Technische Universität München
Fakultät für Informatik

Diplomarbeit in Informatik

Analyse und Bewertung der Integrationsmöglichkeiten von Content- und Dokumenten-Management-Systemen verschiedener Hersteller mittels JSR 170 API

Analysis and Evaluation of the Integration Possibilities of Content and Document Management Systems from Different Vendors Using the JSR 170 API

Ivaylo Tonev

Aufgabensteller:	Prof. Dr. Florian Matthes
Betreuer:	Dr. Thomas Büchner
	Dipl. Inf. Univ. Andreas Vogel
Abgabedatum:	15. September 2008

Erklärung

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. September 2008 Ivaylo Tonev

Zusammenfassung

Die Menge der digital erstellten und verwalteten unstrukturierten Daten wie Tabellen- und Textverarbeitungsdateien in Unternehmen wird immer größer. Damit diese zunehmende Flut von Informationen zu beherrschen ist, werden in Organisationen Informationssysteme wie Content- und Dokumenten-Management-Systeme eingesetzt. In großen Unternehmen existiert oft eine große Anzahl von derartigen Informationssystemen. Um die Konsistenz der Daten zu erhöhen und den Zugriff zu vereinfachen werden Integrationsprojekte durchgeführt.

Bei der Realisierung einer Integrationslösung spielen die von den Produkten bereitgestellten Schnittstellen für Zugriff auf die Inhalte eine entscheidende Rolle. Die Produkte verschiedener Hersteller stellen fast ausschließlich selbst entworfene, produktspezifische Schnittstellen bereit. Daraus resultiert, dass der Zugriff auf Content im Rahmen des Unternehmens nicht einheitlich ist. Die Realisierung von Integrationslösungen wird erschwert. Demgegenüber existiert mit der Spezifikation JSR 170 API eine standardisierte Schnittstelle für den Zugriff auf Content.

In Rahmen dieser Arbeit sollen die Integrationsmöglichkeiten von Content- und Dokumenten-Management-Systemen verschiedener Hersteller mittels JSR 170 API analysiert und bewertet werden. Zunächst werden die Integrationsmöglichkeiten im ECM-Bereich anhand von Integrationsszenarien identifiziert. Dies wird durch die Einführung eines formalen Modells erreicht. Danach werden die technische Realisierung von Integrationslösungen und die dabei entstehenden Herausforderungen analysiert. Die ECM-Produkte Alfresco, Day Communique und EMC Documentum werden in Bezug auf die von ihnen bereitgestellten Schnittstellen zum Zugriff auf die verwalteten Inhalte betrachtet.

Die Realisierung der identifizierten Szenarien wird diskutiert und der durch die Verwendung der JSR 170 API entstehende Nutzen wird bewertet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

A2A	Application-to-Application-Integration
API	Application Programming Interface,
B2B	Business-to-Business
BBAI	B2B Applikation Integration
CAD	Computer Aided Design
CIFS	Common Internet File System
CMS	Content-Management-System
CTO	Chief Technical Officer
DFC	Documentum Foundation Classes
DMCL	Documentum Client Libary
DMS	Dokumenten-Management-System
DQL	Documentum Query Language
EAI	Enterprise Application Integration
ECM	Enterprise-Content-Management
ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IDC	International Data Corp.
ISO/OSI	Open Systems Interconnection Basic Reference Model
IT	Information Technology
JAAS	Java Authentication and Authorization Service
JCR	Java Content Repository
JDBC	Java Database Connectivity
JDK	Java Development Kid
JMS	Java Message Service
JSR 170	Java Specification Request 170
JTA	Java Transaction API
JVM	Java Virtual Machine

ODBC	Open Database Connectivity
RDBMS	Relational Database Management System
REST	Representational State Transfer
RMI	Remote Method Invocation
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discover and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
WCMS	Web-Content-Management-System
WDK	Web Development Kit
WebDAV	Web-based Distributed Authoring and Versioning
WSDL	Web Service Description Language
XML	Extensible Markup Language

1 Einleitung

Das Umfeld von den in Unternehmen eingesetzten Informationstechnologien wird immer komplexer. Unternehmen müssen stetig ihre Strukturen und Strategien den dynamischen Markverhältnissen anpassen, um wettbewerbsfähig zu bleiben. Somit wachsen die Anforderungen an die Kooperation, Kommunikation und Koordination von Arbeitsgruppen, die auf unterschiedliche Informationsbestände zugreifen müssen [Ka99a].

Von Analysten der International Data Corp. (IDC) wird eine Zunahme der Menge an digital erstellten und gespeicherten Informationen bis zum Jahr 2010 um das Sechsfache, von 161 Exabytes (2^{30} Bytes) auf 988 Exabytes [In07] vorausgesagt.

Ein Großteil des Wissens eines Unternehmens liegt in Form von unstrukturierter elektronischer Information vor, der eine immer höhere Bedeutung beigemessen wird. Darüber hinaus hat Information nur dann einen anhaltenden Wert, wenn sie als Wissen und in Prozessen nutzbar gemacht wird [Ka03a].

Diese zunehmende Flut von meistens unstrukturierten Inhalten wie Tabellen- und Textverarbeitungsdateien wird in Organisationen mittels Informationssystemen wie Content- und Dokumenten-Management-Systeme verwaltet.

In größeren Unternehmen werden oft mehrere Systeme gleichzeitig betrieben und Inhalte sind von unterschiedlichen Quellen zu beziehen. Um die Nutzung der Inhalte einer größerer Anzahl von Mitarbeitern zu ermöglichen oder bei Ablösung von Altsystemen werden Integrationsprojekte angestoßen. Die Realisierung einer Integrationslösung erweist sich als problematisch, da die Produkte verschiedener Hersteller bisher fast ausschließlich selbst entworfene, produktspezifische Schnittstellen für den Zugriff auf die verwalteten Inhalte bereitstellen.

Da hohe Flexibilität im Umgang mit Informationstechnologien heute für viele Unternehmen wettbewerbsentscheidend ist, gewinnt die Interoperabilität der Systeme zunehmend an Bedeutung. Technologien und Standards, die die Kooperation und Integration heterogener Systeme ermöglichen, werden immer notwendiger.

Ein wichtiger Schritt in Richtung Standardisierung ist die Veröffentlichung der Spezifikation der JSR 170 API, mit deren Hilfe der Zugriff auf Informationen aus vollkommen unterschiedlichen Umgebungen auf eine einheitliche und standardisierte Art und Weise ermöglicht wird.

1.1 Ziele der Arbeit

In der vorliegenden Arbeit sollen zunächst die Gründe für Integration von ECM-Systemen in Unternehmen untersucht werden. Eine Systematisierung der Gründe soll vorgenommen werden.

Darauf aufbauend sollen Integrationsszenarien identifiziert werden, die die Integration im ECM-Bereich vollständig charakterisieren.

Die technische Realisierung von Integration und die dabei entstehenden Schwierigkeiten sollen betrachtet werden. Technologien, die die Kommunikation zwischen ECM-Systemen ermöglichen und die von aktuellen Produkten unterstützt werden, sind zu beschreiben.

Im letzten Teil der Arbeit soll die Realisierung der definierten Integrationsszenarien diskutiert werden. Insbesondere soll die Verwendung von JSR 170 bei der Umsetzung der Szenarien analysiert und bewertet werden.

1.2 Gliederung der Arbeit

Die vorliegende Arbeit ist in acht Kapiteln gegliedert.

Im Kapitel 1 und Kapitel 2 wird das thematische Feld der vorliegenden Arbeit beschrieben. Es werden Begriffe im ECM-Bereich definiert. Die allgemeine Architektur der Systeme und die grundlegenden Konzepte für Integration von Anwendungssystemen werden betrachtet. Weiterhin wird JSR 170 vorgestellt.

Im Kapitel 3 werden die Gründe für Integration in Unternehmen untersucht und in Gruppen zusammengefasst.

Im Kapitel 4 wird ein theoretisches formales Modell für Integration eingeführt. Das Modell wird anschließend auf die Gründe für Integration angewendet. Auf diese Weise werden alle möglichen Integrationsszenarien im ECM-Umfeld abgeleitet.

Im Kapitel 5 wird näher auf die technische Realisierung von Integration eingegangen. Der Begriff API wird definiert und die technischen Schwierigkeiten bei der Erstellung von Integrationslösungen werden erläutert.

Im Kapitel 6 werden einige führende ECM-Produkte vorgestellt und bezüglich der angebotenen Schnittstellen analysiert.

Im Kapitel 7 findet die Analyse der in Kapitel 4 abgeleiteten Integrationsszenarien statt. Der Nutzen bei der Verwendung von JSR 170 für die Realisierung dieser Szenarien wird bewertet.

Im Kapitel 8 werden die Ergebnisse der Arbeit zusammengefasst

2 Grundlagen

In diesem Kapitel werden grundlegende Begriffe und Konzepte, die für das Verständnis dieser Arbeit notwendig sind, erklärt.

2.1 Begriffe und Definitionen

2.1.1 Content

In der vorliegenden Arbeit ist Content[1] ein zentraler Begriff. Für Systeme, die beliebige Inhalte verwalten, hat sich der Begriff des Content etabliert.

In [Ka03b] wird Content als „Information in strukturierter, schwach strukturierter und unstruktrurierter Form, die in elektronischen Systemen zur Nutzung bereitgestellt wird“, definiert.

- Content in *strukturierter Form* sind Daten, die in einem standardisierten Layout z.B. aus Datenbankentabellen bereitgestellt werden.
- Content in *schwach strukturierter Form* sind Informationen und elektronische Dokumente wie z.B. Textverarbeitungsdateien, die zum Teil Layout und Meta-Daten mit sich Tragen, jedoch nicht standardisiert sind.
- Content in *unstrukturierter Form* sind beliebige Informationen, bei denen keine Trennung nach Dokumenteninhalt, Layout und Metadaten vorhanden ist wie z.B. Bilder und Videos.

Content besteht aus dem Dokumenteninhalt und den dazugehörigen Metadaten. Die Metadaten sind nicht immer für den Benutzer sichtbar. Sie werden für die Verarbeitung und Kontrolle des Inhalts gebraucht. In [Ka03c] werden Metadaten als „Attribute, die Dokumenten zur Indizierung und Identifizierung mitgegeben werden. Sie beinhalten Informationen über Daten, wie z. B. Herkunft, Urheber und Aktualität.“

Weiterhin wird unstrukturierter Content nach der Erfassung in einem elektronischen System mit Metadaten wie beispielsweise das Erstellungsdatum und den Titel versehen. Die Begriffe Indizierung, Indexierung und Vergabe von Metadaten werden im Zusammenhang oft synonym verwendet [ZGB05a].

Metadaten können uneingeschränkt an die Bedürfnisse der Suche angepasst werden, um das Wiederauffinden von Content zu erleichtern.

[1] Content und Inhalt werden in der vorliegenden Arbeit synonym verwendet.

2.1.2 Content-Management-Systeme

In [Ka03d] wird ein Content-Management-System als ein System, das folgende Funktionalitäten bereitstellt, definiert:

- Erstellung von Content
- Verwaltung von Content
- Bereitstellung von Content
- Kontrolle von Content
- Individualisierung von Content

Die Begriffe Content-Management, Content-Management-System und CMS werden von Anbietern und Benutzer häufig vermischt verwendet. Bei der Betrachtung des thematischen Umfelds Content-Management wird daher zwischen der übergreifenden Kategorie Content-Management-Systeme (CMS) und der speziellen Ausprägungen Web-Content-Management-System (WCMS) und Enterprise-Content-Management-Systeme (ECMS) unterschieden.

Der Begriff WCMS erweitert den Begriff CMS und umfasst die Verwaltung von Content auf Internet-basierten Webseiten und Portalen [Ka03e]. WCMS konzentriert sich somit auf die Bereitstellung von Content für offene Benutzergemeinschaften im Internet oder unternehmensinternen Netzen.

Web-Content-Management-Systeme werden eingesetzt um die unzulänglichen Möglichkeiten von HTML zur Gestaltung von Webseiten mit professionellen Tools zu überwinden. Typische Aufgaben von WCM-Systemen sind die Gestaltung von Webseiten, Verwaltung von Inhalten, datenbankgestützte Informationsbereitstellung, Personalisierung, sowie automatisierte Inhaltspublikation.

In einem Content-Management-System sind Inhalt, Struktur und Darstellung entkoppelt. Das bedeutet, dass das System keine fertigen Webseiten speichert und beim Aufruf präsentiert. Die Darstellung wird über Vorlagen unter Einbezug verschiedener Inhalte und Layouts dynamisch generiert [Bü01].

Web-Content-Management-Systeme unterstützen den Webseitenerstellungsprozess durch Werkzeuge und Technologien, die Mitarbeitern mit unterschiedlichen Kompetenzen nutzen. Beispielsweise können Redakteure unter festgelegten Arbeitsschritten Inhalte verwalten und publizieren, ohne über spezielle technische Kenntnisse für die Darstellung des Inhalts zu verfügen, währen Designer in der Lage sind, Gestaltungsvorgaben für Webseiten systemübergreifend zu definieren.

Im weiteren Verlauf der Arbeit wird jedoch nur der Begriff Content-Management-System verwendet.

2.1.3 Dokumenten-Management-Systeme

Dokumenten-Management-Systeme (DMS) sind etablierte Produktkategorien, die die Aufgabe haben, Dokumente, die die zentralen Informationsträger von Institutionen und Unternehmen sind, in elektronischer Form zu verwalten und zu verarbeiten.

Dokumenten-Management-Systeme im engeren Sinn sind ursprünglich aus der Notwendigkeit entstanden, für die Beherrschung enorm wachsender Dateibestände in Unternehmen geeignete Managementfunktionen und Dienste zur Verfügung zu stellen. Durch den Einsatz von DMS-Systemen werden Restriktionen herkömmlicher hierarchischer Ablagesysteme, wie zum Beispiel Dateisysteme, überwunden. DMS-Systeme ermöglichen datenbankgestützte Verwaltung elektronischer Dokumente, synchronisierte Bearbeitung, Versionierung, eine sichere Ablage und Bereitstellung der Information unabhängig von Autor, Ort und Zeit [Ka03f].

In [Ka99b] wird ein Dokument-Management-System im weiteren Sinn als ein Sammelbegriff für Produkte zur Erfassung, Verwaltung, Speicherung, Archivierung, Verteilung, Kontrolle und Bereitstellung von Dokumenten, mit Recherche und Prozesssteuerungsfunktionalität definiert.

Zu dieser Klassifizierung gehören neben dem Dokumenten-Management im engeren Sinn auch all jene erweiternden Funktionen, welche eine große ECM-Lösung ausmachen. Im Fokus steht jedoch immer noch die Verwaltung elektronischer Dokumente.

2.1.4 Enterprise-Content-Management-Systeme

In [Ai08] wird folgende Definition für Enterprise-Content-Management gegeben: „Enterprise-Content-Management umfasst die Technologien zur Erfassung, Verwaltung, Speicherung, Bewahrung und Bereitstellung von Content und Dokumenten zur Unterstützung von organisatorischen Prozessen.“

Die Vision von ECM-Systemen ist alle Informationen eines Unternehmens auf einer einheitlichen Plattform zur Nutzung intern, im Partnerverbund und extern bereitzustellen [Ka99c]. Traditionell eigenständige Disziplinen werden in Form einer umfassenden Infrastrukturlösung zusammengeführt, um ein zentrales Unternehmens-Repository zu erschaffen.

ECM-Komponenten und Techniken lassen sich nach [Ka07] in fünf Hauptkomponenten aufteilen: Erfassung (Capture), Verwaltung (Manage), Speicherung (Store), Bewahrung (Preserve) und Ausgabe (Deliver).

Weiterhin wird die Manage-Komponente in folgenden Disziplinen aufgeteilt:

- Documenten-Management
- Collaboration
- Web-Content-Management

- Records-Management
- Workflow

Collaboration bezeichnet Funktionen von Softwaresystemen, welche die Zusammenarbeit in einer Gruppe unterstützen. Records Management bezeichnet die reine Verwaltung von Records - wichtigen aufbewahrungspflichtigen oder aufbewahrungswürdigen Informationen. Workflow ist die prozessgesteuerte Zusammenführung und Nutzung von Informationen [Wi08a].

In der vorliegenden Arbeit wird in dieser Hinsicht der Begriff Enterprise-Content-Management-System als Oberbegriff für Content- und Documententen-Management-Systeme verwendet.

2.2 Allgemeine Architektur von ECM-Systemen

Moderne ECM-Systeme sind komplexe Softwareanwendungen, welche eine dreischichtige Architektur, wie in Abbildung 1 dargestellt verwenden.

Von den sechziger bis etwa Anfang der achtziger Jahre des letzten Jahrtausends waren die betrieblichen Anwendungssysteme durch ihre monolithische Architektur geprägt [Ma05]. Das Anwendungssystem wurde zentral auf einem Großrechner (Mainframe) installiert. Jeder Anwender hatte über ein Terminal Zugriff auf die Informationen und konnte Aktionen ausführen. Die Nachteile von monolithischen Anwendungssystemen liegen in der schweren Wartbarkeit und in den mangelnden Schnittstellen, was die Integrierbarkeit der Systeme verhindert.

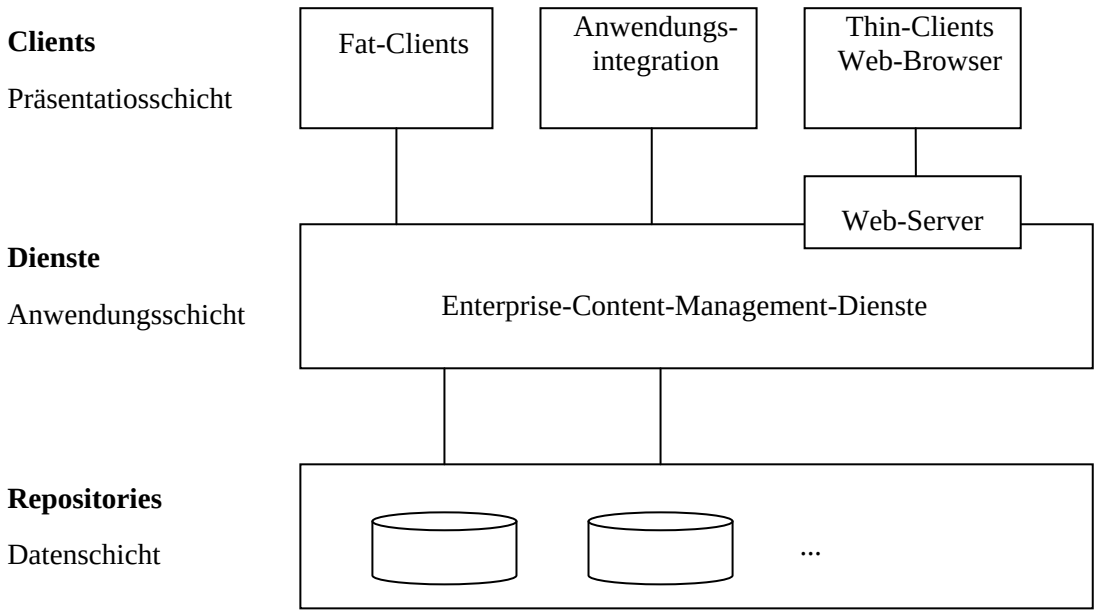


Abbildung 1: Dreischichtiges Architekturmodell [Ka99d]

Mit der Verbreitung der PCs fand die Client/Server-Architektur große Verwendung. Grundlegend für die Dezentralisierungsbestrebungen ist die Erkenntnis, dass betriebliche Anwendungssysteme prinzipiell aus drei Systemkomponenten bestehen [Du03]:

- dem Systemteil *Präsentationsschicht*, der Systemteile zur Abwicklung der Benutzerinteraktion umfasst,
- Dem Systemteil *Anwendungsschicht*, in dem die eigentliche Verarbeitungslogik kodiert ist,
- dem Systemteil *Datenschicht*, der die Systemteile zur Verwaltung der Daten beinhaltet.

Auf der *Präsentationsschicht* werden verschiedene Clients unterschieden, die so genannten Fat-Client vom Thin-Client. Als Thin-Client wird eine mit gewöhnlichen Webbrowsern bedienbare Schnittstelle zum System bezeichnet. In diesem Fall wird ein Großteil der Logik in serverseitigen Komponenten ausgeführt. Ein Fat-Client hingegen ist eine speziell entwickelte, meist eigenständige Client-Applikation, die wesentlich mehr Funktionalität lokal bereitstellt. Dabei kann es sich zum Beispiel um eine Import Schnittstelle handeln, die mit Scannern interagiert [ZGB05b]. Weitere Anwendungen, die die ECM-Dienste nutzen, werden auch in dieser Schicht eingeordnet.

Die *Anwendungsschicht* beinhaltet die Hauptlast der Verarbeitung. Hier befindet sich der Applikationsserver, auf dem die Geschäftslogik in Form von Anwendungen, die unterschiedliche Enterprise-Content-Management-Dienste bereitstellen, aufgesetzt ist. Es werden weiterhin Schnittstellen für Zugriff auf die Dienste angeboten, die anderen Systemen zur Verfügung gestellt werden.

Auf der *Datenschicht* befinden sich die Repositories, in denen der Content langfristig gespeichert ist. Repositories können weitere Systeme wie relationalen Datenbanken oder Dateisysteme beinhalten.

2.3 Systemintegration

ECM-Systeme werden selten als eigenständige Systeme, sondern als Ergänzung zu einer vorhandenen Anwendungslandschaft eingeführt [ZGB05c]. Die Architektur der Systeme sollte die Integration und Anbindung anderer Systeme ermöglichen, so dass Inhalte gemeinsam ausgetauscht und verwendet werden können [Ka99e]. Zur Erfüllung dieser Anforderungen bieten die Systeme Schnittstellen für den Remote-Zugriff an. In Kapitel 6 werden Produkte verschiedener Hersteller in Hinblick auf die angebotenen Remote-Schnittstellen analysiert.

