Route übertragen werden. Das vereinfacht nicht nur die Erstellung der Routen, sondern verbessert auch die Lesbarkeit der Lösung. Ein entscheidender Punkt, der den Einsatz von Apache Camel in Integrationsprojekten so erfolgreich macht, ist die Quelloffenheit und die große Community. Beides reduziert die Einstiegsbarriere und erhöht die Anwendbarkeit erheblich. Die Möglichkeit Java Beans in Routen einsetzten zu können ermöglicht existierende Klassen und Methoden in Camel-Routen zu integrieren und so wiederzuverwenden. So können diese Beans von Entwicklern, auch ohne Kenntnisse von Apache Camel, gepflegt und verwaltet werden. Mit dem umfangreichen *TypeConverter* System lassen sich komfortabel Umwandlungen in unterschiedliche Datenformate durchführen. Auch an dieser Stelle können einfache Java Beans und somit existierende und einfach zu wartende Klassen verwendet werden.

Im Weiteren konnte durch die Erweiterbarkeit des Frameworks, wie in Kapitel 5.2 beschrieben, eine neue Teamcenterkomponente für Apache Camel erzeugt werden. Die Teamcenterkomponente ermöglicht die Wiederverwendbarkeit von implementierten Funktionen in unterschieldichen Integrationsprojekten. Dazu bietet sie eine API für Teamcenterservices die innerhalb der Camel-Routen verwendet werden kann. So können Teamcenterservices innerhalb von Camel-Routen definiert werden und die Teamcenterclientimplementierung muss nicht für jedes Integrationsprojekt neu erstellt werden. Der Umfang der API ist eine Menge von Funktionen, mit welchen die grundlegenden Aufgaben eines Datenaustauschs zwischen Teamcenter und anderen Anwendungen ausgeführt werden können. Dazu gehören neben den Servicefunktionen auch die Verwaltung der Verbindungen zu Teamcenter. Die Teamcenterkomponente bietet Möglichkeiten zu steuern, wie oft neue Verbindungen zu Teamcenter aufgebaut werden. So kann der Zugriff auf den Teamcenterserver geregelt werden und eine Überlastung des Servers durch

eine zu hohe Anzahl von Anfragen verhindert werden.

Das Teamcenterclientframework bietet umfangreiche Möglichkeiten, Geschäftsfunktionen in Teamcenter durchzuführen. Jedoch stellt es gleichzeitig eine hohe Einstiegsbarriere dar. Zwar ist die Dokumentation der Services in den letzten Versionen von Teamcenter stetig verbessert worden. Trotzdem ist sie nicht vollständig oder fehlerfrei. Als ein Beispiel soll die teilweise unvollständige Dokumentation von Enumerationen angeführt werden, welche die Erstellung zugehöriger Eingabestrukturen erschwert, wodurch gewisse Services nicht ohne weiteres ausgeführt werden können. In diesen Fällen hilft nur der Trial and Error Ansatz bis das gewünschte Ergebnis erreicht wird. Auch gibt es nur eine begrenzte Anzahl von Anwendungsbeispielen und eine kaum exisitierende Community. Diese Tatsache begründet sich darauf, dass ein entsprechendes Know How und Erfahrung nur sehr aufwendig und kostspielig aufgebaut werden kann. Unternehmen nutzen es in Folge als Alleinstellungsmerkmal, um sich vom Mitbewerb abheben zu können und teilen dieses Wissen dementsprechend ungern.

In Form von Fallbeispielen wurden typische Teilaufgaben von Integrationsprojekten dargestellt und gezeigt, wie einfache Lösungen auf Basis erprobter Komponenten erstellt werden können. Diese Lösungen können beliebig kombiniert und erweitert werden. So können auch komplexe Anforderungen in großen Integrationsprojekten erfüllt werden.

7.1. Erstellen einer Integrationslösung

Apache Camel bietet einen flexiblen Weg, Integrationslösungen zu erstellen und zu betreiben. Die Teamcenterkomponente für Camel ermöglicht zusätzlich eine Einbindung von Teamcenter in eine Integrationslösung. Um jedoch eine lauffähige Anwendung für eine Integrationslösung erstellen zu können, ist ein grundlegendes Verständnis für die EIP notwendig, zusätzlich werden Kenntnisse im Bereich der Java Programmierung und über das Apache Camel Framework benötigt. Soll die Integrationslösung später in einem OSGI Container, oder mit Spring auf einem Webserver betrieben werden, benötigt man auch ein Verständnis für diese Technologien.