2.3.1 Definition

Allgemein bezeichnet Integration die „Wiederherstellung des Ganzen“ [Du00] durch das Verbinden oder Vereinigen logisch zusammengehörender Teile und bestrebt sowohl das Bemühen, bisher getrennte Vorgänge oder Strukturen zusammenzuführen, als auch das Ergebnis dieser Tätigkeiten. [Ka02a]

Die Integration in der Informatik, speziell in der Softwaretechnik, dient zur Verknüpfung von verschiedenen Anwendungen[2] [Wi08b]. Dabei handelt es sich im Sinne von [Bu03] um heterogene, autonome und verteilte Systeme.

Heterogen bedeutet in diesem Zusammenhang, dass die Systeme unterschiedliche konzeptionelle Modelle für die Darstellung und Verarbeitung der Geschäftssemantik verwenden. Bei der Integration von zwei oder mehreren Anwendungssystemen ist eine Datentransformation notwendig, die die Repräsentation und Bedeutung der Daten von dem einen System auch korrekt in das andere System ohne Änderung der Semantik abbildet.

Autonom bedeutet, dass ein System seinen Zustand unabhängig von den anderen Systemen ändern kann. Daher wird von einer Integrationslösung erwertet, Zustandsänderungen zu beobachten und zu koordinieren.

Die Systeme sind auch verteilt und teilen in der Regel keine Daten und Zustände miteinander. Der Zugriff auf Funktionen oder die Übertragung von Daten von einer Anwendung auf eine andere bietet meist einen Mehrwert, da diese Daten für die Weiterverarbeitung benutzt werden können und eine effektive Informationsbereitstellung gewährleisten.

Die Eigenschaften Heterogenität, Autonomität und Verteiltheit von Systemen sind die wesentlichen Herausforderungen einer Integrationslösung.

2.3.2 Integration betrieblicher Anwendungssysteme

Die Integration betrieblicher Anwendungssystemen kann nach [Ku96] auf drei Wegen erfolgen:

- durch die vollständige Neuentwicklung eines umfassenden Anwendugssystems
- durch die Entwicklung integrationsfähiger Einzelkomponenten, die schrittweise zusammengefügt werden, oder
- durch die nachträgliche Integration vorhandener Anwendugssysteme.

Bezüglich der Intergrationsreichweite wird eine Unterscheidung zwischen der innerbetrieblichen und der zwischenbetrieblichen Integration vorgenommen [Ka02b]. Die Applikationssicht der internen Integration wird in der Literatur unter dem Stichwort „Enterprise Application Integration (EAI)“, behandelt. Das Entsprechende Pendant dazu, die Applikationsaspekte der externen Integration, wird unter dem Begriff „B2B Applikation Integration (BBAI)“ diskutiert [SWD03].

Im Gegensatz zu einer EAI-Lösung als umfassende Aufgabe im Unternehmen liegt der Fokus dieser Arbeit auf die Anwendungsintegration, die mit dem Begriff „Application-to-

[2] Die Begriffe Applikation, Anwendung, Anwendungssystem, (System, Softwaresystem) werden synonym verwendet

Application-Integration (A2AI)“ bezeichnet wird [Wi08c]. Es werden die softwaretechnische Integrationsmöglichkeiten zwischen Anwendungssysteme untersucht.

Insbesondere wird in der vorliegenden Arbeit die nachträgliche Integration vorhandener Content- und Dokumenten-Management-Systemen betrachtet.

2.3.3 Einteilung nach Integrationskonzepte

Für die Realisierung einer Integrationslösung stehen unterschiedliche, grundlegende Methoden zur Verfügung. Diese allgemeinen Methoden werden in [Ka02c] als *Integrationskonzepte* bezeichnet. Sie beschreiben die Eigenschaften und die Mechanismen der Integration und definieren die Möglichkeiten für Integration von Anwendungen.

In [Ka02c] wird darauf hingewiesen, dass in der Literatur keine einheitliche Abgrenzung unterschiedlicher Integrationskonzepte gefunden werden kann. Jedoch kann die Integration an drei unterschiedlichen Punkten eines Anwendungssystems ansetzen: der Präsentations-, der Daten- oder der Funktionsebene. Entsprechend werden die Präsentationsintegration, die Datenintegration und die Funktionsintegration als häufig genutzte Integrationskonzepte unterschieden.

2.3.3.1 Präsentationsintegration

Bei der Präsentationsintegration wird die Anwendungsintegration über die existierenden Benutzerschnittstellen der Anwendungen realisiert (Abbildung 2).

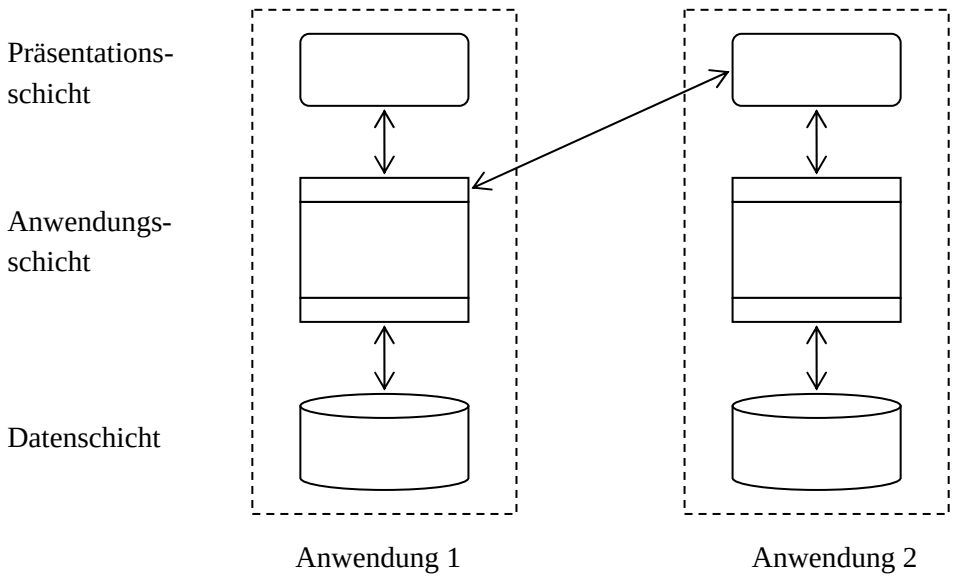


Abbildung 2: Präsentationsintegration [Ka02c]

Wenn ein System keine geeigneten Schnittstellen zum Zugriff auf die Daten oder die Funktionalität bereitstellt wird Präsentationsintegration eingesetzt. Oft kann nur auf diese

Weise die konsistente Führung der Anwendung ermöglicht werden. Notwendige Integritäts- und Plausibilitätsprüfungen werden somit zuverlässig durchgeführt.

Ein Beispiel für Präsentationsintegrarion ist das *Screen Scraping* [Ke02a]. Dabei handelt es sich um Altanwendungen, die keine Datenbank bzw. Anwendungsschnittstellen besitzen und deren Programme nicht erweiterbar sind. Eine Integration lässt sich nur über ihre Benutzerschnittstelle realisieren. Die Verbindung zur Anwendung wird durch die Simulation eines Benutzerdialogs hergestellt.

Eine Weitere alternative stellt die Integration von Anwendungen, die browserbasierte Benutzerschnittstellen bereitstellen. Die Integration erfolgt mittels Links. Die Benutzeranfragen werden transparent an die unterschiedlichen Server weitergeleitet.

Die Präsentationsintegration kann mit verfügbaren Entwicklungswerkzeugen einfach und schnell durchgeführt werden. Allerdings sind insbesondere die Performanz des Datenaustausches und die Sicherheit des Verfahrens kritisch zu überprüfen. Daher bleibt die Präsentationsintegration eine Notlösung für die Integration.

2.3.3.2 Datenintegration

Bei der Datenintegration wird die Integration durch den direkten Zugriff auf die Daten, die von den verschiedenen Anwendungen erzeugt, verwaltet und gespeichert werden, realisiert.

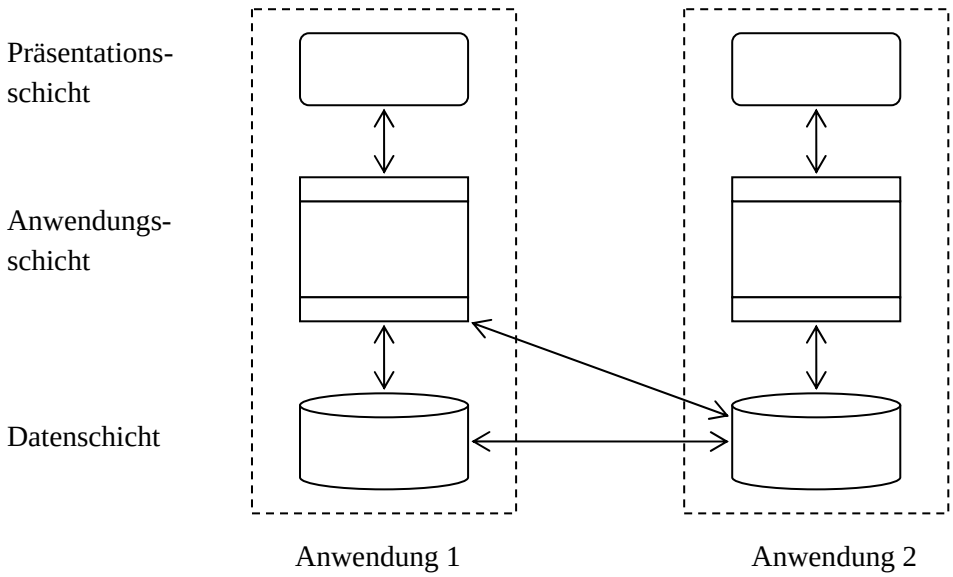


Abbildung 3: Datenintegration [Ka02c]

Bei der Datenintegration werden die Präsentations- und die Anwendungsschicht ignoriert und der Zugriff findet direkt auf den Datenbanken der Anwendungen. Die Datenintegration wird angewendet, wenn Applikationen gemeinsame oder redundante Daten nutzen.

Datenintegration kann grundsätzlich auf zwei Weisen realisiert werden. Einerseits kann der direkte Austausch von Daten zwischen verschiedenen Datenbanken ermöglicht werden.

Andererseits lässt sich der einheitliche Zugriff auf eine Vielzahl von Datenbanken durch die Verwendung einer virtuellen Datenbank realisieren. In diesem Fall wird der Begriff föderierten Datenbanken verwendet [Li03].

Die Nachteile bei diesem Integrationskonzept liegen in den möglichen Integritätsproblemen, die durch das Umgehen der Applikationslogik entstehen könnten. Außerdem lässt sich bei der Datenintegration Anwendungslogik nicht wiederverwenden.

2.3.3.3 Funktionsintegration

Die Funktionsintegration ist das weitgehende Konzept. Mittels Funktionsintegration lassen sich eine Reihe von Integrationsaufgaben lösen, inklusive der typischen Anwendungen der Präsentations- oder Datenintegration. Durch die Integration auf der Ebene der Anwendungslogik wird eine flexible Wiederverwendung von Anwendungsfunktionalitäten (Methoden) ermöglicht.

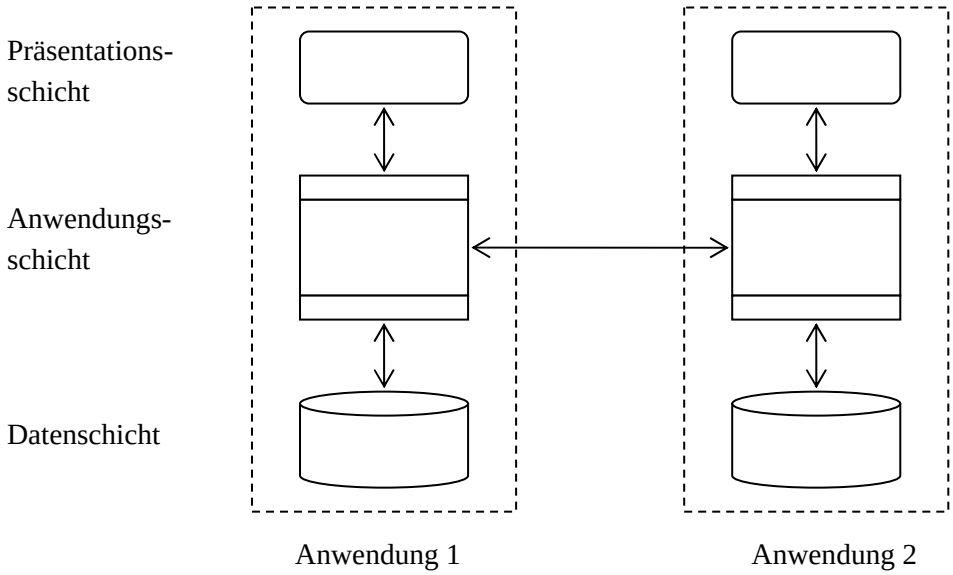


Abbildung 4: Funktionsintegration [Ka02c]

Bei diesem Integrationskonzept ruft eine Anwendung eine Methode auf, das von einer anderen Anwendung bereitgestellt wird. Im Unterschied zur Datenintegration wird hier die Anwendungslogik nicht umgegangen. Die Konsistenzhaltung geschieht auf der Funktionsebene. Auf diese Weise können existierende Funktionalitäten durch andere Systeme genutzt und flexibel zusammengefügt werden.

Für die Realisierung der Funktionsintegration können verschiedene Technologien wie RMI (Kapitel 5.4.3), Web Services (Kapitel 5.4.4) u.a. angewendet werden.

Ein Grenzfall bezüglich der Zuordnung innerhalb der Integrationskonzepte sind die datenorientierte Schnittstellen wie JSR 170. Technologisch betrachtet stellt deren Nutzung einen Funktionsaufruf dar. Obwohl der Aufruf solcher Methoden hauptsächlich Änderungen

auf der Datenschicht verursacht, werden sie in dieser Arbeit auf der Ebene der Funktionsintegration eingeordnet. Der Grund dafür liegt darin, dass solche datenorientierten Schnittstellen explizit als Dienst von den betrachteten Systemen auf der Anwendungsschicht bereitgestellt werden sollen, um von anderen Systemen benutzt zu werden.

Wesentliche Nachteile der Funktionsintegration ergeben sich aus der Komplexität bei der Realisierung der Integrationslösung. Notwendigen Änderungen sind an den zu integrierenden Anwendungen zu treffen, das mit Risiken und Kosten verbunden ist.

2.4 Java Content Repository

2.4.1 Java Specification Request 170

JSR 170: Content Repository for Java™ Technology API (JCR) ist ein Java Specification Request (JSR), der am 17 Jun, 2005 veröffentlicht wurde. Der CTO von Day Software AG David Nüscheler leitet die Spezifikationsgruppe, die große Infrastruktur-Anbieter wie IBM, Oracle, HP, BEA sowie zahlreiche Vertreter der ECM-Industrie umfasst. Apache Jackrabbit [Ap08a] ist die Open-Source-Referenz-Implementierung der Spezifikation.

Das Ziel der API wird folgenderweise in [Ja05] definiert:

> "The API should be a standard, implementation independent, way to access content bi-directionally on a granular level within a content repository."

Ein Content-Repository wird weiterhin als ein Informations-Management-System definiert, das eine Untermenge der traditionellen Repositories ist und Content-Services wie Versionierung, Volltextsuche, granulare Zugriffrechte und Überwachung implementiert. Abbildung 5 zeigt die zentralen Eigenschaften von JCR ein Vergleich zu relationale Datenbanken und Dateisysteme.

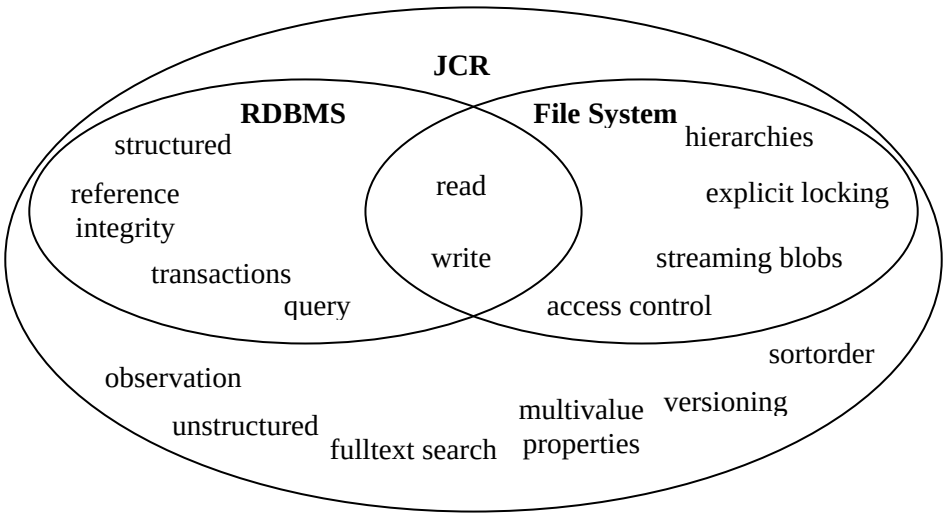


Abbildung 5: JCR im Vergleich zu RDBMS und Dateisystem [Da08a]

Die Spezifikation ist in zwei Teile gegliedert. Level 1 bietet Lese- und Level 2 zusätzliche Schreibeoperationen an. Darüber hinaus wird eine Reihe von zusätzlichen optionalen Funktionalitäten definiert, die Repositories von jedem Level unterstützen können.

Im Folgenden wird der JSR 170 Standard anhand der Spezifikation [Nü05] und [Fi05] präsentiert.

2.4.2 Repository Modell

Ein Content-Repository besteht aus einem oder mehreren *Workspaces*. Jeder *Workspace* besteht aus einem Baum von *Items*. Ein *Item* kann entweder ein *Node* oder ein *Property* sein. Abbildung 6 zeigt das UML-Content-Meta-Modell von JSR170. Witerhin haben Nodes Namen und bestimmen die Struktur des Contents, während Properties den Content enthalten. Durch Vater-Kind-Beziehungen zwischen Nodes bzw. zwischen Nodes und Properties wird eine Hierarchie definiert. Jedes Node kann beliebig viele Kinder-Nodes oder Kinder-Properties besitzen. Ein einziges Root-Node pro Workspace hat keinen Vater. Alle anderen Nodes haben einen Vater. Properties haben nur einen Vater (ein Node) und können keine Kinder haben, sie sind die Blätter des Baumes. Der Content in dem Repository ist somit innerhalb der Werte der Properties gespeichert.

Zur Veranschaulichung des Modells kann Abbildung 8 in Betracht gezogen werden, die das symbolische Modell eines Content Repositorys darstellt. Nodes sind als Kreise und Properties als Rechtecke dargestellt.

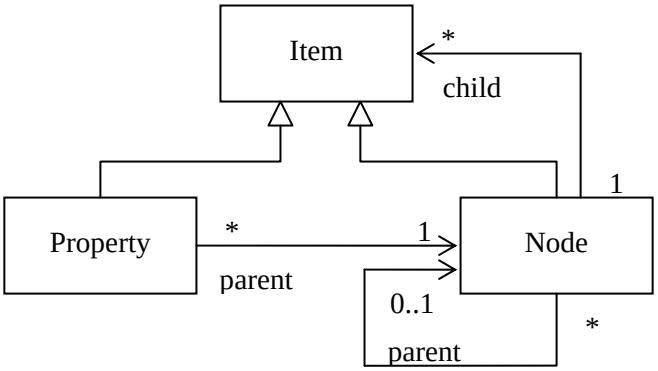


Abbildung 6: JSR 170 Content-Meta-Modell (UML-Diagramm) [Nü05]

Ein Property kann genau von einem der aufgezählten Typen sein:

- `PropertyType.STRING`
- `PropertyType.BINARY`
- `PropertyType.DATE`
- `PropertyType.LONG`
- `PropertyType.DOUBLE`
- `PropertyType.BOOLEAN`

- `PropertyType.NAME`
- `PropertyType.PATH`
- `PropertyType.REFERENCE`

Die Properties werden auf Java-Klassen abgebildet. Beispielsweise wird `PropertyType.STRING` auf `java.lang.String` abgebildet. `NAME`-Properties werden für die Speicherung von Zeichenketten, die das Namespace qualifizieren, wie die Namen der Node-Typen oder die Namen der Repository-Items. `PATH`-Properties representieren einen Pfad in dem Workspace und können auf weitere Items referenzieren. Die Integrität ist aber nicht gegeben, da ein `PATH`-Property auf einem ungültigen Pfad verweisen kann. `REFERENCE`-Properties werden dagegen verwendet, um auf Nodes im Workspace eindeutig zu verweisen. Jedes Node hat einen eindeutigen UUID (Universally Unique Identifier) [Wi08d], dessen Wert in dem `REFERENCE`-Properties gespeichert wird.

Bei der Erzeugung eines Nodes wird ihm ein Node-Typ zugewiesen. Der Typ eines Nodes definiert, welche Kind-Nodes und Properties ein Node haben darf. Ein Node-Typ darf einen oder mehreren Node-Typen vererben. Der Grundtyp von allen Node-Typs ist nt:base. Dabei werden alle Property- und Child-Node-Definitionen von dem Supertyp übernohmen. Weitere eigene Property- und Child-Node-Definitionen können definiert werden.

Jeder Node-Typ in einem Java Content Repository besitzt die folgenden Attribute:

- Name – Jeder in dem Repository registrierter Node-Typ muss einen eindeutigen Namen haben.
- Supertypes – Ein Node-Typ muss einen oder mehrere andere Node-Typs erweitern.
- Mixin status – Ein Node kann entweder *primary* oder *mixin* sein. (Kapitl 2.4.3.7)
- Orderable child nodes status – Ein Node-Typ bestimmt, ob die Kinder-Nodes *clien-orderable* sind. Dies bedeutet, dass die Kinder-Nodes eine Reihenfolge haben können.
- Property definitions – Ein Node-Typ bestimmt welche Properties ein Node haben darf.
- Child Node Definitions – Ein Node-Typ bestimmt welche Kinder-Nodes ein Node haben darf.
- Primary Item Name – Ein Node-Typ definiert den Namen eines primary Items.

Für eine ausführliche Beschreibung der Definition eines Node-Typs wird an dieser Stelle auf die JSR 170 Spezifikation verwiesen. In JSR 170 werden außerdem einige vordefinierte Node-Typs wie beispielsweise nt:file und nt:folder bereitgestellt.

2.4.3 Level 1 Funktionalität

JCR Level 1 spezifiziert Repositories mit Lesezugriff. Dies ermöglicht, dass beliebige Anwendungen implementiert werden können, denen insbesondere Lesezugriff auf Inhalte, die auf Legacy-Plattformen verwaltet werden, ermöglicht werden kann. Darüber hinaus

unterstützt Level 1 den Export mittels XML und ermöglicht die Migration von Content zu anderen Systemen oder Level 2 Repositories.

Level 1 beinhaltet folgende zentrale Funktionen:

2.4.3.1 Zugriff auf das Repository

Ein Client verbindet sich mit dem Repository durch den Aufruf einer Login-Methode mit dem gewünschten Workspace und Zugriffsdaten als Parameter. Wenn der Login erfolgreich ist, erhält der Client eine Session in dem angegebenen Workspace, der in Abhängigkeit von seinen Zugriffsrechten gefiltert wird. Der Client kann nur auf Nodes oder Properties zugreifen, auf die er berechtigt ist. Ein Repository kann entweder das eigene Zugriffskontrollsystem oder die Dienste eines externen Mechanismus, wie die Java Authentication and Authorization Service (JAAS) nutzen [Su08a].

2.4.3.2 Lesen von Repository-Content

Nachdem eine gültige Session erworben ist, kann der Client Items innerhalb des Workspaces durch Traversieren des Baumes, durch direkten Zugriff auf ein bestimmtes Node, oder durch Traversieren von Abfrage-Ergebnissen beziehen. Das Traversieren des Baumes erfolgt nach dem Zugriff auf das Root-Node. Danach kann auf die Kinder des Root-Nodes navigiert werden, die weiterhin traversiert werden können. Die direkte Abfrage von einem Knoten kann durch die Verwendung eines Pfades, wie zum Beispiel "/Firma/Mitarbeiter/Adresse", oder durch Benutzung des eindeutigen UUID eines referenzierbaren Knotens. Jede Methode hat ihre Vor-und Nachteile, in Abhängigkeit von der Struktur der Inhalte und der gewünschten Operationen.

2.4.3.3 Namespaces

Der Name eines Nodes oder eines Propertys kann ein Präfix haben, der mittels eines Doppelpunkts vom dem entsprechenden Namespace abgegrenzt sein. Namespaces in JCR sind wie XML Namespaces gestaltet: das Präfix bezieht sich auf einen Namensraum, identifiziert durch eine URI. Durch diesen Klassifizierer werden Namenskollisionen minimiert. Jeder konforme Repository-Namensraum hat eine Registrierung, dass die Zuordnung eines Namespace-Präfixes zu dem entsprechenden URI [Ne05] ermöglicht. Darunter werden einige eingebauten Namespace- Präfixe für das JCR selbst vorbehalten. In Level 1 Repositories kann das Präfix, das in einem Namensraum registriert ist, vorübergehend von einem anderen Präfix im Rahmen einer Sitzung überschrieben werden.

2.4.3.4 XML Mappings

JCR Level 1 unterstützt zwei mögliche Abbildungen des JCR-Datenmodells auf XML: das System-View und das Dokument-View. Das System-View-Mapping ermöglicht eine komplette verlustfreie Serialisierung von einem Workspace in eine XML-Datei. Das bedeutet, dass der gesamte Inhalt eines Workspaces exportiert werden kann. Der Vorteil des System-Views liegt darin, dass jedes gültige Content-Repository in XML exportiert werden kann.

Nachteilig ist, dass die daraus resultierende Datei schwer lesbar für einen Menschen ist. Im Gegensatz dazu ist das Dokument-View so konzipiert, dass es von Menschen lesbar ist. Dies wird aber auf Kosten der Vollständigkeit erreicht. Der Vorteil von dem Dokument-View entsteht in erster Linie durch die vereinfachten XPath-Abfragen.

2.4.3.5 Export von Repository-Content

Level 1 unterstützt den Export von dem Content des Repositorys sowohl für das System-View als auch für das Dokument-View. Das XML-Output ist dann entweder ein Stream oder SAX-Events.

2.4.3.6 Recherche von Repository-Content

XPath [W399] ist eine Abfragesprache, die ursprünglich für die Auswahl von Elementen aus einem XML-Dokument entworfen wurde. XPath bietet eine komfortable Syntax für die Suche in einem Java Content Repository, weil der Modell-Baum von Nodes und Properties analog zu einem XML-Dokument-Baum von Elementen, Attributen und Content-Elementen ist. XPath-Ausdrücken können definiert und ausgeführt werden, als wären sie auf das Dokument-View des aktuellen Workspaces angewandt. Die Rückgabe ist eine Tabelle mit Property-Namen und deren Belegung.

2.4.3.7 Node Types

Jedes Node in einem Repository JCR muss höchstens einen *primary* Node-Typ haben. Der primäre Node-Typ definiert die Namen, Typen und andere Merkmale der Properties und der Kind-Nodes, die für das gegebene Node erlaubt oder erforderlich sind. Ein Node kann auch durch ein oder mehrere "mixin" Typen zusätzliche Merkmale erwerben. (z.B. weitere Kind-Knoten, Properties und ihren jeweiligen Namen und Typen). JCR Level 1 stellt Methoden für die Identifizierung des Node-Types von bestehenden Nodes und für Auslesen der Definitionen von Node-Typen, die im Repository verfügbar sind.

2.4.3.8 Zugriffsrechte

In Repositories mit einem Berechtigungskonzept, wird die Methode `checkPermission()` benutzt, um abzufragen, ob eine bestimmte Session die Erlaubnis zur Durchführung von bestimmten Aktionen abhängig von der einschlägigen Zugriffskontrolle hat. Allerdings werden in der Spezifikation keine Mechanismen für die Festlegung der Zugriffskontrolle definiert.

2.4.4 Level 2 Funktionalität

JCR Level 2 erweitert die Funktionalität um das Lesens und Schreiben von Content, Import aus anderen Quellen, darunter auch aus anderen JCR Repositories, und die Verwaltung von der Definition von Content und die Strukturierung mittels erweiterbaren Node-Typen. Zusätzlich zu den Level 1 Funktionalitäten, muss ein Level 2 Repository die folgenden Funktionen unterstützen:

2.4.4.1 Schreiben von Repository-Content

Level 2 umfasst Methoden für das persistente Hinzufügen, Verschieben, Kopieren und Entfernen von Items in einem Workspaces innerhalb einer Session, wenn der Client eine Speicherung der Änderungen anfordert. Es werden auch Methoden zum Verschieben, Kopieren und Klonen von Items zwischen Workspaces bereitgestellt. Ein Client kann auch seine ungespeicherten Änderungen mit den aktuellen Zustand des Workspaces vergleichen, um z.B. zu entdecken, dass Änderungen möglicherweise von einem anderen Client gespeichert wurden. Der Client kann entweder den aktuellen Stand übelnehmen oder die Änderungen ganz verwerfen.

2.4.4.2 Hinzufügen und Löschen von Namespaces

Level 2-Repositories haben die Fähigkeit die gespeicherten Namespaces in der Namespace-Registrierung hinzuzufügen, zu entfernen und zu ändern, mit Ausnahme der built-in Namespaces von JCR.

2.4.4.3 Import von Repository-Content

Level 2 erlaubt, dass beliebige XML-Dokumente in das Repository als Baum von Nodes und Properties importiert werden. Wenn das XML-Dokument in der Form der JCR-System-View ist, ist der Import äquivalent zu einem "Wiederherstellen" des exportieren Contents nachdem zuerst die XML-Export-Funktionalität von Level 1 ausgeführt wurde. Andernfalls wird der XML-Import wie ein Dokument-View verarbeitet. Namespace-Deklarationen aus der XML-Datei werden in den Repository-Namensraum registriert und ein Baum von JCR-Nodes und Properties mit Struktur und Namen aus dem XML-Dokument wird angelegt.

2.4.4.4 Zuordnung von Node Types

Level 2 bietet Methoden für die Zuweisung von `primary` und `mixin` Types zu Nodes. In einigen Fällen erfolgt die Zuordnung automatisch auf der Grundlage der Definition des Vater-Knotens. Node-Typen können daher zur Durchsetzung von Datentyp-Einschränkungen auf Content eingesetzt werden.

2.4.5 Optionale Funktionalität

JCR bietet eine standardisierte Schnittstelle mit zusätzlichen Content-Services als optionale Funktionalität an. Diese Funktionen sind unabhängig voneinander und sind insbesondere nicht von Level 2 abhängig, so dass jede Eigenschaft individuell von jedem Level 1 und Level 2 Repository implementiert werden kann.

2.4.5.1 Transaktionen

Transaktionen können durch die Einhaltung der Java Transaction API (JTA) unterstützt werden. [Su08b] Bei JTA sind zwei allgemeine Ansätze für die Transaktionen vorgesehen: Container- und User-Managed-Transaktionen. Bei den Container-Managed-Transaktionen

wird die Transaktionverwaltung von dem Application-Server verwaltet. Bei den User-Managed-Transaktionen verwaltet der Clienet die Transaktionsgrenzen in der Anwendung. Ein JCR Implementierung muss diese beiden Ansätze unterstützen, wenn Transaktionen bereitstellt werden.

2.4.5.2 Versionierung

Die Versionierung ermöglicht, dass der Zustand eines Knotens erfasst wird, so dass er später angezeigt oder wiederhergestellt werden kann. Ein Repository muss eine spezielle Versions-Storage-Area haben, die aus Versions-Geschichten besteht. Das ist eine Sammlung von Nodes-Versionen, die miteinander durch eine Nachfolger-Beziehung verbunden sind. Eine neue Version wird an der Versions-Geschichte eines versionierbaren Nodes hinzugefügt, wenn einer seiner Workspace-Instanzen eingecheckt ist. Jede neue Version wird zu der Versions-Geschichte als Nachfolger von einem oder mehreren der vorhandenen Versionen hinzugefügt. Die sich daraus ergebende Versions-Geschichte ist ein gerichteter azyklischer Graph der Zustände des Nodes, die seine Änderungen im Laufe der Zeit protokollieren.

2.4.5.3 Überwachung

Anwendungen können Ereignisse überwachen, die Änderungen in einem Workspace verursachen, und gegebenenfalls darauf reagieren. Der Beobachtungs-Mechanismus versendet Ereignisse bei persistenten Änderung in dem Workspace.

2.4.5.4 Locks

Locks ermöglichen einem Benutzer vorübergehend Knoten zu sperren, um zu verhindern, dass andere Nutzer sie ändern. Diese Funktion wird normalerweise verwendet, um den Zugang zu einem Knoten zu reservieren, da JCR Level 2 automatisch widersprüchliche Updates durch die Verwendung von unabhängigen Workspaces verhindert.

2.4.5.5 SQL Suche

Die SQL Suche bietet eine zusätzliche Abfrage Sprache zu der XPath-Suche von Level 1.

2.4.6 Beispiel – JCR DA Content-Modell

In Abbildung 7 zeigt das UML-Modell von einer Beispielsanwendung JCR DA, die ein JCR zum speichern von Content benutzt. Es werden die in der Spezifikation definierten Node-Typen *nt:base, nt:hierarchyNode, nt:file, nt:folder, mix:referenceable* und *nt:resource* verwendet. In dem Namespace *da* wird der NoderType *da:da_content* registriert, der die Properties *da:created, da:createdBy* und *da:lastModifiedBy* zum Speichern des Erstellungdatums, des Namen des Erstellers und des letzten Bearbeiters definiert. Alle anderen Attribute werden entsprechend vererbt. Mit diesem Modell kann eine einfache DMS-Lösung realisiert werden, das auch eine Hierarchie durch Verzeichnisse realisiert. Abbildung 8 zeigt die tatsächliche symbolische Darstellung des Modells und die Zugehörigkeit der

Properties zu den entsprechenden Nodes. In Anang A ist die Definiton des Node-Types *da:da_content* in XML dargestellt.

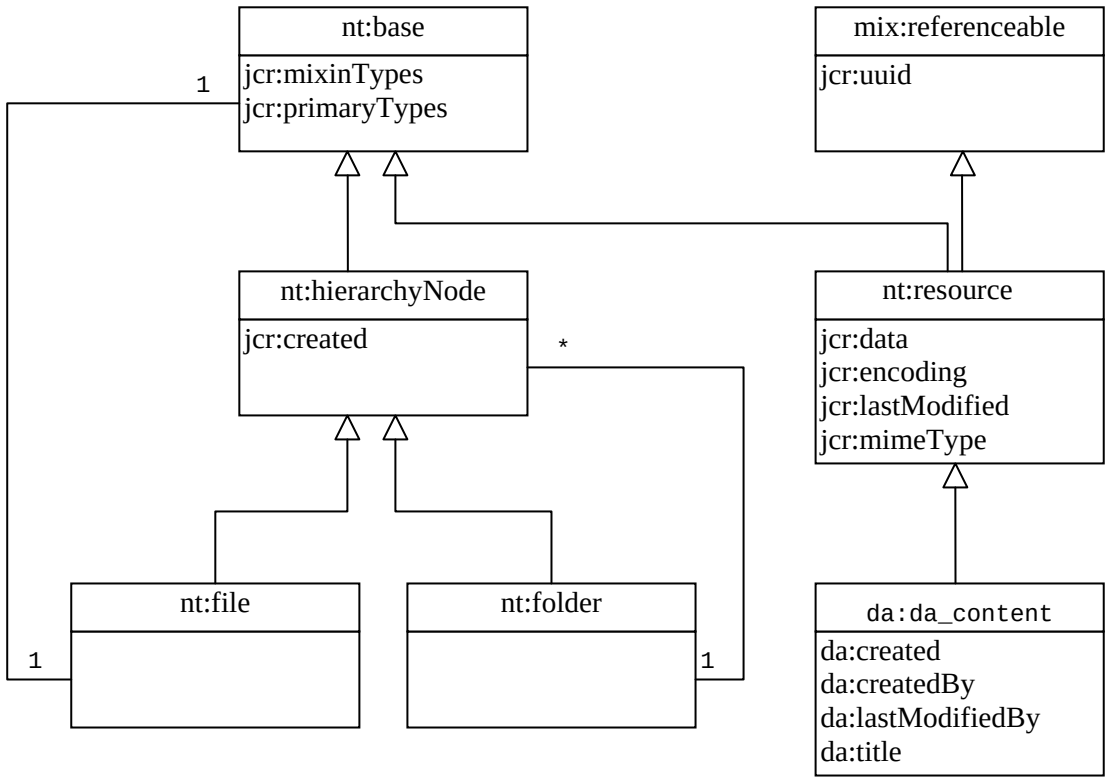


Abbildung 7: JCR DA – Content-Modell (UML Klassendiagramm)

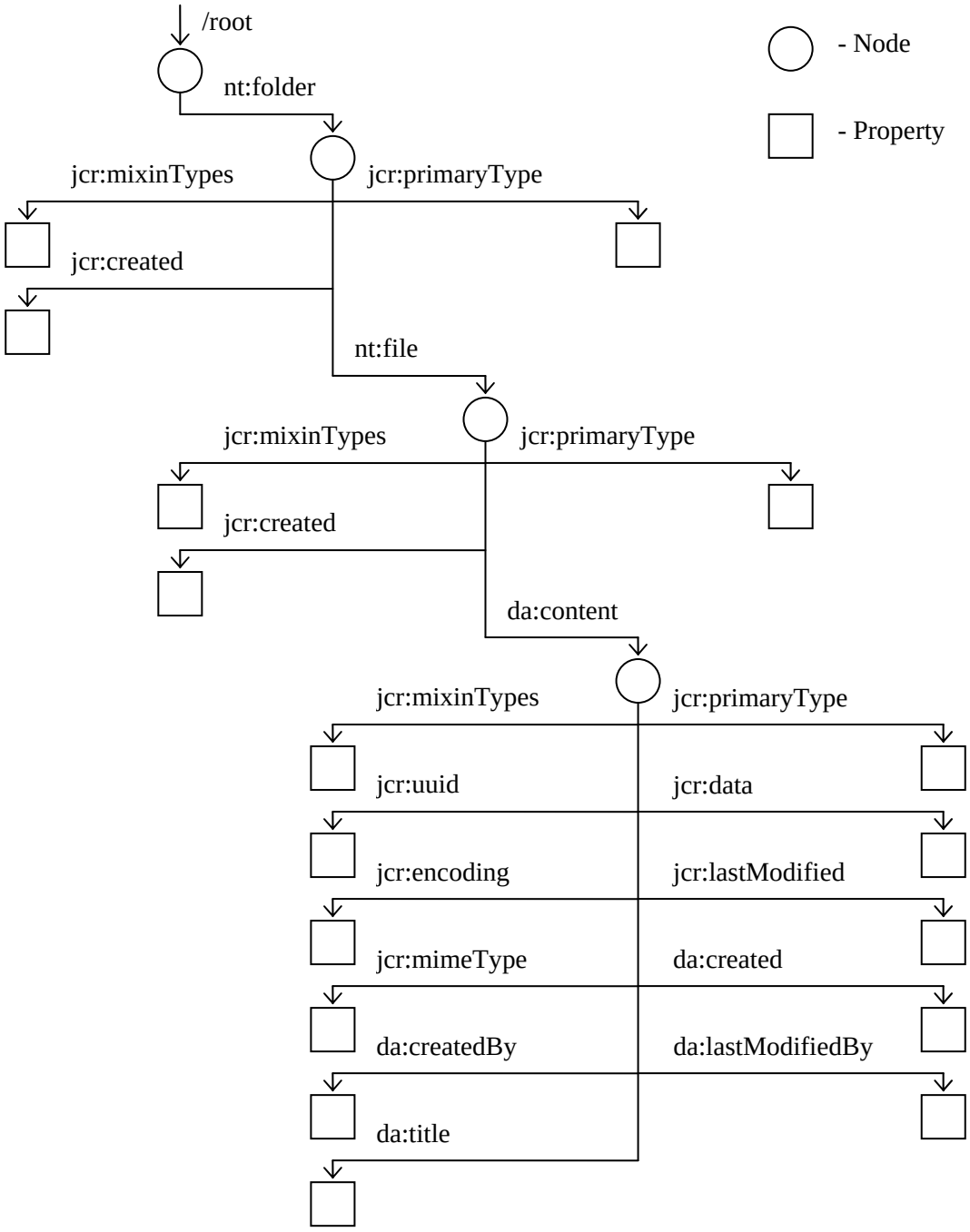


Abbildung 8: Symbolische Darstellung der Struktur des JCR DA Content-Modells

3 Gründe für Integration von ECM-Systemen

Integrationsprojekte von ECM-Systemen werden aus verschiedenen Gründen in einem Unternehmen angestoßen. In Rahmen dieser Arbeit wurden Expertengesprächen mit Beratern im ECM-Bereich des IT-Beraterunternehmens Logica Deutschland GmbH & Co. KG durchgeführt, bei denen die wesentlichen Gründe für die Integration identifiziert und in den folgenden Gruppen zusammengefasst wurden*: „Reduktion der Anzahl der Systeme“*, *„Austausch eines Produktes“, „Mehrfache Nutzung von Inhalten“* und *„systemübergreifende Funktionen“*.

3.1 Reduktion der Anzahl der Systeme

Nach Aussage der ECM-Experten der Firma Logica versuchen viele Unternehmen Inhalte zusammenzuführen um die Anzahl der im Unternehmen betriebenen ECM-Systeme zu reduzieren. Durch die damit verbundene Konsolidierung der Infrastruktur im Unternehmen kann in der Regel eine Effizienzsteigerung erzielt werden.

3.1.1 Anzahl der Instanzen reduzieren

Ist ein ECM-Produkt bei einem Unternehmen über längere Zeit etabliert, so werden über die Jahre oft mehrere getrennte Instanzen desselben Produktes für unterschiedliche Aufgaben oder Abteilungen installiert und betrieben. Der Einsatz getrennter Instanzen hat kurzfristige Vorteile für kleine und mittelgroße Abteilungen da hierfür wenig Abstimmungsbedarf besteht. Die langfristigen Nachteile werden typischerweise erst später sichtbar.

Content ist daher auf die verschiedenen Instanzen verteilt und von mehreren Systemen zu beziehen. Verwandte Inhalte werden ungleich schnell geändert und verarbeitet, was zur Existenz von Content in unterschiedlichen Versionen in den verschiedenen Instanzen führen kann. Eine konsistente Verwaltung der Inhalte ist in diesem Fall stark erschwert.

Weiterhin kann es schwierig sein, festzustellen, in welchem System sich jeweils die Daten befinden, die man gerade sucht. Das führt dazu, dass der Nutzen stark eingeschränkt wird.

Weitere Nachteile ergeben sich bei Änderungen in den Unternehmensanforderungen, weil Modifikationen an vielen Systemen gleichzeitig vorgenommen werden müssen.

Die Reduzierung der Anzahl der verwendeten Instanzen der Systeme stellt eine Möglichkeit dar, um die Konsistenz der Inhalte zu steigern. In dieser Hinsicht wird Redundanz durch die zentrale Verwaltung von Content vermieden. Außerdem werden durch die gesenkte Anzahl der verwendeten Systeme noch die Betriebs- und Lizenzkosten reduziert.

3.1.2 Anzahl der Produkte reduzieren

In einem Unternehmen können nicht nur eine Vielzahl von Produktinstanzen, sondern auch Produkte verschiedener Hersteller gleichzeitig betrieben werden. Die Heterogenität der eingesetzten Systeme verschärft zusätzlich die schon in Kapitel 3.1.1 dargestellte Problematik.