Grundsätzlich kann eine Integrationslösung immer nach dem gleichen Schema erstellt werden. Begonnen wird mit der Erhebung der Anforderungen. Dazu können herkömmliche Methoden aus dem Bereich der Anforderungsanalyse genutzt werden. Auf Basis der Anforderungen wird anschließend der Umfang und die Funktion der Integrationslösung mit Hilfe der Enterprise Integration Patterns spezifiziert. Im nächsten Schritt wird ein Java-Projekt angelegt und das Grundgerüst der Camel Anwendung erzeugt. Damit bei der Routendefinition alle benötigten Komponenten verfügbar sind, müssen die erforderlichen Abhängigkeiten im Projekt angegeben werden. Build-Management-Tools können bei der Anlage des Projekts und der Verwaltung der Abhängigkeiten helfen, sind aber nicht zwingend erforderlich. Ist das Java-Projekt angelegt, kann mit der Erstellung der Routen begonnen werden. Dabei wird mit einer der verfügbaren DSLs die EIP Lösung

in Camel-Routen übersetzt. Abschließend muss entschieden werden, auf welche Art die Anwendung betrieben werden soll. Typisch ist der Betrieb als eigenständige Javaanwendung, als Webanwendung in einem Webcontainer oder in einem OSGI Container.

Wie bei jeder Entwicklung ist der Aufwand der Entwicklung von den Anforderungen abhängig. Es hat sich jedoch gezeigt, dass der Einsatz des Apache Camel Frameworks den Entwicklungsaufwand stark reduziert. Die Gründe dafür sind vor allem die Verfügbarkeit von vielen unterschiedlichen Komponenten, aber auch die Implementierung der EIP in Camel. So ist es möglich, fertig konzipierte Integrationslösungen schnell in Camel-Routen übersetzten zu können.

7.2. Ausblick

Als eine Weiterentwicklung zu der existierenden Teamcenterkomponente kann die Implementierung eines Consumers angestrebt werden. Der Consumer kann TEAMCENTER auf gewisse Events untersuchen. Dazu gehören unter anderem Aktionen ausgewählter Benutzer oder die Anlage ausgewählter Objekttypen. Auch die Überwachung von existierenden Objekten kann von einem solchen Consumer übernommen werden. In der aktuellen Version der Komponente können diese Aktionen nur über Umwege mit Hilfe von Funktionen in TEAMCENTER ausgeführt werden.

Eine weitere vielversprechende und nützliche Weiterentwicklung ist die Implementierung eines Monitorings- und Systemüberwachungs- beziehungsweise Steuerungssystems. APACHE CAMEL bietet mit JMX eine breitgefächerte Menge an Überwachungs- und Steuerungsfunktionen. Diese können über eine entsprechende Implementierung aufbereitet werden. Alternativ kann auch eine Web Console wie HAWTIO[19] als Basis verwendet werden. Diese bringt bereits grundlegende Visualisierungs- und Steuerungsmöglichkeiten für APACHE CAMEL mit und müsste nur um Funktionen für TEAMCENTER erweitert werden.

[19] http://hawt.io/

Literatur

Volker Arnold, Hendrik Dettmering, Torsten Engel, and Andreas Karcher. *Product Lifecycle Management beherrschen: Ein Anwenderhandbuch für den Mittelstand.* Springer, 2011. ISBN 9781466641945.

Christudas A Binildas. *Service Oriented Java Business Integration.* Packt Publishing Ltd, 1 edition, 2008. ISBN 9781847194404.

Ethan Cerami. *Web Services Essentials.* O'Reilly Media, 2002. ISBN 978-0-596-00224-4.

Scott Cranton and Jakub Korab. *Apache Camel Developer ' s Cookbook.* Packt Publishing Ltd., 2013. ISBN 9781782170303.

Rufus Credle, Jonathan Adams, Kim Clark, Yun Peng Ge, Hatcher Jeter, Joao Lopes, Samir Nasser, and Kailash Peri. *Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB.* ibm.com/redbooks, 2007. ISBN 9780738489087. URL `http://ibm.com/redbooks`.

Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Entwurfsmuster.* Addison-Wesley, 2004. ISBN 9783827321992.

Roger Heutschi. *Serviceorientierte Architektur: Architekturprinzipien und Umsetzung in die Praxis.* Springer-Verlag, 2007. ISBN 9783540673095.

Clarence Ho and Rob Harrop. *Pro Spring 3.* Apress, 1 edition, 2012. ISBN 1430241071.

Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Pearson Education, 2012. ISBN 9780133065107.

Claus Ibsen and Jonathan Anstey. *Camel in Action.* Manning Publications Co., 1 edition, 2010. ISBN 9781935182368.

Martin Kalin. *Java Web Services: Up and Running.* O'Reilly Media, 2 edition, 2013. ISBN 9781449365110.

Doug Kaye. *Loosely Coupled: The Missing Pieces of Web Services.* RDS Strategies LLC, 2003. ISBN 1881378241, 9781881378242.

Martin Keen, Amit Acharya, Susan Bishop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, and Paul Verschueren. *Patterns : Implementing an SOA Using an Enterprise Service Bus.* ibm.com/redbooks, 2004. ISBN 9780738490007. URL `http://ibm.com/redbooks`.

Martin Keen, Hong Hua Chin, Chidambaram Ganapathi, David Ghazaleh, På l Krogdahl, Wendy Neave, Mandeep Sahni, and Jacob Thorwart. *Patterns : Extended Enterprise SOA and Web Services.* ibm.com/redbooks, 2006. ISBN 9780738494036. URL `http://ibm.com/redbooks`.

Ramnivas Laddad. *AspectJ in Action*. Manning Publications Co., 2003. ISBN 978-1930110939.

W. Lam. *Enterprise Architecture and Integration: Methods, Implementation and Technologies: Methods, Implementation and Technologies*. IGI Global research collection. Information Science Reference, 2007. ISBN 9781591408895.

Zakaria Maamar, Youakim Badr, Noura Faci, and Quan Z Sheng. *Advanced Web Services*. Springer Science & Business Media, 2014. ISBN 978-1-4614-7534-7. doi: 10.1007/978-1-4614-7535-4. URL http://link.springer.com/10.1007/978-1-4614-7535-4.

Russell Miles. *AspectJ Cookbook*. O'Reilly Media, 2004. ISBN 978-0-596-00654-9.

Oracle. *BEA AquaLogic Service Bus 3.0 Documentation - Introduction*. Oracle. URL http://docs.oracle.com/cd/E13171_01/alsb/docs30/concepts/introduction.html.

R. Ramanathan. *Service-Driven Approaches to Architecture and Enterprise Integration*. Advances in Systems Analysis, Software Engineering, and High Performance Computing:. Information Science Reference, 1 edition, 2013. ISBN 9781466641945.

Arnon Rotem-Gal-Oz. *SOA Patterns*. Manning Publications, 1 edition, 2012. ISBN 9781933988269.

Guido Schmutz, Daniel Liebhart, and Peter Welkenbach. *Service-Oriented Architecture: An Integration Blueprint*. Packt Publishing Ltd., 2010. ISBN 9781849681049. URL www.packtpub.com.

Siemens PLM Software. TEAMCENTER - Answers for Industry. Technical report, Siemens Product Lifecycle Management Software Inc., 2013. URL http://www.plm.automation.siemens.com/de_de/products/teamcenter/index.shtml#lightview%26uri=tcm:73-79817%26title=TeamcenterOverview-TeamcenterBrochure-27055%26docType=.pdf.

Siemens PLM Software. *Teamcenter 10.1 Services Guide*. Siemens Product Lifecycle Management Software Inc., plm00076 edition, 2014a.

Siemens PLM Software. *Teamcenter 10.1 Getting Started with Customization*. Siemens Product Lifecycle Management Software Inc., plm00003 edition, 2014b.

Robert Thullner. *Implementing enterprise integration patterns using open source frameworks*. Masterarbeit, Technische Universität Wien, 2008. URL http://permalink.obvsg.at/AC05037468.

Yuli Vasiliev. *SOA and WS-BPEL*. Packt Publishing Ltd., 2007. ISBN 9781847192707.

Craig Walls. *Spring in Action*. Manning Publications Co., 3 edition, 2011. ISBN 9781935182351.

Willie Wheeler. *Spring in Practice*. Manning Publications Co., 1 edition, 2013. ISBN 1935182056.