Die Tatsache, dass sich ECM-Produkte in der angebotenen Funktionalität meist signifikant unterscheiden, unterstützt den Trend, dass Mitarbeiter ihre Aufgaben an verschiedenen Plattformen erledigen, was die inkonsistente Verwaltung von Content vorantreibt.

Jedes Produkt stellt außerdem eine eigene proprietäre Schnittstelle für Zugriff auf die Inhalte bereit. Daraus resultiert, dass der Zugriff auf Content im Rahmen des Unternehmens nicht einheitlich ist. Mit der Anzahl der verschiedenen Produkte steigt auch die Anzahl der Schnittstellen für Zugriff auf die Systeme. Dies führt dazu, dass Inhalte isoliert verwaltet werden und für Systeme eines anderen Herstellers gesperrt bleiben [Mo07].

Das Unternehmen kann dann eine Ein-Produkt-Strategie wählen und die Anzahl der eingesetzten Produkte verringern. Auf diese Weise wird eine Konsolidierung der Systemlandschaft bewirkt. Der Einsatz einer einheitlichen und stabilen Plattform steigt außerdem die Benutzerzufriedenheit und erzielt Lerneffekte bei den Mitarbeitern. Ein weiterer Grund für die Verringerung der Anzahl der verwendeten Produkte sind die reduzierten Einzellizenzkosten für die Menge der End-Benutzer.

3.2 Austausch eines Produktes

Unternehmen haben hohe Anforderungen an ECM-Systeme, die in der Regel langfristig im Einsatz sind. Beispielsweise müssen Eingangsrechnungen in Deutschland nach HGB mindestens 10 Jahre aufbewahrt werden [De99]. Erwünschte Aufbewahrungsfristen können mehrere Jahrzehnte betragen. Vergleichsweise können Kundenakten im Lebensversicherungsbereich eine Aufbewahrung von 100 Jahren und mehr erfordern.

Diesen Zeiträumen steht eine immer schneller voranschreitende technische Entwicklung von Hard- und Software entgegen, sowie Veränderungen in Struktur und Organisation von Anbieter- und Anwenderunternehmen [ZGB05d].

In [Ka03g] wird in diesem Zusammenhang der Begriff *Continuous Migration* benutzt. Die kontinuierliche Migration ist eine Strategie zur langfristigen Sicherstellung der Verfügbarkeit von Information durch regelmäßige und kontrollierte Migrationen von den Informationsbeständen. Bereits bei der Auswahl eines Herstellers ist zu berücksichtigen, dass der Wegfall von Produkten oder von Anbietern nicht unvorhersehbar, sondern der Regelfall ist. In diesem Sinne ist der Migrationsplannung schon bei der Erstinstallation große Bedeutung beizumessen.

Da Unternehmen einerseits in Jahrzehnten Informationsverfügbarkeit denken und andererseits sicherstellen wollen, dass eingesetzte ECM-Produkte ohne hohen finanziellen Aufwand

ausgewechselt werden können, werden im Folgenden typische Gründe für die Migration vorgestellt.

3.2.1 Migration zu Produkten, die Standards Implementieren

Enterprise-Content-Management-Systeme bieten Dienste, die über das gesamte Unternehmen hinweg zur Verfügung stehen. Darüber hinaus werden Produkte unterschiedlicher Hersteller oftmals zusammengefügt, da kein Anbieter in der Lage ist alle möglichen Anforderungen an ein ECM-System zu erfüllen [Ka99f].

Die Architektur der Systeme soll daher Möglichkeiten zur Anbindung anderer Systeme bieten. In diesem Zusammenhang sind standardisierte Schnittstellen und Dokumentenformate für das Sicherstellen einer langfristigen Verfügbarkeit der Inhalte von großem Nutzen.

Die Verwendung von standardisierten Technologien im ECM-Umfeld bietet den Vorteil, dass diese meist weiter verbreitet und oft auch für ein breiteres Feld von Anwendungsszenarien konzipiert sind. Standardisierte Schnittstellen können dazu beitragen, dass die Daten nicht in einer Dateninsel isoliert werden, sondern für eine größere Menge verschiedener Anwendungen zur Verfügung stehen [Ka99g].

In [Ka99h] werden die Vorteile zusammengefasst, die durch Standardisierung für den Anwender entstehen: Investitionssicherheit, Verfügbarkeit von Komponenten verschiedener Hersteller, Prüfbarkeit und Migrationssicherheit.

Zusätzlich werden auch die Gründe, aus denen Standards für Hersteller von Bedeutung sind erfasst: Standardschnittstellen und Formate für die Produktentwicklung, Qualitätssicherung, langfristige Einhaltung von Formaten und Produktakzeptanz.

In diesem Sinne kann eine unternehmensweite Einigung auf Standards Migrationsprojekte zu Produkten, die Standards implementieren, anstoßen.

3.2.2 Migration zu Open-Source-Technologien

Berater der Firma Logica bestätigen, dass die Zahl der Unternehmen wächst, die sich für Open Source Technologien interessieren.

Es gibt mittlerweile Open-Source-Produkte, die umfassende ECM-Funktionalitäten anbieten. Allgemein spielen Open-Source-Projekte im Internet und Intranetbereich eine ständig wachsende Rolle und entwickeln sich für die Anwender sukzessiv zu einer durchaus attraktiven Alternative [ZGB05e]. Die Entwicklung von Produkten wie Alfresco (Kapitel 6.1) wird intensiv vorangetrieben.

Der Vorteil bei der Nutzung von Open-Source-Technologien liegt nicht nur in den Lizenzverträgen, sondern auch in der freien Verfügbarkeit des Quellcode der Anwendungen. Die Software darf beliebig kopiert, verbreitet und genutzt werden [Wi08e].

3.2.3 Migration zu ECM-Produkten führender Hersteller

Zum Zeitpunkt der Erstellung dieser Arbeit wird der ECM-Markt von proprietären Produkten etablierter Softwarehersteller dominiert. Aktuelle Analysen von Gartner [Sh07] und Forrester [Mc07] ergeben, dass der Markt von führenden Herstellern wie EMC und IBM beherrscht wird.

Im ECM-Markt findet eine Konsolidierung durch die Bildung von Allianzen, Firmenzukäufe und Kooperationen statt. Einige Anbieter setzen auf die Kooperationen mit Produkten von Partnerfirmen, andere auf die Übernahme ganzer Lösungen um das eigene Produktportfolio zu bereichern. Kleinere Anbieter sind gezwungen sich dem Trend anzuschließen, um nicht verdrängt zu werden.

Durch diese Marktkonzentration entstehen stärkere Anbieter mit klaren Strategien und eindeutiger Positionierung. Der Fokus liegt in der Interoperabilität verschiedener Systemkomponenten und der Entwicklung vertikalen Anwendungen, die alle ECM-Disziplinen abdecken. Dementsprechend werden durch diese Vorteile die Risiken für Endanwender reduziert [Ka99i].

Angesichts der Dynamik des ECM-Marktes und der Produktvielfalt ist es für die Anwender schwer, die richtige Entscheidung bei der Einführung eines ECM-Systems zu treffen. Ein ECM-System hat nachhaltige Auswirkungen auf die Infrastruktur, Organisation und Informationsverfügbarkeit eines Unternehmens. Das System soll den Zugriff auf alle gespeicherten Informationen jederzeit, auch nach Jahren oder Jahrzehnten ermöglichen.

Viele Unternehmen setzten daher auf Produkte, die sich in der Vergangenheit bewährt haben. Führende Hersteller sind in der Lage große Referenzinstallationen vorzuweisen und gewinnen damit leichter das Vertrauen der Kunden. Überdies sind der Serviceaspekt und die sichergestellte Anpassung an neue gesetzliche Vorgaben von großer Bedeutung [Ka99j].

3.2.4 Migration von stark angepasster Software zu Produktstandards

Trotz des Erfolgs der ECM-Suiten werden bis heute häufig in Unternehmen auch eigene Anwendungen entwickelt, die ECM-Funktionalitäten implementieren. Dies können z.B. Web-Content-Management-Systeme sein, die von der eigenen IT-Abteilung entworfen werden. Mit der Zeit werden die Systeme weiterentwickelt und die Funktionalität wird erweitert. Da aber die Architektur der Systeme nicht von Anfang an als umfangreiche Lösung geplant wurde, leidet die Qualität der Anwendung bei zunehmender Komplexität. Die Systeme werden sehr schwer wartbar und erweiterbar.

Etablierte Produkte dagegen bieten eine solide Architektur, sowie einen standardisierten und umfangreichen Funktionsumfang. Die Integration einer ECM-Suite in eine bestehende IT-Infratsruktur ist mithilfe der klassichen ECM-Schnittstellen in der Regel wesentlich einfacher als bei Individualsoftware.

Typischerweise sind Eigenentwicklungen im ECM-Umfeld nur bis zu einer gewissen Größe betreibbar. Da sowohl die Unternehmen selbst als auch deren Datenvolumen einem

fortwährenden Wachstumstrend unterworfen sind, kommt es häufig vor das ECM-Individualsoftware an ihre Leistungsgrenzen stößt. Der Umstieg auf ein professionell entwickeltes Produkt ist in dem Fall nahezu unumgänglich.

Der wesentliche Vorteil der Produkthersteller besteht darin, dass die Entwicklungskosten einer ECM-Lösung auf mehrere Kunden verteilt werden kann. Dasselbe Prinzip gilt auch für Dienstleister die Beratungs- und Entwicklungsleistungen zu einem Produkt anbieten. Während für eine Eigenentwicklung zusätzliche Arbeitskräfte immer wieder neu angelernt werden müssen, gibt es bei etablierten ECM-Produkten eine große Zahl von Partnerfirmen, die Dienstleistungen anbieten.

3.3 Mehrfache Nutzung von Inhalten

ECM-Systeme werden nicht als reine Funktionsinseln betrieben. Vielmehr stellen die Systeme Funktionalitäten bereit, um über Schnittstellen mit kaufmännischen oder technischen Anwendungen integriert zu werden [ZGB05f]. In dieser Hinsicht können auch vorhandene Fachanwendungen mit ECM-Funktionalität erweitert werden.

3.3.1 Vermeidung von redundanter Datenspeicherung

Die redundante Datenspeicherung, wie schon in Kapitel 3.1.1 dargestellt, kann die Inkonsistenz der verwalteten Inhalte verursachen.

Beispielsweise werden in Unternehmen oft gleichzeitig Content- und Dokumenten-Management-Systeme eingesetzt. Das CMS-System kann verschiedene Inhalte zum Download den Benutzern anbieten. In diesem Fall sollen die Inhalte in dem Repository des Systems gespeichert werden.

Die Inhalte selbst werden bei deren Erstellung und Bearbeitung von Mitarbeitern des Unternehmens in einem Dokumenten-Management-System verwaltet. Nachträglich werden die Inhalte in das CMS durch manuelles Kopieren übertragen. Dieser Prozess erzeugt eine Redundanz der Inhalte, die in verschiedenen Systemen gleichzeitig zu verwalten sind. Folglich muss bei jeder neuer Version der Inhalte der Kopiervorgang wiederholt werden, damit die aktuellen Inhalte den Nutzern des CMS-Systems zur Verfügen stehen.

Um Inkonsistenzen und redundante Datenspeicherung zu vermieden, werden die Inhalte nunmehr nur in das DMS-System verwaltet. Nach einer Integration der Systeme, kann das CMS-System die zu darstellenden Inhalte von dem DMS-System beziehen.

3.3.2 Inhalte für mehrere ECM-Systeme zugreifbar machen

Im Gegensatz zu dem Austausch eines Produktes (Kapitel 3.2), bei dem Inhalte auf ein neues System migriert werden, kann ein Unternehmen entscheiden, dass die Alt-Systeme parallel weiterhin betrieben werden. Es wird deswegen eine Integrationslösung gebraucht, die den Zugriff auf die gesperrten Inhalte ermöglicht. Die Inhalte werden in den Alt-Systemen verwaltet, die dann sukzessive ausaltern und nach einiger Zeit nicht mehr genutzt werden

[Ka03h]. Mitarbeiter bekommen somit Zugriff auf diesen Inhalten über die Benutzerschnittstellen der neuen Umgebungen. So kann der Umstieg auf die neuen Systeme stufenweise erfolgen. Neue Inhalte werden in den aktuellen Systemen gepflegt.

Eine Integrationslösung wird auch gebraucht, wenn nach den Unternehmensanforderungen existierende Anwendungen Zugriff auf weitere Datenquellen bekommen sollen. Die Anwendungen selbst sollen nicht geändert werden.

Inhalte, die in mehreren Systemen verwaltet werden, können über Schnittstellen anderen ECM-Systemen zur Verfügung gestellt werden. Somit kann eine Integrationslösung den Einheitlichen Zugriff auf Inhalte ermöglichen, die in unterschiedlichen Systemen gespeichert sind. Inhalte stehen in Folge dessen über das gesamte Unternehmen hinweg zur Verfügung, unabhängig davon in welchem System sie verwaltet werden.

3.3.3 Integration von ECM-Funktionalität in bestehende Anwendungen

Viele betriebliche Anwendungen bieten keine ECM-Funktionalitäten haben aber standardisierte Schnittstellen um diese zu nutzen.

Office-Anwendungen, die für das Bearbeiten von Dokumenten benutzt werden, können erweitert werden, damit das direkte Laden und Speichern von Dokumenten in das DMS ermöglicht wird. Somit stehen den Benutzern Funktionen wie Versionierung und Suche automatisch zur Verfügung. ECM-Systeme bieten darüber hinaus unterschiedliche Lösungen für die Gruppenarbeit.

Weiterhin können DMS- und ERP-Systemen integriert werden. Über die vom DMS bereitgestellten Schnittstellen werden der Transfer und die Recherche von Dokumenten ermöglicht. Beispielsweise können Excel-Kalkulationen, CAD-Zeichnungen, Outlook-Termine in engen Kontext mit den strukturieren ERP-Prozessen gebracht und elektronisch verknüpft werden [ZGB05g]. Die Anbindung erfordert eine Integrationslösung, die es erlaubt, das DMS aus dem ERP-System heraus aufzurufen und Suchanfragen zu übergeben.

3.4 Systemübergreifende Funktionen

Durch eine Integrationslösung können systemübergreifende Funktionen wie Recherche und statistische Auswertungen über einer Mehrzahl von ECM-Systemen auf eine einheitliche Weise ermöglicht werden.

3.4.1 Recherche

Eine der zentralen Aufgaben eines ECM-Systems ist die effiziente Bereitstellung der verwalteten Inhalte. Die Inhalte, die benötigt werden, sollten jederzeit konsistent und eindeutig gefunden werden. In [Ka99k] wird die Recherche in einem ECM-System allgemein in zwei Kategorien unterschieden: Suche anhand von Attributen und inhaltliche Suche.

Bei der Suche nach Attributen werden Inhalte anhand ihrer strukturierten Metadaten recherchiert. Die bei der Erfassung der Inhalte definierten Attribute können für die Suche

verwendet werden. Beispielsweise kann nach Inhalten gesucht werden, die von einem bestimmten Autor erstellt wurden oder die in einem gegebenen Zeitraum bearbeitet wurden. Die Kombination der Suchkriterien mit booleschen Operatoren (und, oder, nicht), relationalen Operatoren (gleich, größer gleich, kleiner gleich, Bereiche) oder Wildcards kann die Suchmöglichkeiten erweitern, da es sich hier um strukturierten Daten handelt

Die inhaltliche Suche basiert auf einer Volltextindexierung und ermöglicht das Durchsuchen unstrukturierter Inhalte nach Termen. Die inhaltliche Suche ist vorteilhaft wenn große Mengen unstrukturierter Daten vorliegen, sowie bei unbekannten Rechercheanforderungen. Die Stärke der Volltextsuche ist die Tatsache, dass, unabhängig von der Beschaffenheit und Organisation der Daten, nahezu jede gewünschte Information gefunden werden kann. Allerdings besteht auch die Gefahr, dass mit zunehmendem Datenbestand bei einfachen Anfragen immer mehr nicht zutreffende Ergebnisse geliefert werden, wodurch relevante Inhalte nur mühsam oder gar nicht gefunden werden [ZGB05h].

Beide Ausprägungen der Recherche werden in ECM-Systemen intensiv eingesetzt. Da die Zusammenführung aller ECM-Systemen in dem meisten Unternehmen nicht durchführbar ist, kann eine Metasuche beim Auffinden von Informationen hilfreich sein. Eine Metasuche schickt die Suchanfrage an alle angeschlossenen Systeme und bereitet die Ergebnisse in Form einer einheitlichen Liste auf.

3.4.2 Statistische Auswertungen

Durch statistische Auswertungen können Informationen in mehreren ECM-Systemen analysiert werden. In dieser Hinsicht kann eine umfassende Sicht auf den Content in den ECM-Systemen im Unternehmen gewonnen werden. Außerdem können Muster, Trends und andere, oft wertvolle Erkenntnisse über die Inhalte abgeleitet werden.

Unter Betrachtung verschiedener Kriterien können Informationen über die verwalteten Inhalte bezogen werden. Beispielsweise könnte von Interesse sein, wie viele Dokumente in den Systemen verwaltet werden. Welche Inhalte von welchen Mitarbeitern bearbeitet werden. Zu welchem Zeitpunkt welche Daten veröffentlicht werden.

Weiterhin können risikobehafteten Daten und Content identifiziert werden. So kann ein Unternehmen das Risiko bei Rechtsstreitigkeiten reduzieren, sich besser gegen Betrugsversuche schützen und die Kontrollmöglichkeiten verbessern. Darüber hinaus benötigen Unternehmen eine einheitliche Sicht über alle ihre Daten, um sicherzustellen, dass Geschäftsdokumente entsprechend der gesetzlichen Bestimmungen, Branchenvorgaben und Unternehmensrichtlinien aufbewahrt werden. Außerdem sollen die Zugriffsrechte der Anwendergruppen kontrolliert werden.

Statistische Auswertungen erfordern den Datenzugriff auf verschiedenste Arten von Daten. Softwarekomponenten, die statistische Auswertungen in großen Unternehmen betreiben, müssen daher typischerweise mit vielen verschiedenen ECM-Systemen und Datenbanken kommunizieren.

4 Identifikation von typischen technischen Integrationsszenarien

In diesem Kapitel wird ein formales Model für Integration eingeführt. Auf diese Weise können alle möglichen Integrationsszenarien identifiziert werden. Allerdings ist nicht jedes mögliche Szenario in der Praxis sinnvoll. Aus diesem Grund werden zuerst alle möglichen Fälle, die eine Integration darstellen, betrachtet und danach wird das Modell auf die in Kapitel 3 identifizierten Gründe für Integration angewendet. Anschließend werden die Ergebnisse zusammengefasst und eine Menge der grundlegenden Integrationsszenerien, die die Integration im ECM-Bereich vollständig charakterisieren wird abgeleitet.

4.1 Definition eines formalen Modells für die Integration von ECM-Systemen

In Kapitel 2.3.3 wurde die Einteilung der Integrationskonzepte nach [Ka02c] betrachtet. Eine wesentliche Erkenntnis dabei ist, dass die Integration an drei unterschiedlichen Punkten eines Anwendungssystems ansetzen kann: der Präsentations-, der Daten- oder der Funktionsebene. Es wurde noch vereinbart, dass Integration mittels datenorientierter Schnittstellen auf der Ebene der Funktionsintegration eingeordnet wird.

Im Folgenden wird ein Modell eingeführt, mit dessen Hilfe die Integration von Systemen über datenorientierten Schnittstellen formalisiert werden kann. Dies wird durch die Einführung einer formalen Notation erreicht, die die erforderlichen Aktionen für die Durchführung eines Szenarios eindeutig definiert.

4.1.1 UML Modell für Integration

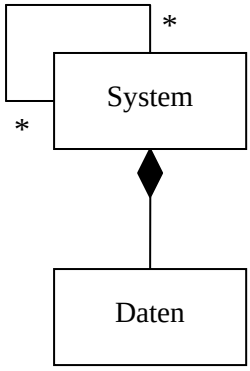


Abbildung 9: Systemintegration (UML-Klassendiagramm)

Die allgemeine Architektur eines ECM-Systems wurde in Kapitel 2.2 dargestellt. Dementsprechend hat jedes System ein eigenes Repository, in dem der Content gespeichert

wird. Im Allgemeinen kann ein ECM-System wie in Abbildung 9 dargestellt als eine Komposition der Anwendung selbst und der verwalteten Inhalte gesehen werden.

Ferner bietet jedes System datenorientierte Schnittstellen für Zugriff auf die Inhalte. Diese Schnittstellen können dann zur Datenübertragung von anderen Systemen benutzt werden. Insofern steht ein System mit einem anderen in Beziehung, wenn das eine System die datenorientierten Schnittstellen des anderen nutzt. Dies wird hier als eine bidirektionale Assoziation zwischen zwei Systemen betrachtet, da die Funktionsaufrufe der angebotenen Schnittstellen in beiden Richtungen erfolgen können.

Mit Hilfe dieses UML-Modells lässt sich eine Systemlandschaft von Anwendungen beschreiben, die mittels datenorientierter Schnittstellen integriert sind.

4.1.2 Formale Integrationsnotation

Für die weitere Analyse des Begriffs Integration wird hier eine formale Notation eingeführt, mit deren Hilfe alle möglichen Ausprägungen der Integration mittels des beschriebenen Modells genauer betrachtet werden. Dabei handelt es sich um elementare Aktionen, die hintereinander ausgeführt werden. Die Bedeutung der Aktionen ist in Tabelle 1 definiert.

Aktion	Bedeutung
$SN_{(a)}$	*System New* – Ein neues System hinzufügen
$SD_{(a)}$	*System Delete* – Ein existierendes System *a* entfernen
$CN_{(a,b)}$	*Connection New* – Neue Verbindung von System *a* nach System *b* erzeugen
$CD_{(a,b)}$	*Connection Delete* – Eine existierende Verbindung zwischen System *a* und System *b* löschen
$DT_{(a,b)}$	*Data Transfer* – Einmaliger Datentransfer. System *a* liest und schreibt Daten auf System *b*
$DT^*_{(a,b)}$	Der Datentransfer kann 0 bis n Mal stattfinden.
$DT^+_{(a,b)}$	Der Datentransfer kann 1 bis n Mal stattfinden.