A. Erweiterte Optionen für die TcComponent

Input Header		Output Header		Description
tc.GIARO.infos	GetItemAndRelated ObjectsInfo[]	tc.GIARO.result	com.teamcenter. services.loose.core. __2008__06. DataManagement. GetItemAnd RelatedObjectsInfo[]	Eigenständig, andere Parameter werden nicht berücksichtigt wenn dieser vorhanden ist
tc.GIARO.item.ids	String[]			Ergebnisstruktur für eine Liste von ItemIds. Der Input Header kann ein String aus item ids sein die durch Kommas getrennt sind
tc.GIARO.item.uids	String, String[]			Ergebnisstruktur für eine Liste von Item Uids. Der Input Header kann ein String aus uids sein die durch Kommas getrennt sind
tc.GIARO.rev.uids	String[]			Ergebnisstruktur für eine Liste von ItemRevision Uids
tc.GIARO.rev.id	String			Definiert eine spezielle Revision die zusätzlich zum Item in der Antwort mitgeliefert werden soll. Nur wirksam wenn tc.GIARO.item.ids oder tc.GIARO.item.uids vorhanden ist.
tc.GIARO.rev .processing	String			Definiert welche zu einem Item zugehörige Revisionen, in der Antwort mitgeliefert werden sollen. Erlaubte Werte: „None“ (Defautl), „All“, „Ids“. Für die Inputs aus tc.GIARO.rev.uids wird automatisch „Ids“ verwendet
tc.GIARO.dataset. processing	String			Definiert welche zu einerm Item/ItemRevision zugehörige Datasets, in der Antwort mitgeliefert werden sollen. Erlaubte Werte: „None“ (Default), „All“, „Min“. Für die Inputs aus
tc.GIARO.dataset.datasettypename	String[]			Liste von Datasettypnamen, definiert Datasets die falls vorhanden zusätzlich in der Antwort mitgeliefert werden.Nur wirksam wenn tc.GIARO.dataset. processing=“All“
tc.GIARO.dataset.relationtypename	String[]			Liste von Relationen, definiert die Datasets, die mit einer bestimmten Relation mit einem Item/ItemRevision verknüpft sind, zusätzlich in der Antwort mitgeliefert werden.Nur wirksam wenn tc.GIARO.dataset. processing=“All“

Tabelle 7: Erweiterte Optionen für TcComponent - Teil 1

Input Header		Output Header		Description
tc.LO.uids	String[]	tc.LO.result	com.teamcenter.soa. client.model. ServiceData	Liste von Uids für Objekte die aus Teamcenter geladen werden sollen. Ergebnisse werden als Plain Objects im ServiceData Objekt zurückgegeben
tc.SP.propinfo	HashMap<String, HashMap<String, String> >	tc.SP.result	com.teamcenter.services. loose.core._2010_09. DataManagement. SetPropertyResponse	Eine HashMap<*uid*, HashMap<*Attributename*, *Attributewert*> > die zu ändernden Objekte sind über die uid definiert, alle der Attributename definiert die zu ändernde Eigenschaft, der Attributewert definiert den Zielwert
tc.REV.uid	String	tc.REV.result	com.teamcenter.services. loose.core._2008_06. DataManagement. ReviseResponse2	Uid des Objekts von dem eine neue Revision erzeugt werden soll
tc.REV.id	String			Revisions Id der neuen Revision (optional)
tc.REV.name	String			Name der neuen Revision (optional)
tc.REV.desc	String			Beschreibung der neuen Revision (optional)
tc.file.uids	String[]	tc.file.result	bei tcMethode=putFiles com.teamcenter.soa. client.model. ServiceData, bei tcMethode=getFiles com.teamcenter.soa. client.GetFileResponse	Liste von Uids der ImanFiles
tc.file.target.dir	String			Zielverzeichnis der Dateien
tc.file.names	String[]			Nur bei tcMethode=putFiles erlaubt. Pfad zur Datei die hochgeladen werden soll. Muss in der gleichen Reihenfolge wie tc.file.references definiert werden.
tc.file.references	String[]			Nur bei tcMethode=putFiles erlaubt. Name der benannten Referenz. Muss in der gleichen Reihenfolge wie tc.file.names definiert werden.