Tabelle 1: Formale Integrationsnotation

Eine Systemlandschaft wird im Wesentlichen durch Hinzufügen oder Entfernen von Systemen oder Beziehungen verändert. Für eine Integration wird zusätzlich ein Datenaustausch vorausgesetzt. Auf diese Weise lässt sich ein Integrationsfall eindeutig durch eine Reihenfolge von Aktionen beschreiben. Folglich kann die Menge aller möglichen Integrationsfälle mittels der eingeführten Notation charakterisiert werden. Dies ist der

wesentliche Grund für die Verwendung dieser Notation, weil auf diese Weise die sinnvollen Integrationsszenarien identifiziert werden können.

Zuerst wird die Erstellung eines neuen Systems mit $SN_{(a)}$ definiert. Der Index in den Klammern ist eine natürliche Zahl und dient zur eindeutigen Identifikation eines Systems.

Die schon beschriebene Beziehung von zwei Systemen wird hier durch eine Verbindung realisiert. $CN_{(a,b)}$ erzeugt eine Verbindung von $System_a$ nach $System_b$. Dies bedeutet das $System_a$ die datenorientierten Schnittstellen *von* $System_b$ nutzen kann.

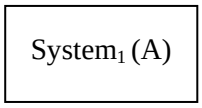
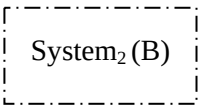
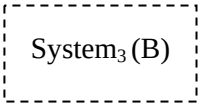
Solange die Verbindung zwischen den Systemen hergestellt ist, kann ein Datenaustausch stattfinden. Der einmalige Datentransfer wird mit $DT_{(a,b)}$ bezeichnet, wobei $System_a$ Daten von $System_b$ liest. Die Operatoren $^+$ und * werden zusammen mit der Aktion *DT* verwendet, um den Datenaustausch näher zu charakterisieren. Bei der Anwendung von $DT^*_{(a,b)}$ wird der Datentransfer 0 bis n Mal und bei $DT^+_{(a,b)}$ 1 bis n Mal stattfinden.

Eine Verbindung zwischen zwei Systemen $System_a$ und $System_b$ lässt sich mittels $CD_{(a,b)}$ schließen. Dadurch wird die Beziehung zwischen den beiden Systemen gelöscht und der Datentransfer kann nicht mehr stattfinden.

Sind alle eingehenden und ausgehenden Verbindungen zu einem $System_a$ gelöscht, so ist es möglich das System selbst mit $SD_{(a)}$ aus der Systemlandschaft zu entfernen.

4.1.3 Symbolische Darstellung von Integration

Zur schematischen Veranschaulichung der Integrationsfälle werden zusätzlich noch Symbole für die Darstellung eingeführt. In Tabelle 2 ist die Bedeutung der Symbole definiert.

Symbol	Bedeutung
	Vorhandenes System
	Neu erstelltes System
	Zu löschendes System
	Angebotene Schnittstelle
	Angeforderte Schnittstelle

—◯—	Verbindung zwischen zwei Systemen. Die Verbindung ist in die Richtung der angebotenen Schnittstelle.

Tabelle 2: Symbolische Integrationsnotation

Die Verbindung zwischen zwei Systemen wird durch die Lollipop-Notation dargestellt. Dabei wird zwischen einer angebotenen und einer angeforderten Schnittstelle unterschieden. In Kapitel 5.2 wird näher auf die Verbindung der Systeme eingegangen.

4.1.4 Notwendige Bedingung für Integration

Die Hintereinanderausführung der definierten elementaren Aktionen kann eine gegebene Systemlandschaft verändern. Allerdings stellen nicht alle möglichen Kombinationen der Aktionen einen Integrationsfall.

Die notwendige Bedingung für die Klassifikation einer Reihe von Aktionen als einen Integrationsfall ist die Erstellung einer Verbindung mit *CN* und die Durchführung mindestens eines Datenaustausches mit DT^+ vor dem Schließen der Verbindung mit *CD*. Diese Art von Integration wird hier als *temporäre Integration* bezeichnet. Der Datenaustausch kann nur stattfinden, solange die Verbindung hergestellt ist. Die auszuführende Reihe von Aktionen ist: *CN*, DT^+, *CD*.

Wird die Verbindung nicht geschlossen, darf *DT** die Bereitschaft auf einen Datentransfer ausdrücken. Es handelt sich dann um eine *persistente Integration*. In diesem Fall ist ein Datenaustausch immer möglich. Die Aktionen, die hier hintereinander ausgeführt werden sollen, sind: *CN*, *DT**.

Bei der Definition der Symbolischen Darstellung wurde zwischen dem Index und dem Typ eines Systems unterschieden. In der formalen Notation wird aber auf die Angabe des Systemtyps verzichtet, damit die eingeführte Notation möglichst einfach bleibt. Weiterhin wird angenommen, dass die Aktion DT eine Datentransformation durchführt, wenn Daten zwischen Systemen unterschiedlichen Typs transportiert werden. Des Weiteren wird angenommen dass die transportierten Daten auf das Zielsystem gespeichert bleiben und die explizite Einführung einer Aktion zum Speichern der Daten wird unterlassen.

4.2 Beschreibung einer vollständigen Liste der möglichen Integrationsfälle

Im Folgenden werden die Integrationsmöglichkeiten zwischen zwei Systemen unter Benutzung der eingeführten Notation analysiert. Diese Vorgehensweise lässt sich analog auf eine beliebige Anzahl von Systemen erweitern. Es werden unterschiedliche Integrationsfälle analysiert und die Ergebnisse werden tabellarisch dargestellt, wobei folgende Kriterien zur vollständigen Beschreibung eines Falls verwendet werden:

- Voraussetzungen – die kleinste Menge von Aktionen, die ausgeführt werden sollen, so dass ein beschriebener Fall eintreten kann,

- Aktionen – die formale Beschreibung eines Falls, das mit Hilfe von elementaren Aktionen dargestellt wird,
- Ausprägungen – alle möglichen Ausprägungen der definierten Aktionen, die für zwei Systeme theoretisch möglich sind.

Die möglichen Fälle werden mit Hilfe der symbolischen Darstellung für Integration abgeleitet. Alle möglichen Kombinationen bei zwei Systemen ergeben:

- vorhandenes System und vorhandenes System
- vorhandenes System und neues System
- neues System und zu löschendes System
- vorhandenes System und zu löschendes System
- neues System und neues System
- zu löschendes System und zu löschendes System

Der Fall *zu löschendes System und zu löschendes System* wird in der folgenden Analyse nicht behandelt, da er für eine gegebene Systemlandschaft keine sinnvolle Integration darstellt, da alle Daten verloren gehen, wenn die beiden Systeme gelöscht werden.

Der Fall *neues System und neues System* wird nach der Erstellung der Systeme analog zum Fall *vorhandenes System und vorhandenes System* und deswegen nicht separat behandelt.

4.2.1 Zwei vorhandene Systeme

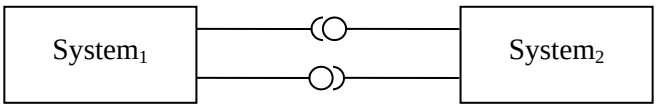


Abbildung 10: Vorhandenes System und vorhandenes System

Wenn zwei Systeme zur Verfügung stehen, kann jedes System mit dem anderen durch *CN* verbunden werden. Nach Herstellung der Verbindung kann der Datenaustausch mit *DT* ermöglicht werden.

4.2.1.1 Fall: Persistente Integration

Bei der persistenten Integration von zwei bestehenden Systemen bleibt die Verbindung auf Dauer erhalten, so dass der Datenaustausch *DT** nicht zeitlich begrenzt ist. Eine Verbindung kann in beiden Richtungen aufgebaut werden.

Voraussetzungen	$SN_{(1)}$, $SN_{(2)}$
Aktionen	**CN, DT***
Ausprägungen	$CN_{(1,2)}$, $DT^*_{(1,2)}$ $CN_{(2,1)}$, $DT^*_{(2,1)}$

Tabelle 3: Persistente Integration

4.2.1.2 Fall: Temporäre Integration

Bei der temporären Integration dagegen wird die Verbindung nach einer bestimmten Anzahl von *DT* Aktionen durch *CD* geschlossen. Danach kann kein Datenaustausch zwischen den beiden Systemen erfolgen. Hier wird die DT^+ Aktion benutzt, da *DT* mindestens einmal stattfinden soll, damit dieser Fall als Integration gilt.

Voraussetzungen	$SN_{(1)}$, $SN_{(2)}$
Aktionen	**CN, DT^+, CD**
Ausprägungen	$CN_{(1,2)}$, $DT^+_{(1,2)}$, $CD_{(1,2)}$ $CN_{(2,1)}$, $DT^+_{(2,1)}$, $CD_{(2,1)}$

Tabelle 4: Temporäre Integration

4.2.2 Hinzufügen eines neuen Systems

Eine Systemlandschaft kann erweitert werden, in dem ein neues System mittels *SN* hinzugefügt wird. Das neue System kann dann eigenständig arbeiten, oder mit den schon existierenden Systemen persistent oder temporär integriert werden. Dabei erfolgt die Integration analog zu der Integration von zwei vorhandenen Systemen (Kapitel 4.2.1).

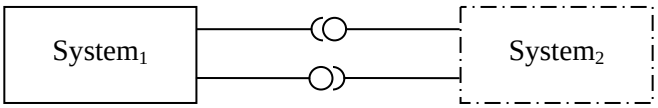


Abbildung 11: Vorhandenes System und neues System

4.2.2.1 Fall: Neues System und persistente Integration

Nach dem Hinzufügen eines neuen Systems mit *SN* wird eine persistente Integration durchgeführt.

Voraussetzungen	$SN_{(1)}$, $SN_{(2)}$
Aktionen	**SN, CN, DT***
Ausprägungen	$SN_{(2)}$, $CN_{(1,2)}$, $DT^{*}_{(1,2)}$ $SN_{(2)}$, $CN_{(2,1)}$, $DT^{*}_{(2,1)}$

Tabelle 5: Neues System und persistente Integration

4.2.2.2 Fall: Neues System und temporäre Integration

Nach dem Hinzufügen eines neuen Systems mit *SN* wird eine temporäre Integration durchgeführt.

Voraussetzungen	$SN_{(1)}$, $SN_{(2)}$
Aktionen	**SN, CN, DT^{+}, CD**
Ausprägungen	$SN_{(2)}$, $CN_{(1,2)}$, $DT^{+}_{(1,2)}$, $CD_{(1,2)}$ $SN_{(2)}$, $CN_{(2,1)}$, $DT^{+}_{(2,1)}$, $CD_{(2,1)}$

Tabelle 6: Neues System und temporäre Integration

4.2.3 Löschen eines vorhandenen Systems

Eine weitere Möglichkeit zur Integration stellt das Löschen eines Systems dar. Damit die notwendige Bedingung für Integration erfüllt ist, muss mindestens ein temporärer Datentransfer erfolgen. Zusätzlich sollen alle eingehenden und ausgehenden Verbindungen zu dem System vor dem Löschen geschlossen werden, um die Konsistenz der modellierten Systemlandschaft zu garantieren.

Ein System kann gleichzeitig in verschiedenen Integrationsfällen beteiligt sein. Es können beispielsweise folgende Aktionsfolgen unabhängig voneinander auftreten:

Voraussetzung: $SN_{(1)}$, $SN_{(2)}$,

$CN_{(2,1)}$, $DT^{+}_{(2,1)}$, $CD_{(2,1)}$ (1)

$CN_{(1,2)}$, $DT^{*}_{(1,2)}$ (2)

Im ersten Fall ist eine temporäre Integration durchgeführt und die Verbindung ist geschlossen. Im zweiten Fall bleibt die Verbindung erhallten, da $System_1$ einen persistenten Datenaustausch mit $System_2$ voraussetzt. Wird jetzt $System_2$ gelöscht, wird es zu einem inkonsistenten Zustand kommen. Deshalb wird das Schließen aller eingehenden und ausgehenden Verbindungen zu dem zu löschenden System vorausgesetzt.

Der Fall *neues System und zu löschendes System* (Abbildung 12) wird hier analog zum Fall *vorhandenes System und zu löschendes System* (Abbildung 13) behandelt, da für beide Fälle nach dem Ausführen von *SN* für das Erstellen des neuen Systems die gleichen Bedingungen gelten.

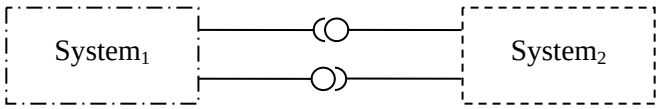


Abbildung 12: Neues System und zu löschendes System

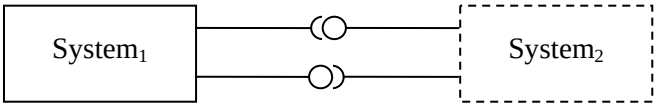


Abbildung 13: Vorhandenes System und zu löschendes System

4.2.3.1 Fall: Temporäre Integration und Löschen eines Systems

Nachdem ein konsistenter Zustand der Systemlandschaft gesichert ist kann der hier beschriebene Integrationsfall realisiert werden.

Voraussetzungen	$SN_{(1)}$, $SN_{(2)}$	
Aktionen	**CN, DT^+, CD, SD**	
Ausprägungen	$CN_{(1,2)}$, $DT^+_{(1,2)}$, $CD_{(1,2)}$, $SD_{(1)}$	(3)
	$CN_{(1,2)}$, $DT^+_{(1,2)}$, $CD_{(1,2)}$, $SD_{(2)}$	
	$CN_{(2,1)}$, $DT^+_{(2,1)}$, $CD_{(2,1)}$, $SD_{(1)}$	
	$CN_{(2,1)}$, $DT^+_{(2,1)}$, $CD_{(2,1)}$, $SD_{(2)}$	(4)

Tabelle 7: Temporäre Integration und Löschen eines Systems

Dazu wird eine neue Verbindung aufgebaut und eine oder mehrere temporäre Datenübertragungen mit darauf folgendem Schließen der Verbindung werden durchgeführt. Dies garantiert, dass alle Verbindungen geschlossen sind und das löschen des Systems kann letztendlich erfolgreich sein.

4.2.4 Einschränkungen

Mit Hilfe der eingeführten Notation für Integration wurde die Menge aller möglichen Integrationsfälle identifiziert und formal beschrieben. Allerdings sind nicht alle Ausprägungen der Fälle aus praktischer Sicht sinnvoll.

Das Löschen eines Systems wird als Integration verstanden, wenn vor dem Ausführen der Aktion *SD* mindestens ein Datenaustausch *DT* mit einem anderen System stattgefunden hat. Wenn aber der Datenaustausch nur in die Richtung des zu löschenden Systems erfolgt, bietet dieser Integrationsfall keinen praktischen Nutzen. Einerseits ist der Datentransfer zu einem System, das aus der Systemlandschaft entfernt wird, nicht sinnvoll. Andererseits gehen die im gelöschten System verwalteten Daten verloren, weil keine Migration der Daten zu einem anderen System stattgefunden hat. Solche Integrationsfälle sind die mit (3) und (4) gekennzeichneten Ausprägungen.

4.3 Anwendung des Modells auf die Gründe für Integration von ECM-Systemen

In Kapitel 3 wurden die Gründe für die Durchführung einer Integration aus der Business-Sicht zusammengefasst. Diese werden jetzt mit der eingeführten formalen Notation beschrieben. Ziel ist festzustellen, welche Integrationsfälle in Unternehmen im ECM-Bereich vorkommen. Aus der Menge aller möglichen Fälle soll anschließend eine minimale Anzahl von Integrationsszenarien abgegrenzt werden, die ausreichend sind, um die Integration mittels datenorientierter Schnittstellen im ECM-Bereich vollständig zu charakterisieren.

4.3.1 Anzahl der Instanzen reduzieren

Um die Anzahl der Instanzen in einer gegebenen Systemlandschaft zu reduzieren, müssen Systeme entfernt werden. Hier wird der in Kapitel 4.2.3.1 beschriebene Fall – *Temporäre Integration und Löschen eines Systems* angewendet. Dementsprechend ist vorerst eine Migration der gespeicherten Daten mit *CN*, DT^+, *CD* notwendig, um deren Verlust zu vermeiden. Da die Systeme den gleichen Typ haben ist keine Transformation der Daten erforderlich. Danach kann das Löschen der nicht mehr gebrauchten Systeme mit *SD* durchgeführt werden.

4.3.1.1 Migration auf vorhandenes System

Bei zwei gegebenen Systemen $System_1$ und $System_2$ kann die Anzahl der Instanzen reduziert werden, indem die Daten von $System_2$ nach $System_1$ migriert werden und anschließend $System_2$ gelöscht wird. Dies ist in Tabelle 8 sowohl mit Aktionen als auch symbolisch veranschaulicht.

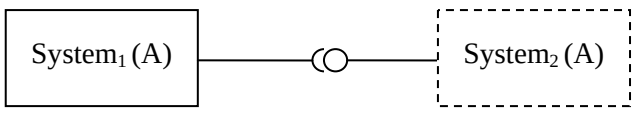
Aktionen	$CN_{(1,2)}$, $DT^+_{(1,2)}$, $CD_{(1,2)}$, $SD_{(2)}$
Symbolische Darstellung	

Tabelle 8: Migration auf vorhandenes System

4.3.1.2 Migration auf neues System

Bei zwei gegebenen Systemen *System₁* und *System₂* kann ein neues System *System₃* erstellt werden. Die Anzahl der Instanzen wird hier reduziert, indem die Daten von *System₁* und *System₂* auf *System₃* migriert werden und *System₁* und *System₂* danach gelöscht werden (Tabelle XY).

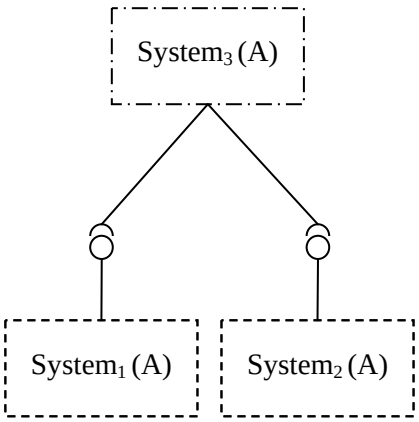
Aktionen	$SN_{(3)}$, $CN_{(3,1)}$, $DT^{+}{}_{(3,1)}$, $CD_{(3,1)}$, $CN_{(3,2)}$, $DT^{+}{}_{(3,2)}$, $CD_{(3,2)}$, $SD_{(1)}$, $SD_{(2)}$
Symbolische Darstellung	

Tabelle 9: Migration auf neues System

4.3.2 Anzahl der Produkte reduzieren

Die Anzahl der Produkte wird reduziert, indem die Anzahl der Typen der vorhandenen Systeme reduziert wird. Dies kann durch das Ersetzen aller Systeme eines bestimmten Typs mit Systemen von einem anderen schon vorhandenen Typ erreicht werden. Alternativ können die Daten von Systemen eines bestimmten Typs auf andere schon vorhandene Systeme anderen Typs migriert werden. Hier wird wieder der in Kapitel 4.2.3.1 beschriebene Fall – *Temporäre Integration und Löschen eines Systems* angewendet. Die notwendigen Aktionen sind *CN*, *DT*$^{+}$, *CD*. Da es sich um eine Migration zwischen Systemen unterschiedlichen Typs handelt, wird eine Transformation der Daten durchgeführt.

4.3.2.1 Austausch eines Systems

Für den Austausch eines Systems von Typ B mit einem System von Typ A wird zuerst ein neues System von Typ A erstellt. In Tabelle 10 ist *System₁* das zu ersetzende System und *System₂* – das neue System. Die Daten von *System₁* werden nach *System₂* migriert und transformiert. Anschließend wird *System₁* gelöscht.

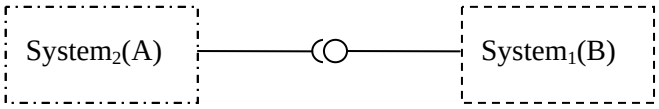
Aktionen	$SN_{(2)}$, $CN_{(2,1)}$, $DT^{+}_{(2,1)}$, $CD_{(2,1)}$, $SD_{(1)}$
Symbolische Darstellung	

Tabelle 10: Austausch eines Systems

4.3.2.2 Migration auf vorhandenes System

Die Verringerung der Anzahl der Produkte kann durch das Migrieren der Daten von Systemen eines bestimmten Typs auf andere vorhandene Systeme anderen Typs erreicht werden. In Tabelle 11 werden die Daten von *System₂* nach *System₁* migriert und transformiert. Anschließend wird *System₂* gelöscht.

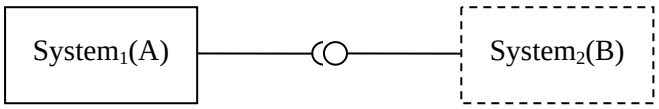
Aktionen	$CN_{(1,2)}$, $DT^{+}_{(1,2)}$, $CD_{(1,2)}$, $SD_{(2)}$
Symbolische Darstellung	

Tabelle 11: Migration auf vorhandenes System

4.3.3 Austausch eines Produktes

Bei dem Austausch eines Produktes handelt es sich um den Austausch von einem System eines bestimmten Typs mit einem System eines anderen Typs. Dies wurde schon in Kapitel 4.3.2.1 Austausch eines Systems behandelt.