Tabelle 8: Erweiterte Optionen für TcComponent - Teil 2

Input Header		Output Header		Description
tc.CO.typename	String	tc.CO.result	com.teamcenter.services. loose.core._2008_06. DataManagement. CreateResponse	Typ des anzulegenden Objekts, verpflichtende Information
tc.CO.string.name	String, String[]			Namen der Eigenschaften vom Typ String
tc.CO.string.value	String, String[]			Werte der String Eigenschaften, nur in Kombination mit tc.CO.string.name erlaubt. Für jeden Namen muss ein Wert angegeben werden. Werte können als einzelner String angegeben werden und müssen durch ein Komma getrennt werden.
tc.CO.boolean.name	String, String[]			Namen der Eigenschaften vom Typ Boolean
tc.CO.boolean.value	String, String[]			Werte der Boolean Eigenschaften, nur in Kombination mit tc.CO.boolean.name erlaubt. Für jeden Namen muss ein Wert angegeben werden. Werte können als einzelner String angegeben werden und müssen durch ein Komma getrennt werden.
tc.CO.int.name	String, String[]			Namen der Eigenschaften vom Typ Integer
tc.CO.int.value	String, String[]			Werte der Integer Eigenschaften, nur in Kombination mit tc.CO.int.name erlaubt. Für jeden Namen muss ein Wert angegeben werden. Werte können als einzelner String angegeben werden und müssen durch ein Komma getrennt werden.
tc.CO.double.name	String, String[]			Namen der Eigenschaften vom Typ Double
tc.CO.double.value	String, String[]			Werte der Double Eigenschaften, nur in Kombination mit tc.CO.double.name erlaubt. Für jeden Namen muss ein Wert angegeben werden. Werte können als einzelner String angegeben werden und müssen durch ein Komma getrennt werden.
tc.QUERY.name	String	tc.QUERY.result	com.teamcenter.services. loose.query._2007_09. SavedQuery. SavedQueriesResponse	Name der Query in Teamcenter
tc.QUERY.entries	String[]			Liste von Querykriterien
tc.QUERY.values	String[]			Eine Liste von String Werten mit den Suchwerten zu den Querykriterien. Die Werte müssen in der gleichen Reihenfolge wie in tc.QUERY.entries eingetragen werden.

Tabelle 9: Erweiterte Optionen für TcComponent - Teil 3

Input Header		Output Header		Description
tc.BOM.window.create	boolean	tc.BOM.window.result	com.teamcenter.services.loose. cad._2007_01.StructureManagement .CreateBOMWindowsResponse	Anlegen eines neuen Stücklistenfensters
tc.BOM.window.bomview	String			Zu öffnende Stücklistenansicht
tc.BOM.window.item	String			Item mit der zu öffnenden Stücklistenansicht.
tc.BOM.window.itemrev	String			Item Revision mit der zu öffnenden Stücklistenansicht
tc.BOM.window.revrule	String			Optional, Revisionsregeln die auf das Stücklistenfenster angewendet werden soll, wird keine angegeben wird die Default Regel verwendet
tc.BOM.expand	String	tc.BOM.expand.all.result oder tc.BOM.expand.one.result und im Körper der Eingabenachricht	com.teamcenter.services. loose.cad._2008_06. StructureManagement. ExpandPSAllLevelsResponse2 oder com.teamcenter.services. loose.cad._2008_06. StructureManagement. ExpandPSOneLevelResponse2	Operation zum expandieren einer Stücklistenzeile. Erlaubte Werte „one“, „all“
tc.BOM.expand.parent	String			Optional, Uid der zu expandierenden Stücklistenzeile, wird hier nichts angegeben, wird automatisch die Topzeile des erzeugten Stücklistenfensters verwendet. Wurde kein Stücklistenfenster erzeugt schlägt die Operation fehlt
tc.BOM.window.close	String			Optional, Uid des zu schließenden Stücklistenfenster
tc.BOM.window.close.after	Boolean			Optional, Default = true, Schließt automatisch das Stücklistenfenster am Ende der Operation
tc.WF.template	String	tc.WF.result und im Körper der Eingabenachricht	com.teamcenter.services. loose.workflow._2008_06. Workflow.InstanceInfo	Verpflichtend, Name der Workflowvorlage von der ein neuer Workflow erzeugt werden soll
tc.WF.name	String			Name des neuen Worflowprozesses
tc.WF.subject	String			Betreff des neuen Workflows
tc.WF.description	String			Beschreibung des neuen Workflows
tc.WF.attachments	String			Uids der zu verknüpfenden Objekte
tc.WF.attachmenttypes	String			Nur in Verbindung mit tc.WF.attachments berücksichtigt. Definiert die Art der Verknüpfung. Erlaubte Werte sind EPM_target_attachment oder EPM_reference_attachment

Tabelle 10: Erweiterte Optionen für TcComponent - Teil 4

B. Weiterführende Listings für TcComponent

Listing 33: TcCredentialManager

```
1  package at.acam.camel.component.tc.internal;
2  import org.slf4j.Logger;
3  import org.slf4j.LoggerFactory;
4  import com.teamcenter.schemas.soa._2006_03.exceptions.InvalidCredentialsException;
5  import com.teamcenter.schemas.soa._2006_03.exceptions.InvalidUserException;
6  import com.teamcenter.soa.client.CredentialManager;
7  import com.teamcenter.soa.exceptions.CanceledOperationException;
8  /**
9  * The CredentialManager is used by the Teamcenter Services framework to get the
10 * user's credentials when challanged by the server. This can occur after a period
11 * of inactivity and the server has timed-out the user's session, at which time
12 * the client application will need to re-authenitcate. The framework will
13 * call one of the getCredentials methods (depending on circumstances) and will
14 * send the SessionService.login service request. Upon successfull completion of
15 * the login service request. The last service request (one that caused the challange)
16 * will be resent.
17 *
18 * The framework will also call the setUserPassword setGroupRole methods when ever
19 * these credentials change, thus allowing this implementation of the CredentialManager
20 * to cache these values so prompting of the user is not requried for re-authentication.
21 */
22 public class TcCredentialManager implements CredentialManager {
23         final static Logger logger = LoggerFactory.getLogger(TcCredentialManager.class);
24     private String name          = null;
25     private String password      = null;
26     private String group         = "";
27         // default group
28         private String role          = "";
29         // default role
30         private String discriminator = "";
31         // always connect same user
32         // to same instance of server
33     /**
34         * Return the type of credentials this implementation provides,
35         * standard (user/password) or Single-Sign-On. In this case
36         * Standard credentials are returned.
37         *
38         * @see com.teamcenter.soa.client.CredentialManager#getCredentialType()
39         */
40         public int getCredentialType()      {
41                 return CredentialManager.CLIENT_CREDENTIAL_TYPE_STD;
42         }
43
44         /**
45         * Prompt's the user for credentials.
46         * This method will only be called by the framework when a login attempt has
47         * failed, because of InvalidCredentialsException. No retry will be made, an exception will be thrown
48         *
49         * @see com.teamcenter.soa.client.CredentialManager#getCredentials(com.teamcenter.schemas.soa._2006_03
               .exceptions.InvalidCredentialsException)      */
50         public String[] getCredentials(InvalidCredentialsException e)      throws CanceledOperationException
                 {
51                 logger.error(e.getMessage());
52                 try {
53                         throw new Exception("Invalid Credentials, establishing a connection to teamcenter
                              failed");
54                 } catch (Exception e1) {
55                         // TODO Auto-generated catch block
56                         e1.printStackTrace();
57                 }
58                 return null;
59         }
60
61         /**
62         * Return the cached credentials.
63         * This method will be called when a service request is sent without a valid
64         * session ( session has expired on the server).
65         *
66         * @see com.teamcenter.soa.client.CredentialManager#getCredentials(com.teamcenter.schemas.soa._2006_03
               .exceptions.InvalidUserException)
```

```
67          */      public String[] getCredentials(InvalidUserException e)
68          {       String[] tokens = { name, password, group, role, discriminator };
69          return tokens;
70              }
71          /**
72          * Load user name and pass
73          * @return
74          * @throws CanceledOperationException
75          */
76          public String[] getCredentials()
77          {
78                  String[] tokens = { name, password, group, role, discriminator };
79                  return tokens;
80          }
81
82          /**
83          * Cache the group and role
84          * This is called after the SessionService.setSessionGroupMember service
85          * operation is called.
86          *
87          * @see com.teamcenter.soa.client.CredentialManager#setGroupRole(java.lang.String,
88          *      java.lang.String)
89          */
90          public void setGroupRole(String group, String role)
91          {
92                  if (group!=null){
93                          this.group = group;
94                  }
95                  if(role!=null){
96                          this.role = role;
97                  }
98          }
99
100         /**
101         * Cache the User and Password
102         * This is called after the SessionService.login service operation is called.
103         *
104         * @see com.teamcenter.soa.client.CredentialManager#setUserPassword(java.lang.String,
105         *      java.lang.String, java.lang.String)
106         */
107         public void setUserPassword(String user, String password, String discriminator){
108                 this.name = user;
109                 this.password = password;
110                 this.discriminator = discriminator;
111         }
112 }
```