4.3.4 Mehrfache Nutzung von Inhalten

Mehrfache Nutzung von Inhalten bedeutet, dass ein System die datenorientierten Schnittstellen eines anderen Systems dauerhaft nutzt. Hier wird der in 4.2.1.1 beschriebene Fall – *Persistente Integration* verwendet. Demnach baut ein System eine Verbindung mit anderen Systemen auf. Die Verbindung bleibt erhalten und wird nicht geschlossen, so dass der Datenaustausch jederzeit erfolgen kann. *CN, DT** sind die erforderlichen Aktionen für die Realisierung dieses Falls.

In Tabelle 12 ist die Verbindung von zwei Systemen gleichen Typs dargestellt. *System₁* kann stets Daten von *System₂* lesen, ohne dass eine Transformation notwendig ist, da die beiden Systeme den Gleichen Typ haben.

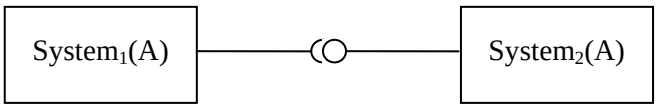
Aktionen	$CN_{(1,2)}$, $DT^*_{(1,2)}$
Symbolische Darstellung	

Tabelle 12: Punkt-zu-Punkt Integration von Systemen gleichen Typs

Die Verbindung von zwei Systemen, die einen unterschiedlichen Typ haben, ist in Tabelle 13 beschrieben. Bei einem Datentransfer ist eine Transformation der Inhalte notwendig.

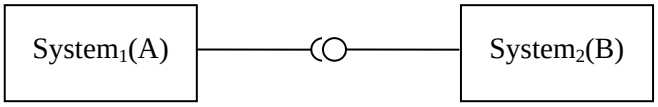
Aktionen	$CN_{(1,2)}$, $DT^*_{(1,2)}$
Symbolische Darstellung	

Tabelle 13: Punkt-zu-Punkt Integration von Systemen unterschiedlichen Typs

Eine Kombination von den vorigen zwei Fällen ist in Tabelle 14 beschrieben. In diesem Fall kann ein System Inhalte von mehreren anderen Systemen beziehen, die nicht unbedingt vom gleichen Typ sein können.

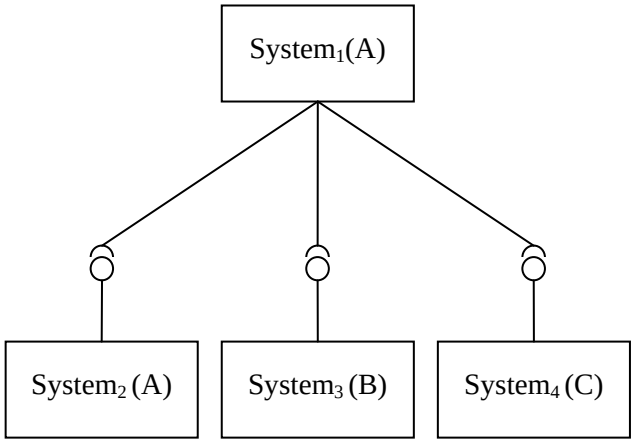
Aktionen	$CN_{(1,2)}$, $DT^*_{(1,2)}$; $CN_{(1,3)}$, $D\ T^*_{(1,3)}$; $CN_{(1,4)}$, $D\ T^*_{(1,4)}$
Symbolische Darstellung	

Tabelle 14: Punkt-zu-Punkt Integration von Systemen gemischten Typs

Jetzt wird der Ansatz von Tabelle 14 erweitert. Ein weiteres führendes System wird mit dem System verbunden, das eine föderierte Sicht auf eine Gruppe von Systemen anbietet.

In [Co06] wird das Konzept für ein föderiertes System vorgestellt. Ein *föderierendes System* wird mit mehreren anderen Systemen verbunden. Anfragen an dieses System werden in einer für das aufrufende System transparenten Art und Weise aber lediglich an das jeweils korrekte System, wo die geforderten Daten gespeichert sind, weitergeleitet.

In Tabelle 15 ist *System*$_2$ ein föderiertes System, dass für *System*$_1$ Daten von *System*$_3$, *System*$_4$ und *System*$_5$ bereitstellt. Auf diese Weise kann ein einheitlicher Zugriff auf Systeme unterschiedlichen Typs ermöglicht werden. Das föderierte System sorgt dann für eventuelle Transformationen der Daten. Darüber hinaus bleibt der tatsächliche Speicherort der Inhalte transparent.

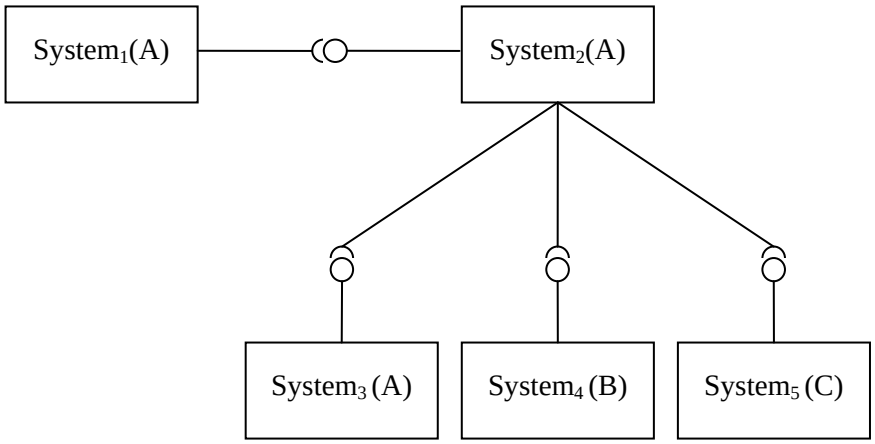
Aktionen	$CN_{(1,2)}$, $DT^*_{(1,2)}$; [$CN_{(2,3)}$, $DT^*_{(2,3)}$; $CN_{(2,4)}$, $DT^*_{(2,4)}$; $CN_{(2,5)}$, $DT^*_{(2,5)}$]
Symbolische Darstellung	

Tabelle 15: Integration mittels föderierenden Systems

4.3.5 Systemübergreifende Funktionen

Systemübergreifende Funktionen wie Suche und statistische Auswertungen setzen Zugriff auf mehrere Systeme voraus. Der in Kapitel 4.2.1.1 beschriebene Fall – *Persistente Integration* lässt sich auch hier verwenden. Dabei kann ein System entweder Verbindungen zu den zu untersuchenden Systemen aufbauen oder die Dienste eines föderierenden Systems benutzen. Die zwei Möglichkeiten sind entsprechend in Tabelle 14 und Tabelle 15 dargestellt.

4.4 Ableitung von typischen technischen Integrationsszenarien

In diesem Kapitel wurden die Gründe für die Durchführung einer Integration in ECM-Bereich (Kapitel 3) mit Hilfe der eingeführten formalen Notation analysiert. Es lassen sich zwei Arten von Szenarien identifizieren: Szenarien, die eine Migration von Daten mittels einer temporären Integration *CN*, *DT*$^+$, *CD* realisieren, und Szenarien, die eine dauerhafte Funktionsintegration mittels einer persistenten Integration *CN*, *DT** bewirken.

Die temporären Integrationsszenarien, die in den Fällen von Reduktion der Anzahl der Instanzen und der Produkte sowie beim Austausch eines Produktes angewendet werden, werden in der weiteren Betrachtung als *Datenmigration* definiert. Dabei spielt der Typ der Systeme eine wesentliche Rolle. Zwecks der Einfachheit der eingeführten Notation wurde angenommen, dass die Aktion *DT* eine Datentransformation automatisch durchführt. In Kapitel 5.3.4 wird gezeigt, dass die Datentransformation ein wesentlicher Punkt bei der Realisierung einer Integrationslösung ist.

Die persistenten Integrationsszenarien, die eine bestehende Verbindung erfordern - die mehrfache Nutzung von Inhalten und die systemübergreifende Funktionen werden hier als *Funktionsintegration* definiert. Dabei wird zwischen der Punkt-zu-Punkt Funktionsintegration, bei der ein System Inhalte von mehreren anderen Systemen bezieht, und der Funktionsintegration mittels eines föderierenden Systems unterschieden.

Mit Hilfe der eingeführten Notation wurde gezeigt, dass die Integrationsszenarien aus der Business-Sicht mittels Funktionsintegration und Datenmigration realisiert werden können. Die Ergebnisse der Untersuchung sind in Tabelle 16, Tabelle 17 und Tabelle 18 zusammengefasst.

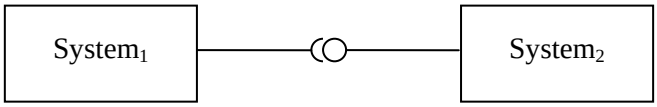
Szenario	Aktionen	Symbolische Darstellung
Datenmigration	CN, DT^{+}, CD	

Tabelle 16: Datenmigration

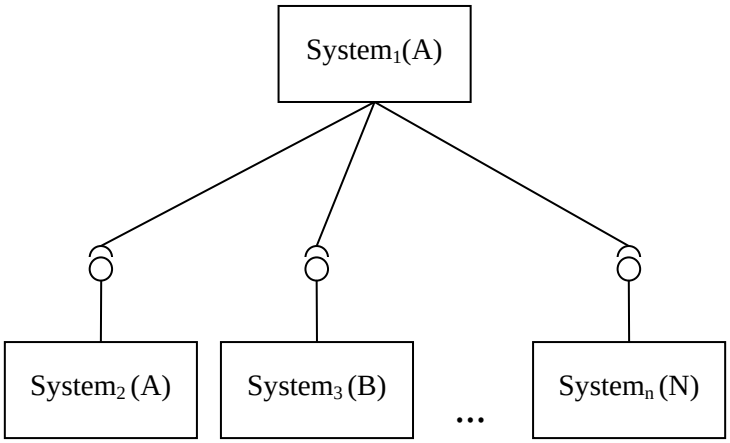
Szenario	Aktionen	Symbolische Darstellung
Punkt-zu-Punkt Funktions-integration	CN, DT*	

Tabelle 17: Punkt-zu-Punkt Funktionsintegration

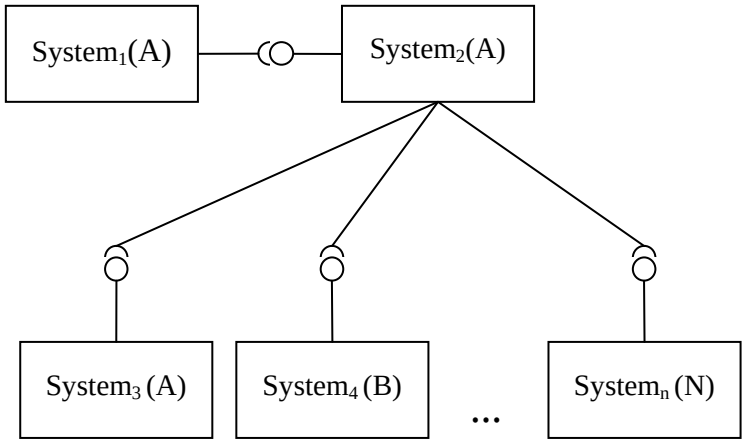
Szenario	Aktionen	Symbolische Darstellung
Funktions-integration mittels eines föderierenden Systems	CN, DT*	

Tabelle 18: Funktionsintegration mittels eines föderierenden Systems

Somit wird die Integration mittels datenorientierter Schnittstellen im ECM-Bereich durch die Szenarien Datenmigration, Punkt-zu-Punkt Funktionsintegration und Funktionsintegration mittels eines föderierenden Systems vollständig charakterisiert.

5 Technische Realisierung von Integration

In Kapitel 4 wurde ein Model für Integration von ECM-Systemen unter Verwendung von datenorientierten Schnittstellen eingeführt. Die Verbindung von Systemen wurde auf dieser abstrakten Ebene mit Hilfe der Lollipop-Notation dargestellt. Im Folgenden wird der Fokus auf die technische Umsetzung der Integration gelegt. Es wird näher untersucht, wie die Verbindung zwischen ECM-Systemen umgesetzt werden kann.

5.1 Application Programming Interface

5.1.1 Definition

Die technische Kopplung von Systemen wird über Funktionsschnittstellen (Application Programming Interface, API) realisiert. Eine API ist eine Schnittstelle, die von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird [Wi08f]. Eine API ist somit eine Sammlung von Methoden, die zum Aufrufen von anderen Systemen angeboten wird. Von Sicht der Integration sind nur die Schnittstellen interessant, die für andere Systeme sichtbar sind. Eine API besteht aus der Schnittstelle selbst und der Dokumentation über die Spezifikation der Schnittstelle [THB05]. In der Regel wird eine Schnittstelle in Form einer Funktionsbibliothek zur Verfügung gestellt. Die Dokumentation einer API beschreibt die Funktionsweise der Schnittstelle und die zulässigen Werte der Parameter und der Rückgabewerte der Methoden. Mögliche Fehlermeldungen werden auch behandelt.

Nach [Si06] wird jede Schnittstelle als eine Menge von Methoden mit folgenden Eigenschaften definiert:

- Syntax
- Semantik
- Protokoll
- Nichtfunktionale Eigenschaften

Die Syntax einer Schnittstelle definiert den Kopf einer Methode – die Rückgabewerte und die Argumente. In [MM02] werden zwei Arten von Methoden unterschieden: Kommandos und Abfragen. Kommandos (command) ändern den Zustand einer Systemkomponente, Abfragen (query) lassen ihn unverändert. Die Semantik einer Schnittstelle legt fest, was die Kommandos bewirken und was die Abfragen bedeuten.

Das Protokoll ist eine Vereinbarung, nach der die Verbindung, Kommunikation und Datenübertragung zwischen den Systemen abläuft [Wi08g]. Das Protokoll kann z.B. festlegen, ob eine Schnittstelle synchron oder asynchron ist. Als nichtfunktionale

Eigenschaften können die Performance und die Robustheit der Schnittstelle angesehen werden.

In dieser Hinsicht sind zwei Schnittstellen vom gleichen Typ, wenn ihre Eigenschaften Syntax, Semantik und Protokoll übereinstimmen.

Eine API ist gemäß obiger Definition die Spezifikation einer Schnittstelle. Sie ist von der Implementierung unabhängig. Somit können verschiedene Implementierungen einer Schnittstelle desselben Typs existieren, die gegeneinander ausgetaucht werden können. Implementierungen dürfen sich nur in Eigenschaften unterscheiden, die nicht zu den Vereinbarungen der API gehören.

5.1.2 Abgrenzung von API und Protokoll

Im allgemeinen Sprachgebrauch werden die Begriffe API und Protokoll oft vermischt und nicht klar abgegrenzt.

In [Wi08g] wird ein Protokoll als Regeln definiert, welche das Format, den Inhalt, die Bedeutung und die Reihenfolge gesendeter Nachrichten zwischen verschiedenen Instanzen festlegen. Ein Protokoll regelt den Ablauf, und stellt gleichzeitig dessen Dokumentation sicher. In dieser Arbeit wird der Begriff Protokoll-Schnittstelle in diesem Zusammenhang verwendet. Beispiele für Protokoll-Schnittstellen sind WebDAV [Wi08h], FTP [Wi08i] und CIFS [Wi08j].

Der Begriff API wird hier gemäß der in Kapitel 5.1.1 eingeführten Definition verwendet. Eine API wird somit von Entwicklern zum Erstellen von Programmen in einer Programmiersprache benutzt. Dementsprechend liegt der Fokus der vorliegenden Arbeit auf der programmatischen Verwendung von APIs.

5.2 Realisierung von Integration

5.2.1 Begriff der Verbindung

Der in der Arbeit benutzte Begriff der Verbindung lehnt sich an der UML-Definition der Lollipop-Notation [Ru05] an. In Abbildung 14 ist die Verbindung von $System_1$ und $System_2$ symbolisch dargestellt.

$System_2$ definiert durch das nicht ausgefüllte Kreissymbol, in UML als Ball bezeichnet, dass es eine gegebene Schnittstelle implementiert. In der vorliegenden Arbeit wird dieses Symbol als *angebotene Schnittstelle* bezeichnet.

$System_1$ deutet durch das Halbkreissymbol, auch Socket genannt, dass es die angegebene Schnittstelle zur Erfüllung seiner Funktionalität benötigt. Dieses Symbol wird hier als *angeforderte Schnittstelle* bezeichnet.

Der Begriff der Verbindung wird durch die Kombination von einer angebotenen und einer angeforderten Schnittstelle definiert. Somit wird das technische Ineinandergfreifen des Anbieters und des zugehörigen Benutzers einer Schnittstelle veranschaulicht.

Weiterhin werden noch einige Begriffe geklärt, die in der Literatur nicht ausreichend diskutiert aber für das Verständnis der vorliegenden Arbeit notwendig sind.

Die explizite Aufteilung in angebotenen und angeforderten Schnittstellen wird eingeführt, um die Unterscheidung zwischen einem Anbieter und einem Benutzer einer Schnittstelle zu verdeutlichen. Insofern bietet ein Anbieter mittels einer angebotenen Schnittstelle Methoden an, die von einem Benutzer durch die angeforderte Schnittstelle aufgerufen werden können. Ein Benutzer wird hier als *führendes System* und der Anbieter als *integriertes System* bezeichnet. Somit wird eine Verbindung durch die Richtung der Methodenaufrufe charakterisiert. Im Sinne der Lollipop-Notation bedeutet dies, dass die Richtung der Verbindung eindeutig durch die Kombination von einem Halbkreissymbol und einem Kreissymbol dargestellt wird. Das System mit dem Halbkreissymbol führt Methodenaufrufe auf das System mit dem Kreissymbol aus.

5.2.2 Vorgehensweisen bei Integration

Eine Integrationslösung wird durch die Verbindung von Systemen durchgeführt. Im Folgenden werden drei Vorgehensweisen für die Realisierung von Integration vorgestellt, die aus der direkten Anwendung des im Kapitel 5.2.1 definierten Begriffs der Verbindung abgeleitet werden.

5.2.2.1 Direkte Verbindung

Die Integration von Systemen kann am einfachsten durch die direkte Verbindung einer angeforderten mit einer angebotenen Schnittstelle vom gleichen Typ realisiert werden. In diesem Fall wird ein System speziell für die Benutzung einer Schnittstelle entworfen. In Abbildung 14 ist die Verbindung von $System_1$ und $System_2$ über die von $System_2$ angebotene Funktionsschnittstelle *$System_2$ API* symbolisch dargestellt. Folglich kann $System_1$ alle angebotenen Methoden von der *$System_2$ API* aufrufen.

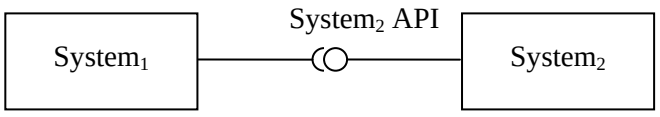


Abbildung 14: Direkte Verbindung

5.2.2.2 Konnektor

Die direkte Verbindung von zwei Systemen ist nicht möglich, wenn die angeforderte und die angebotene Schnittstelle einen unterschiedlichen Typ haben. Dies kommt dann vor, wenn ein System für die Benutzung einer Remote-Schnittstelle vom bestimmten Typ entwickelt worden ist. Eine Schnittstelle von einem anderen Typ wird jedoch angeboten. In diesem Fall wird eine Integrationskomponente entwickelt, die die zwei Systeme mit den jeweiligen Schnittstellen verbindet und die Methodenaufrufe von der angeforderten Schnittstelle auf Methodenaufrufe der angebotenen Schnittstelle übersetzt. Abbildung 15 zeigt die symbolische Darstellung einer

solchen Integrationskomponente. $System_1$ ist für die Nutzung der *Standard API* entwickelt worden. Die Integrationskomponente vermittelt die Aufrufe von $System_1$ an der vom $System_2$ angebotenen Schnittstelle *$System_2$ API*. Diese Komponente wird im weiteren Verlauf der Arbeit als *Konnektor* bezeichnet.

Abbildung 15: Konnektor

5.2.2.3 Migrationskomponente

Sollen zwei Systeme nachträglich integriert werden, die nicht für die Verbindung miteinander entworfen sind, wird in der Regel eine Integrationskomponente entwickelt, die die beiden Systeme über deren datenorientierten Schnittstellen verbindet. In weiterem Verlauf der Arbeit wird diese Komponente *Migrationskomponente* genannt. Abbildung 16 zeigt die symbolische Darstellung einer Migrationskomponente, die sowohl $System_1$ API als auch $System_2$ API nutzen kann und auf diese Weise die Kommunikation zwischen den beiden Systemen ermöglichen kann. Die Migrationskomponente ist in diesem Fall ein führendes System und somit Initiator der Verbindungen und des Datentransfers zwischen den Systemen.

Abbildung 16: Migragionskomponente

5.3 Technische Schwierigkeiten bei Integration

Der Entwurf und die Entwicklung einer Integrationslösung zur Kopplung von Systemen ist nicht unbedingt eine leichte Aufgabe. Wegen der Heterogenität der Systeme ergeben sich zahlreiche technische Schwierigkeiten. Im Folgenden wird auf diese Hürden näher eingegangen.

5.3.1 Verschiedene Programmiersprachen

Die in dieser Arbeit betrachteten ECM-Systeme basieren auf Microsoft .NET oder Sun Microsystems J2EE-Komponententechnologie. Die Systeme sind demzufolge in verschiedenen Programmiersprachen implementiert. Die Interoperabilität mittels Funktionsschnittstellen ist daher nicht direkt gegeben. Es existieren Bridge-Lösungen wie Ja.Net, J-Integra, JNBridgePro [Pe04], die die Interoperabilität zwischen den Plattformen gewährleisten. Da der Fokus der vorliegenden Arbeit auf Java Schnittstellen liegt und alle

betrachteten Systeme ihre APIs über Java Schnittstellen anbieten, wird die Analyse solcher Interoperabilitätsprobleme unterlassen und auf die angegebene Literaturquelle verwiesen.

5.3.2 Verschiedene APIs

Bei der Integration von ECM-Systemen ist die Qualität und Kompatibilität der von den Systemen bereitgestellten APIs entscheidend. Da der Produktkern in der Regel nicht verändert werden kann, muss Integration über vorhandene Schnittstellen erfolgen. Je einfacher und direkter diese Schnittstellen verbunden werden können, desto geringer ist der Aufwand für die Realisierung einer Integrationslösung.

Die Durchführung einer Integrationslösung wird sowohl bei der direkten Integration, als auch bei einer Integration mittels einer Integrationskomponente (vgl. Kapitel 5.2) wesentlich erschwert, wenn die Systeme APIs unterschiedlichen Typs anbieten. Der Idealzustand in dieser Hinsicht ist eine einheitliche Systemlandschaft mit durchgehend einheitlichen Schnittstellen. Tatsächlich wird dies in den größeren Unternehmen fast nie erreicht, aufgrund der ständigen vielfältigen Veränderungen, denen die IT-Infrastruktur unterworfen ist. Häufig werden in Unternehmen unterschiedliche ECM-Produkte betrieben, welche typischerweise eine eigene proprietäre API mit unterschiedlichen Funktionen bereitstellen.

Die unterschiedlichen APIs sind eine der Hauptschwierigkeiten bei dem Entwurf und Implementierung einer Integrationslösung. Zum einen stellt jede API verschiedene Methoden, die eine unterschiedliche Aufruflogik, Syntax und Semantik haben, bereit. Zum anderen lässt sich die gewünschte Funktionalität nicht mit jeder API realisieren. Darüber hinaus benutzt jede Remote-API ein unterschiedliches Protokoll für die Kommunikation. Die Kommunikationsmechanismen werden in Kapitel 5.4 ausführlicher behandelt.

5.3.2.1 Verschiedene APIs bei direkter Verbindung

Die *direkte Verbindung* von Systemen (Kapitel 5.2.2) kann unter der Verwendung unterschiedlicher APIs realisiert werden. Der JSR 170 API Quellcode-Ausschnitt in Tabelle 19 zeigt die Implementierung einer direkten Verbindung mit einem Java Content Repository. Als Ergebnis wird das Attribut *titel* der Inhalte in einem Verzeichnis ausgegeben.

Die Integration wird realisiert, indem zuerst die RMI-Verbindung mit dem entfernten Java-Content-Repository aufgebaut wird. Für die RMI-Kommunikation wird die Apache Jackrabbit Bibliothek JSR 170 RMI [Ap08b] benutzt. Die Authentifizierung enthält Login, Passwort und auch den Namen des verwendeten Workspaces *da_workspace*. Nachdem eine gültige Session erworben wird, kann zum gewünschten Pfad navigiert werden. Für alle Nodes, die sich unter diesem Node befinden, wird das Property mit dem Namen *titel* mit der Methode `getProperty("da:title")` ausgelesen und danach als Zeichenkette ausgegeben.

```
// Create a new repository client
ClientRepositoryFactory factory = new ClientRepositoryFactory();
Repository repository =
     factory.getRepository("//localhost:1099/jcr");

// Create a new repository session, after authenticating
Session session = repository.login(new SimpleCredentials("admin",
      "admin".toCharArray()),"da_workspace");

// Obtain the root node and get the node by path
Node node = session.getRootNode().getNode("testfolder/content");

// Iterrate the children and print property “Title”
NodeIterator children = node.getNodes();
while (children.hasNext()) {
      Node child = children.nextNode().getNode("jcr:content");
      Property property = child.getProperty("da:title");
      String title = property.getString();
      System.out.println(title);
}
session.logout();
```

Tabelle 19: Quellcodebeispiel – direkte Verbindung zu JCR

In Tabelle 20 wird die gleiche Funktionalität mit einer Documentum Instanz mittels der DFC API (vgl. Kapitel 6.3.3) realisiert. Abbildung 17 zeigt das UML-Modell der Datenstruktur bei einer Standartinstallation von Documentum.

Für die Verbindung mit der Documentum Instanz wird ein lokaler Client erzeugt. Nach der Authentifizierung mit Username und Passwort und Auswahl der Identität *da_workspace* wird eine gültige Session erworben. In Documentum kann mit der Methode `getFolderByPath(String Path)` direkt zu dem gewünschten Verzeichnis navigiert werden, wobei *IDfFolder* ein vordefinierter Objekttyp ist. Das Atribut *r_object_id* wird in Documentum für alle Objekte bei deren Erstellung eindeutig vergeben. Mit der Methode `folder.getContents("r_object_id")` werden die Ids aller Objekte in dem gewünschten Verzeichnis ausgelesen. Danach kann zu jedem Objekt mittels der Id der Titel mit der Methode *getTitle()* ausgelesen werden.

```
//create Client objects
IDfClient client = DfClient.getLocalClient();

//create a Session Manager object
IDfSessionManager sMgr = client.newSessionManager();

//create an IDfLoginInfo object named loginInfoObj
IDfLoginInfo loginInfo = new DfLoginInfo();
loginInfo.setUser("admin");
```

```
loginInfo.setPassword("admin");
sMgr.setIdentity("da_workspace", loginInfo);

IDfSession session = sMgr.getSession("da_workspace");

// Get the Folder by path
IDfFolder folder = session.getFolderByPath("/testfolder/content");

// Iterrate the children and print property “title”
IDfCollection children = folder.getContents("r_object_id");
while(children.next()){
      IDfId childId = children.getId("r_object_id");
      IDfDocument doc = (IDfDocument)session.getObject(childId);
      String title = doc.getTitle();
      System.out.println(title);
}
children.close();
sMgr.release(session);
```

Tabelle 20: Quellcodebeispiel – direkte Verbindung zu Documentum

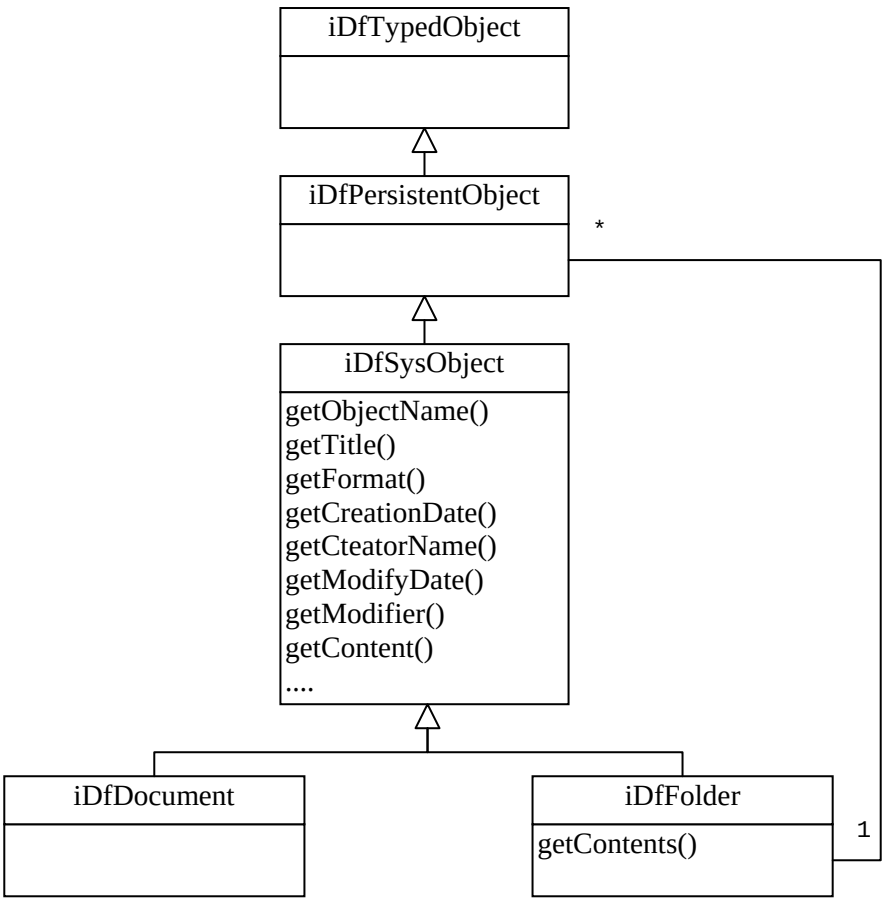


Abbildung 17: Documentum – Content-Modell (UML-Klassendiagramm)

Wie die obigen Quellcode-Beispiele zeigen, kann die gleiche Funktionalität mit unterschiedlichen APIs erreicht werden. Die APIs unterscheiden sich in der Art und Weise, in der ähnliche Methodenaufrufe realisiert werden. Die vorgestellten APIs weisen Unterschiede in der Aufruflogik auf. Daraus resultiert eine unterschiedliche Syntax für semantisch ähnliche Methodenaufrufe. Das Attribut *titel* ist bei der JSR 170 API als ein Property von Typ String mit dem Namen *da:titel* realisiert, dessen Wert mit einer generischen Methode `getProperty("da:title")` für ein Node ausgelesen wird. Bei der DFC API existiert bereits für das Objecttyp IDfDocument eine vordefinierte Methode `getTitle()`, die verwendet werden kann.

Die APIs unterscheiden sich weiterhin in dem unterschiedlichen Aufwand für die Erstellung der Laufzeitumgebung. Bei JSR 170 müssen nur die entsprechenden Bibliotheken in dem Java Classpath aufgenommen werden. Bei der DFC API dagegen muss eine Installation der Laufzeitumgebung mit entsprechender Konfiguration erfolgen [Em02].

Immerhin ist der Aufwand für die Implementierung einer direkten Verbindung abhängig von dem Typ der angebotenen APIs. Sollte die direkte Verbindung zu einer Mehrzahl von Systemen realisiert werden, wächst auch die Anzahl der zu verwendeten APIs, dementsprechend auch der Aufwand.

5.3.2.2 Verschiedene APIs bei Konnektor

Der in Kapitel 5.2.2.2 beschriebene *Konnektor* kann die Anzahl der verwendeten APIs reduzieren, indem als angebotene Schnittstelle eine Standard-API bereitgestellt wird. Ein Beispiel dafür ist der JCR-Documentum-Konnektor (Kapitel 6.3.4), der die DFC API in JSR 170 API übersetzt. In diesem Fall ist die Menge der zur Verfügung stehenden Methoden begrenzt in Vergleich zu der direkten Nutzung der DFC API. Die JSR 170 API ist eine API für standardisierten Zugriff auf Java Content Repositories, und die DFC API ist eine API, die die gesamte Funktionalität eines ECM-Systems beretstellt. Infolgedessen kann nur die Nutzung der kleinsten Schnittmenge aller möglichen Funktionen der beiden APIs erreicht werden.

Tabelle 21 zeigt die Verbindung zu einer Documentum Instanz mittels des JCR-Documentum-Konnektors. In diesem Fall lässt sich die gewünschte Funktionalität sowohl mit der nativen DFC API, als auch mit der Standard JSR 170 API realisieren.

```
// Create a new repository client
      ClientRepositoryFactory factory = new ClientRepositoryFactory();
      Repository repository =
           factory.getRepository("//localhost:1099/jcr");

      // Create a new repository session, after authenticating
      Session session = repository.login(new SimpleCredentials("admin",
            "admin".toCharArray()),"documentum_workspace");
```

```
// Obtain the root node and get the node by path
Node node = session.getRootNode().getNode("testfolder/content");
// Iterrate the children and print property “Title”
NodeIterator children = node.getNodes();
while (children.hasNext()) {
      Node child = children.nextNode();
      Property property = child.getProperty("dctm:title");
      String title = property.getString();
      System.out.println(title);

}

session.logout();
```

Tabelle 21: Quellcodebeispiel – direkte Verbindung zu JCR-Documentum-Konnektor

5.3.2.3 Verschiedene APIs bei Migrationskomponente

Der Implementierungsaufwand einer *Migrationskomponente* (Kapitel 5.2.2.3) hängt analog von den verwendeten APIs ab. Tabelle 23 zeigt ein Quellcode-Beispiel für die Implementierung einer Migrationskomponente. Die Komponente baut eine Verbindung zu einer Documentum Instanz und zu einem JCR auf. Alle Inhalte von einem gewünschten Verzeichnis werden von Documentum in das JCR migriert. Das JCR wird von der in Kapitel 2.4.6 entworfene Beispielsanwenundung JCR DA verwendet. Die Anwendung wird hier gebraucht, um die Darstellund der Problematik der unterdschiedlichen anwendugsspezifische Datenmodelle zu erleichtern. Die JCR DA-Anwendung hat somit nur eine Funktion – das Speichern von Daten.

Documentum iDfDocument	JCR DA nt:file	JCR DA da:content
objectName	Name des Nodes nt:file	
title		da:title
mIMEType		jcr:mimeType
creationDate		da:created
modiefyDate		jcr:lastModified
modiefier		da:lastModifiedBy
content		jcr:data

Tabelle 22: Metadatentrasformation - Documentum nach JCR DA

```
...
//sessions already obtained
IDfFolder srcFolder = sessionDcm.getFolderByPath("/mmik_test/content");
Node destFolder = sessionJCR.getRootNode().getNode("testfolder/content");


//read folder contents by id
children = srcFolder.getContents("r_object_id");

while (children.next()) {
      //for each id get a content object
      IDfId childId = children.getId("r_object_id");
      IDfPersistentObject perstObject = sessionDcm.getObject(childId);

      if(perstObject instanceof IDfDocument){
            //read source content
            IDfDocument doc = (IDfDocument)perstObject;
            String contentName = doc.getObjectName();
            String contentTitle = doc.getTitle();
            String contentMimeType = doc.getFormat().getMIMEType();
            Date contentCreationDate = doc.getCreationDate().getDate();
            String contentCreatorName = doc.getCreatorName();
            Date contentLastModifiedDate = doc.getModifyDate().getDate();
            String contentModifierName = doc.getModifier();
            contentInputStream = doc.getContent();

            //write content to destination
            Node file = destFolder.addNode(contentName, "nt:file");
            Node contentNode = file
                  .addNode("jcr:content", "da:da_content");
            contentNode.setProperty("da:title", contentTitle);
            contentNode.setProperty("jcr:mimeType", contentMimeType);
            calendar.setTime(contentCreationDate);
            contentNode.setProperty("da:created", calendar);
            contentNode.setProperty("da:createdBy", contentCreatorName);
            calendar.setTime(contentLastModifiedDate);
            contentNode.setProperty("jcr:lastModified", calendar);
            contentNode.setProperty("da:lastModifiedBy",
                  contentModifierName);
            contentNode.setProperty("jcr:data", contentInputStream);
      }
}
//Save JCR session

sessionJCR.save();

//close sessions

...
```

Tabelle 23: Quellcodebeispiel Migrationskomponente

Bei der Migration der Inhalte findet eine Transformation der Metadaten statt. Die DFC-Klasse *iDfDocument* wird auf ein JCR-Node von Typ nt:file und einem darunterliegenden JCR-Node von Typ *da:content* abgebildet. Die Attribute werden entsprechend der Tabelle 22 abgebildet.

5.3.2.4 Schlussbetrachtung

In der Regel werden Integrationslösungen von Entwicklerteams implementiert. Die technische Expertise, die ein Team mitbringt, ist entscheidend für die Realisierung eines Integrationsprojekts. Das Team muss die Systeme und deren angebotenen Schnittstellen kennen, um eine korrekte Lösung entwerfen zu können. Fehlende Kompetenzen können entweder durch Schulungen, die von den Herstellern der Systeme oder deren Vertriebspartner kostenpflichtig angeboten werden, oder durch Einbeziehung externer Berater, die über die entsprechenden Kenntnissen verfügen, ausgeglichen werden. Beide Ansätze wirken sich negativ auf die Dauer und die Kosten eines Integrationsprojekts. Darüber hinaus kann bei Altsystemen, die nicht mehr supported werden, der Aufwand so hoch sein, dass eine Integrationslösung nicht gerechtfertig werden kann und eine Migration der Inhalte zu einem modernen System notwendig wird. Der Begriff Content-Silo wird in Zusammenhang mit solchen Systemen verwendet, die die gespeicherten Inhalte isolieren und sie nicht für die Weiterverwendung bereitstellen können.

Dementsprechend kann der Aufwand für die Realisierung einer Integrationslösung reduziert werden, wenn standardisierte APIs wie die JSR 170 API für Remote-Zugriff auf das Content-Repository von ECM-Systemen bereitgestellt werden.

5.3.3 Verschiedene Versionen

ECM-Systeme werden iterativ weiterentwickelt und erweitert. Neue Produkte oder Produktversionen werden kontinuierlich auf den Markt gebracht. Weiterhin haben sich die Entwicklungszyklen für Softwareprodukte auf ein Jahr reduziert [Ka03i]. Verschiedene Versionen eines Produktes stellen daher unterschiedliche Versionen der Remote-APIs bereit.

Die direkte Interoperabilität zwischen Systemen verschiedener Versionen kann nur dann erreicht werden, wenn die Abwärtskompatibilität der APIs gegeben ist. Da meistens neuere Produktversionen erweiterte Funktionalitäten anbieten, wird insofern nicht möglich sein, diese mit der alten API zu benutzen.

Die direkte Nutzung verschiedener Versionen einer API kann wegen rein technischer Gründe nicht möglich sein. Wird für die entfernte Kommunikation RMI (Kapitel 5.4.3) benutzt, müssen sowohl das führende (der Client) als auch das zu integrierende System (der Server) über die gleiche Version der Schnittstelle verfügen, da sonst die Serializierung der Objekte nicht möglich wäre.

Wenn die zwei Versionen der API inkompatibel sind und die direkte Verbindung nicht möglich ist, sollen sie wie APIs unterschiedlichen Typs behandelt werden.

5.3.4 Verschiedene Datenmodelle

Jedes ECM-System hat ein eigenes internes Datenmodell. In der JSR 170 Spezifikation werden außer den vordefinierten Objekttypen insofern keine Vereinbarungen für die Datenmodellierung getroffen.

Wird Content zwischen Systemen mit unterschiedlichen internen Datenmodellen migriert, auch wenn die Systeme die gleiche API bereitstellen, müssen die Inhalte und die Metadaten in das Modell des Zielsystems transformiert werden, damit das Zielsystem die Daten sinnvoll nutzen kann.

JCR Documentum dctm:dm_document	JCR Documentum dctm:dmr_content	JCR DA nt:file	JCR DA da:content
Name des Nodes dctm:dm_document		Name des Nodes nt:file	
dctm:title			da:title
	dctm:content_type		jcr:mimeType
jcr:created			da:created
dctm:r_creator_name			da:createdBy
	jcr:lastModified		da:lastModified
dctm:r_modifier			da:lastModifiedBy
...	jcr:data		jcr:data

Tabelle 24: Metadatentrasformation – JCR-Documentum-Konnektor nach JCR DA

Das in Abbildung 18 dargestellte JCR-Documentum-UML-Modell unterscheidet sich von dem in Abbildung 7 dargestellten DA-UML-Modell. In dem JCR-Documentum-Modell ist *title* ein Attribut vom Objekt *dctm:dm_document*, das *nt:file* erweitert. In dem JCR DA-Modell ist dagegen *tittle* ein Attribut vom Objekt *da:content*, das *nt:resource* erweitert. Dementsprechend werden alle Documentum-Metadaten als Attribute von *dctm:dm_document* modelliert. In dem JCR DA-Modell liegen die Metadaten eine Ebene tiefer in dem *da:content* Object. Dies ist beim Vergleich der jeweiligen symbolischen Darstellungen in Abbildung 8 und Abbildung 19 der Modelle ersichtlich. Der vollständige Vergleich der Objektzugehörigkeit der Attribute der beiden Modelle ist in Tabelle 24 zusammengefasst.

Ferner ist in Kapitel 5.3.2.3 eine Migration unter der Verwendung verschiedener APIs dargestellt. Die unterschiedlichen Modelle der Anwendungen erzwingen in diesem Fall eine

Transformation der Metadaten. Tabelle 22 zeigt die Abbildung der Attribute der Documentum- Klasse iDfDocument auf die Attribute des DA-Modells.

Die Definition der Objektattribute kann in JSR 170 mittels des Interfaces *NodeTypeManager* bezogen werden. Das Interface verfügt über Methoden zum Auslesen von Node-Typen, die in dem Repository registriert sind.