Listing 34: Java Klasse erzeugt durch xjc aus einer Teamcenter Schema Definition

```
1  import java.util.ArrayList;
2  import java.util.List;
3  import javax.xml.bind.annotation.XmlAccessType;
4  import javax.xml.bind.annotation.XmlAccessorType;
5  import javax.xml.bind.annotation.XmlElement;
6  import javax.xml.bind.annotation.XmlRootElement;
7  import javax.xml.bind.annotation.XmlType;
8  /**
9  *Return structure for GetItemAndRelatedObjects operation
10 * <p>Java class for anonymous complex type.
11 *
12 * <p>The following schema fragment specifies the expected content contained within this class.
13 *
14 * <pre>
15 * &lt;complexType>
16 *   &lt;complexContent>
17 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
18 *       &lt;sequence>
19 *         &lt;element name="output" type="{http://teamcenter.com/Schemas/Core/2008-06/DataManagement}
       GetItemAndRelatedObjectsItemOutput" maxOccurs="unbounded" minOccurs="0"/>
20 *         &lt;element ref="{http://teamcenter.com/Schemas/Soa/2006-03/Base}ServiceData"/>
21 *       &lt;/sequence>
22 *     &lt;/restriction>
23 *   &lt;/complexContent>
24 * &lt;/complexType>
25 * </pre>
```

```
26  */
27  @XmlAccessorType(XmlAccessType.FIELD)
28  @XmlType(name = "", propOrder = {      "output",      "serviceData" })
29  @XmlRootElement(name = "GetItemAndRelatedObjectsResponse")
30  public class GetItemAndRelatedObjectsResponse {
31          protected List<GetItemAndRelatedObjectsItemOutput> output;
32          @XmlElement(name = "ServiceData", namespace = "http://teamcenter.com/Schemas/Soa/2006-03/Base",
                required = true)
33          protected ServiceData serviceData;
34
35  /**
36  * Gets the value of the output property.
37  *
38  * <p>
39  * This accessor method returns a reference to the live list,
40  * not a snapshot. Therefore any modification you make to the
41  * returned list will be present inside the JAXB object.
42  * This is why there is not a <CODE>set</CODE> method for the output property.
43  *
44  * <p>
45  * For example, to add a new item, do as follows:
46  * <pre>
47  *    getOutput().add(newItem);
48  * </pre>
49  * <p>
50  * Objects of the following type(s) are allowed in the list
51  * {@link GetItemAndRelatedObjectsItemOutput }
52  */
53          public List<GetItemAndRelatedObjectsItemOutput> getOutput() {
54          if (output == null) {
55              output = new ArrayList<GetItemAndRelatedObjectsItemOutput>();
56                  }
57                  return this.output;
58          }
59
60  /**
61  *Standard ServiceData member
62  * @return
63  *     possible object is
64  *     {@link ServiceData }
65  */
66          public ServiceData getServiceData() {
67          return serviceData;
68          }
69
70  /**
71  * Sets the value of the serviceData property.
72  *
73  * @param value
74  *     allowed object is
75  *     {@link ServiceData }
76  *
77  */
78          public void setServiceData(ServiceData value) {
79          this.serviceData = value;
80          }
81  }
```

Listing 35: MyFirstTeamcenterIntegration/pom.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
        schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
3     <modelVersion>4.0.0</modelVersion>
4     <groupId>at.acam</groupId>
5     <artifactId>MyFirstTeamcenterIntegration</artifactId>
6     <packaging>jar</packaging>
7     <version>1.0-SNAPSHOT</version>
8     <name>A Camel Route</name>
9     <url>http://www.myorganization.org</url>
10    <properties>
11      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
12      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
13    </properties>
14    <dependencies>
15          <!-- ACAM -->
```

```
16          <dependency>
17                  <groupId>at.acam.camel.component</groupId>
18                  <artifactId>tc</artifactId>
19                  <version>0.1.0</version>
20          </dependency>
21          <dependency>
22                  <groupId>at.acam.camel.tc</groupId>
23                  <artifactId>xmladapter</artifactId>
24                  <version>0.0.2</version>
25          </dependency>
26          <!-- Camel -->
27          <dependency>
28                  <groupId>org.apache.camel</groupId>
29                  <artifactId>camel-core</artifactId>
30                  <version>2.15.0</version>
31          </dependency>
32          <dependency>
33                  <groupId>org.apache.camel</groupId>
34                  <artifactId>camel-jaxb</artifactId>
35                  <version>2.14.0</version>
36          </dependency>
37          <!-- logging -->
38          <dependency>
39                  <groupId>org.slf4j</groupId>
40                  <artifactId>slf4j-api</artifactId>
41                  <version>1.7.10</version>
42          </dependency>
43          <dependency>
44                  <groupId>org.slf4j</groupId>
45                  <artifactId>slf4j-log4j12</artifactId>
46                  <version>1.7.10</version>
47          </dependency>
48          <dependency>
49                  <groupId>log4j</groupId>
50                  <artifactId>log4j</artifactId>
51                  <version>1.2.17</version>
52          </dependency>
53          <!-- testing -->
54          <dependency>
55                  <groupId>org.apache.camel</groupId>
56                  <artifactId>camel-test</artifactId>
57                  <version>2.15.0</version>
58                  <scope>test</scope>
59          </dependency>
60    </dependencies>
61    <build>
62      <defaultGoal>install</defaultGoal>
63      <plugins>
64        <plugin>
65          <groupId>org.apache.maven.plugins</groupId>
66          <artifactId>maven-compiler-plugin</artifactId>
67          <version>2.5.1</version>
68          <configuration>
69            <source>1.7</source>
70            <target>1.7</target>
71          </configuration>
72        </plugin>
73        <plugin>
74          <groupId>org.apache.maven.plugins</groupId>
75          <artifactId>maven-resources-plugin</artifactId>
76          <version>2.6</version>
77          <configuration>
78            <encoding>UTF-8</encoding>
79          </configuration>
80        </plugin>
81        <!-- Allows the example to be run via 'mvn compile exec:java' -->
82        <plugin>
83          <groupId>org.codehaus.mojo</groupId>
84          <artifactId>exec-maven-plugin</artifactId>
85          <version>1.2.1</version>
86          <configuration>
87              <mainClass>at.acam.MainApp</mainClass>
88              <includePluginDependencies>false</includePluginDependencies>
89          </configuration>
90        </plugin>
91      </plugins>
92    </build>
```

```
93 </project>
```

C. Weiterführende Dokumente zu den Fallbeispielen

Listing 36: TcObjectSchema.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <schema targetNamespace="http://www.acam.at/Object" elementFormDefault="qualified" xmlns="http://www.w3.org
       /2001/XMLSchema" xmlns:tns="http://www.acam.at/Object">
3      <element name="Object" type="tns:Object"></element>
4      <complexType name="Object">
5          <sequence>
6              <element name="attribute" type="tns:attribute"
7                  maxOccurs="unbounded" minOccurs="1">
8              </element>
9          </sequence>
10         <attribute name="item_id" type="string"></attribute>
11         <attribute name="rev_id" type="string"></attribute>
12         <attribute name="type" type="string"></attribute>
13     </complexType>
14     <complexType name="attribute">
15         <attribute name="name" type="string"></attribute>
16         <attribute name="value" type="string"></attribute>
17     </complexType>
18 </schema>
```

Listing 37: TcWorkflowSchema.xsd

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <schema targetNamespace="http://www.acam.at/TcWorkflow" elementFormDefault="qualified" xmlns="http://www.w3.
       org/2001/XMLSchema" xmlns:tns="http://www.acam.at/TcWorkflow">
3      <element name="TcWorkflow" type="tns:TcWorkflow"></element>
4      <complexType name="TcWorkflow">
5          <sequence>
6              <element name="attachment" type="tns:attachment"
7                  maxOccurs="unbounded" minOccurs="0">
8              </element>
9          </sequence>
10         <attribute name="template" type="string" use="required"></attribute>
11         <attribute name="name" type="string"></attribute>
12         <attribute name="subject" type="string"></attribute>
13         <attribute name="description" type="string"></attribute>
14         <attribute name="startImmediately" type="boolean"></attribute>
15     </complexType>
16     <complexType name="attachment">
17         <attribute name="name" type="string"></attribute>
18         <attribute name="type" type="int"></attribute>
19     </complexType>
20 </schema>
```

Listing 38: removeNamespace.xslt

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xsl:stylesheet version="1.0"
3       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4      <xsl:output method="xml" indent="yes" />
5      <xsl:template match="*">
6          <xsl:element name="{local-name(.)}">
7              <xsl:apply-templates select="@* | node()" />
8          </xsl:element>
9      </xsl:template>
10     <xsl:template match="@*">
11         <xsl:attribute name="{local-name(.)}">
12       <xsl:value-of select="." />
13     </xsl:attribute>
14     </xsl:template>
15 </xsl:stylesheet>
```