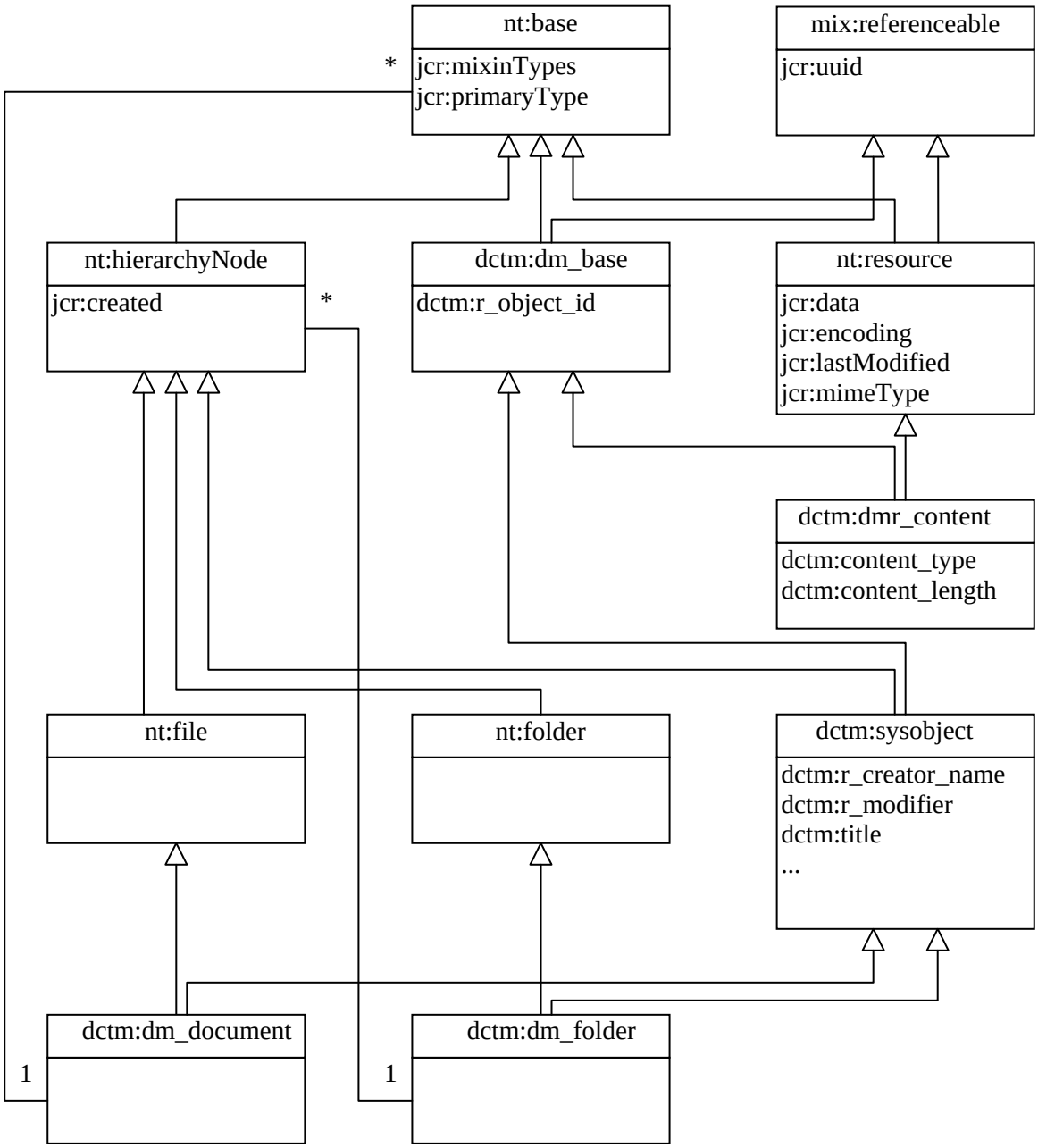


Abbildung 18: JCR-Documentum-Konnektor – Content-Modell (UML Klassendiagramm)

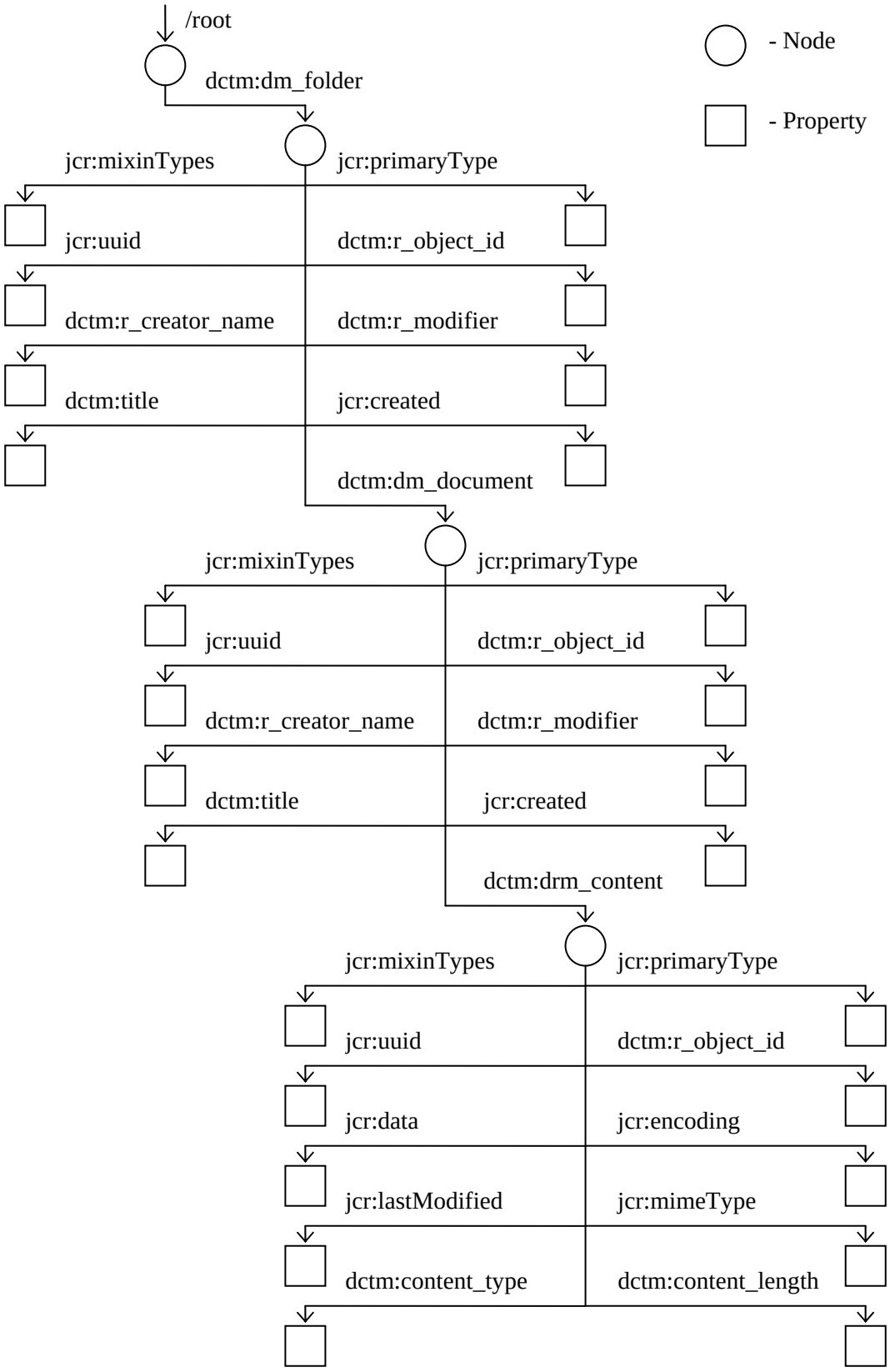


Abbildung 19: JCR-Documentum-Konnektor – symbolische Darstellung der Struktur des Content-Modells

5.3.5 Transport

Damit zwei Systeme über ein Netzwerk kommunizieren können, müssen die bereitgestellten APIs entfernt ansprechbar sein. APIs unterscheiden sich somit in den verwendeten Kommunikations- und Transportmechanismen. In Kapitel 5.4 werden die technischen Grundlagen der Kommunikation, die im ECM-Bereich Verwendung finden erläutert.

Die Kenntnis der Art der Kommunikation einer API ist in Produktivumgebungen, wo zusätzlich noch Firewall-Restriktionen zu berücksichtigen sind, ausschlaggebend für die Realisierung einer Verbindung.

5.4 Kommunikationsorientierte Middleware

5.4.1 Definition

Eine einheitlich anerkannte Definition für Middleware ist in der Literatur nicht zu finden. In [Li03] wird Middleware als jeden Typ von Software bezeichnet, der die Kommunikation zwischen zwei oder mehreren Systemen ermöglicht. In [Li03], [Ka02d] und [Wi08k] lässt sich eine Klassifikation der verschiedenen Middleware-Typen finden. In der vorliegenden Arbeit wird nur die kommunikationsorientierte Middleware im Sinne von [Vo05] behandelt. Hierbei liegt der Schwerpunkt in der Abstraktion der Netzwerkprogrammierung. Middleware hat die Aufgabe die Kommunikationsaufgaben transparent zu übernehmen und die Komplexität und Heterogenität der darunterliegenden Plattformen zu verbergen. Middleware ist im Rahmen des ISO/OSI-Referenzmodels [Wi08l] für Rechnerkommunikation nach [Se99] auf der Ebenen 5 bis 7 anzuordnen (Abbildung 20).

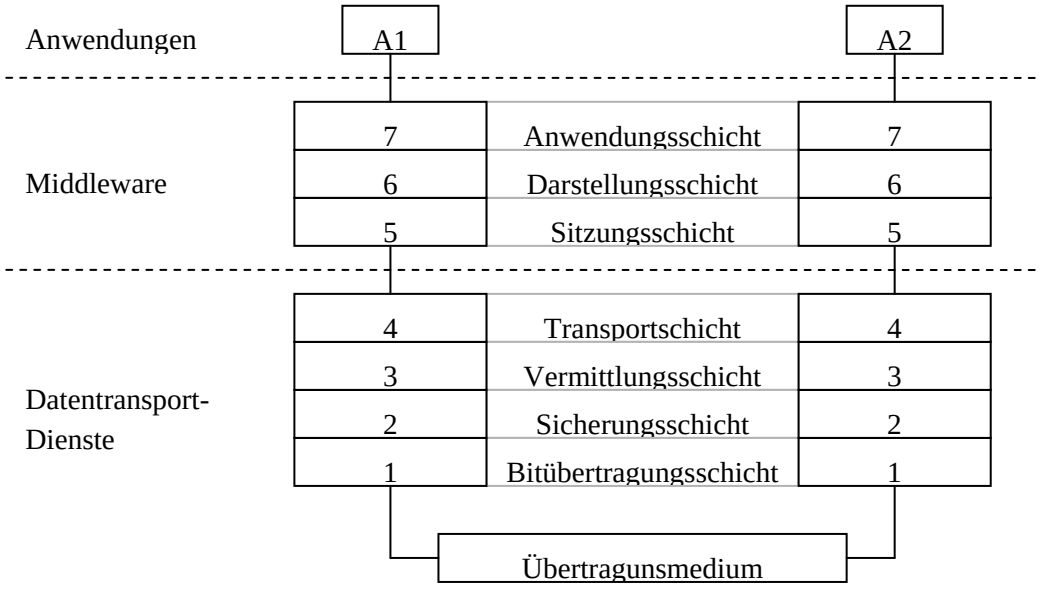


Abbildung 20: Middleware in ISO/OSI-Referenzmodell [Ka02d]

Generell kann zwischen der synchronen und asynchronen Kommunikation unterschieden werden [Ke02b].

Bei einem synchronen Nachrichtenaustausch bleibt der Sender so lange blockiert, bis er eine Antwort bekommen hat. Erst dann kann der Sender die interne Verarbeitung fortsetzten. Der Empfänger sollte somit jederzeit dienstbereit sein.

Bei der asynchronen Kommunikation besteht keine Abhängigkeit zwischen interner Verarbeitung und Nachrichtenaustausch bei Sender und Empfänger. Der Empfänger kann die Nachrichten durch die Nutzung eines Puffers zu einem späteren Zeitpunkt bearbeiten.

Die Begriffe dienen nur dem allgemeinen Verständnis der Kommunikation. Insofern kann ein synchroner Aufruf in einem Thread ausgelagert werden, der blockiert wird, der Haptprozess kann aber weiterhin arbeiten. Umgekehrt kann auch bei einer asynchronen Kommunikation eine Blockierung erreicht werden, wenn der Sender auf eine Antwort des Empfängers wartet, bevor er weiterarbeitet.

In [Ke02b] werden deswegen zusätzlich die Begriffe vorbindungslose und verbindungsorientierte Kommunikation eingeführt. Die verbindungsorientierte Kommunikation erfordert, dass der Sender eine Verbindung zum Empfänger aufbaut und sich an ihm anbindet. Dazu muss er den Empfänger zumindest kennen oder suchen. Sender und Empfänger teilen einen gemeinsamen Zustand bis der Kommunikationsvorgang abgeschlossen ist.

In diesem Sinne sind die im Folgenden präsentierten Kommunikationsmechanismen synchron, blockierend und verbindungsorientiert.

5.4.2 RESTfull API

RESRfull APIs halten sich an den Representational State Transfer (REST) Softwarearchitekturstil, der von Roy Fielding im Rahmen seiner Dissertation [Fi00] formalisiert wurde. Er ist einer der Autoren der Hypertext Transfer Protocol (HTTP) Spezifikation. Somit beruht die Architektur von dem Word Wide Web auf REST-Prinzipien.

5.4.2.1 REST

In [Fi00] werden die Grundprinzipien von REST definiert. Der REST-Architekturstil besteht aus eine Menge von Constrains, die auf die Elemente der Architektur angewendet werden:

- *Client-Server* – Eine Trennung zwischen Client und Server impliziert eine Trennung von Zuständigkeiten. Dies führt zu einer Portabilität der Präsentationsschicht und zur Möglichkeit der Skalierbarkeit der Serverseite.
- *Zustandslosigkeit* – Die nächste Einschränkung ist die Zustandslosigkeit der Kommunikation. Jede Anfrage von dem Client auf den Server muss alle nötigen Informationen enthalten, ohne sich auf den Server gespeicherten Kontext zu verlassen. Der Zustand der Session wird somit vollständig in dem Client verwaltet.

- *Cache* – Ein Cashe wird vorausgesetzt für die Verbesserung der Netzwerkeffizienz. Die übertragenen Daten werden gekennzeichnet, ob sie cachefähig sind und wie lange sie zwischengespeichert werden können.
- *Einheitliche Schnittstelle* – Im vergleich zu anderen netzwerkbasierten Architekturstile besteht die wesentliche Eigenschaft von REST darin, dass zwischen Softwarekomponenten eine einheitliche Schnittstelle verwendet wird.
- *Geschichtetes System* – REST erlaubt Zwischenschichten zwischen Client und Server z.B. in Form von Proxies oder Gateways.
- *Code on Demand* – Die Clientfunktionalität kann zur Laufzeit beispielsweise durch Java-Applets erweitert werden, so dass der Client diese Funktionalität nicht selbst implementieren muss.

Die zentrale Abstraktion in REST ist die Resource, wobei eine beliebige Information wie z.B. ein Dokument, ein Bild oder ein Dienst als Ressource zur Verfügung stehen kann. Eine Resource wird nach außen eindeutig durch ein URI identifiziert und ist über eine einheitliche Schnittstelle zugreifbar.

Die Resourcen werden in Form von Repräsentationen übertragen. Eine Repräsentation stellt einen beabsichtigten Zustand einer Ressource zu einem bestimmten Zeitpunkt dar.

5.4.2.2 Kommunikation

Für die Umsetzung des REST-Architekturstils werden Internet-Technologien verwendet. Als Transportprotokoll wird meistens das HTTP verwendet.

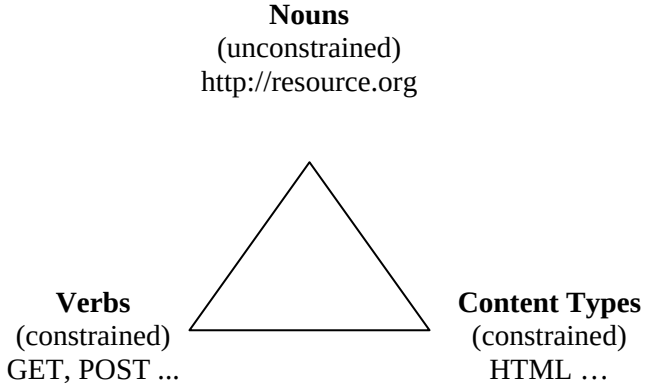


Abbildung 21: REST-Dreieck – Verbs, Nouns und Content Types [Wi08m]

Der Begriff RESTfull API bezeichnet somit APIs, die die REST Prinzipien implementieren. Eine RESTfull API wird durch den REST-Dreieck definiert [Re08]:

- *Nouns* werden gebraucht um die Resourcen durch ein URI eindeutig zu identifizieren.
- *Verbs* sind Operationen, die für eine Resource aufgerufen werden können.

- *Content Types* definieren welche Representationen der Resourcen zur Verfügung stehen.

Abbildung 21 zeigt eine Symbolische Darstellung des REST-Dreiecks. Weiterhin beschreibt die folgende Aufzählung die HTTP-Methoden und deren Bedeutung in REST:

- Mit GET fordert der Client die Repräsentation einer Resource, die durch ein URI identifiziert wird. vom Server an. GET darf keinen Serverzustand ändern und wird deswegen als sicher bezeichnet. GET-Request können beliebig oft geschickt werden.
- Mit POST werden vorhandene Resourcen aktualisiert oder untergeordnete Ressourcen ergänzt. POST ist somit nicht frei von Seiteneffekten.
- Mit PUT werden neue Ressourcen auf dem Server abgelegt oder der Inhalt bestehender Resourcen kann ersetzt werden, falls eine Resource mit der angegebenen URI existiert.
- Mit DELETE löscht der Client eine Resource auf dem Server.

Mit den HTTP-Methoden GET, POST, PUT, DELETE wird für jede Resource eine generische Schnittstelle definiert. In diesem Sinne können die meisten Anwendungsfälle vieler Anwendungen abgedeckt werden, die beispielsweise intern die SQL Befehlen SELECT, INSERT, UPDATE und DELETE verwenden können.

In Tabelle 25 wird ein Beispiel von der Alfresco RESTfull API vorgestellt (Kapitel 6.1.3). Ein Node mit der Referenz *noderef* kann mittels der Methoden GET, PUT, DELETE bearbeitet werden.

```
// Get Item
GET /node/{noderef}
// Update Item
PUT /node/{noderef}
// Delete Item
DELETE /node/{noderef}
```

Tabelle 25: Alfresco RESTfull API

5.4.3 Java Remote Method Invocation (RMI)

RMI (Remote Method Invocation) [Su08c] ist die Java Realisierung des Remote Procedure Calls [Sc08]. RMI ermöglicht eine transparente synchrone Kommunikation von Java-Anwendungen in verschiedenen Adressräumen. Bereits in der ersten benutzbaren Version 1.0.2 des Java Development Kid (JDK) von 1995 stand eine Socket-Bibliothek für Interprozesskommunkitation zur Verfügung. Aufgrund der aufwändigen Programmierung mit Sockets wurde mit der JDK-Version 1.1 die Remote Method Invocation API eingeführt. Mit Hilfe von RMI lassen sich Methoden von Objekten auf entfernten virtuellen Maschinen

(JVM) genauso wie lokale Operationen aufrufen. RMI ermöglicht Orts- und Zugriffstransparenz.

5.4.3.1 Protokoll-Stapel

In [Su08d] wird die RMI-Architektur als eine Schichtenarchitektur beschrieben, die aus drei Schichten (Layer) besteht: der Stub/Skeleton Schicht, der Remote Reference Schicht und der Transportschicht. Abbildung 22 fasst die Schichten auf der Client- und Serverseite zusammen.

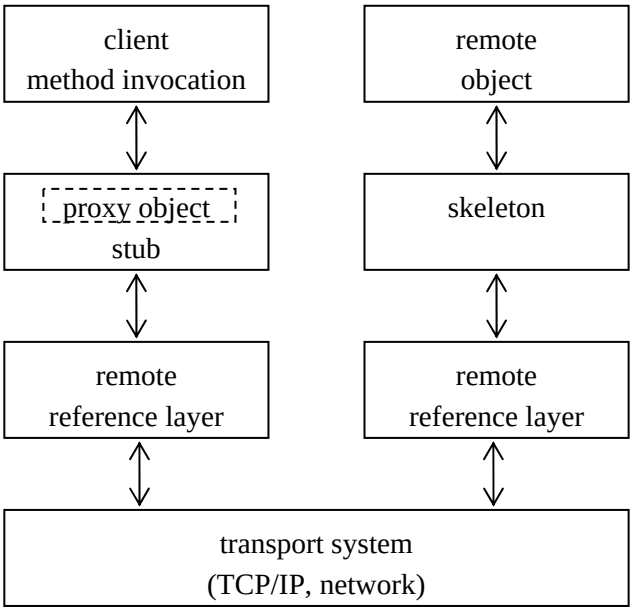


Abbildung 22: RMI Protokoll-Stappel [Sc08]

Die *Stub/Skeleton Schicht* vermittelt die Methodenaufrufe und verbirgt die Kommunikationsdetails. Stubs und Skeletons werden automatisch von dem RMI-Compilers rmic erzeugt. Diese Klassen implementieren die Schnittstelle des Serverobjekts. Sie werden als Proxies für die Realisierung der Netzwerkkommunikation zwischen dem Client und dem Server verwendet. Hier findet auch die Serialisierung und Deserialisierung der Parameter und der Rückgabewerte statt. Seit Java 2 wurde diese Schicht jedoch durch die Reflection API ersetzt und ist in der Praxis aus diesem Grund nicht mehr so häufig anzutreffen.

Die *Remote Reference Schicht* umfasst die Verbindungs-Semantik. Diese Schicht bildet die Referenzen in einer JVM auf Referenzen in einer entfernten JVM ab. Hier werden auch die Übertragungsprotokolle und die Verbindungsdetails verwaltet. RMI implementiert mehrere Protokolle, zwischen denen ausgewählt werden kann. Folgende Übersicht stellt diese Protokolle zusammen und zeigt den jeweiligen Einsatzzweck auf:

- JRMP (Java Remote Method Protocol) ist das Standardprotokoll für RMI
- RMI IIOP zur Integration in CORBA
- RMI über HTTP getunnelt zur Umgehung von Firewalls

- Verschlüsselt über SSL zur sicheren Kommunikation

Die *Transportschicht* ist für den Verbindungsaufbau auf Betriebssystemebene zuständig. Diese Schicht baut auf TCP/IP Sockets auf.

5.4.3.2 Kommunikation

In RMI wird deutlich zwischen einer Client und einer Serverkomponente unterschieden. In Abbildung 23 ist die RMI Kommunikation dargestellt.

Ein Server, der ein Objekt für den entfernten Zugriff zur Verfügung stellen möchte, muss zuerst dieses bei der RMI Registry anmelden. Nach der Registrierung kann ein Client das entfernte Objekt in der RMI Registry aufsuchen. Wenn das Objekt gefunden wurde, wird sein Stub dem Client zurückgegeben. Danach kann der Client auf diesem Stub-Objekt den entfernten Methoden-Aufruf ausführen. Das Stub- und das Skeleton-Objekt verwalten die Kommunikation zwischen dem Klient und dem Server und sind für die Serialisierung und Deserialisierung der Methoden-Parameter und Rückgabewerte zuständig. Auf diese Weise kann das entfernte Objekt als lokales Objekt aufgerufen werden.

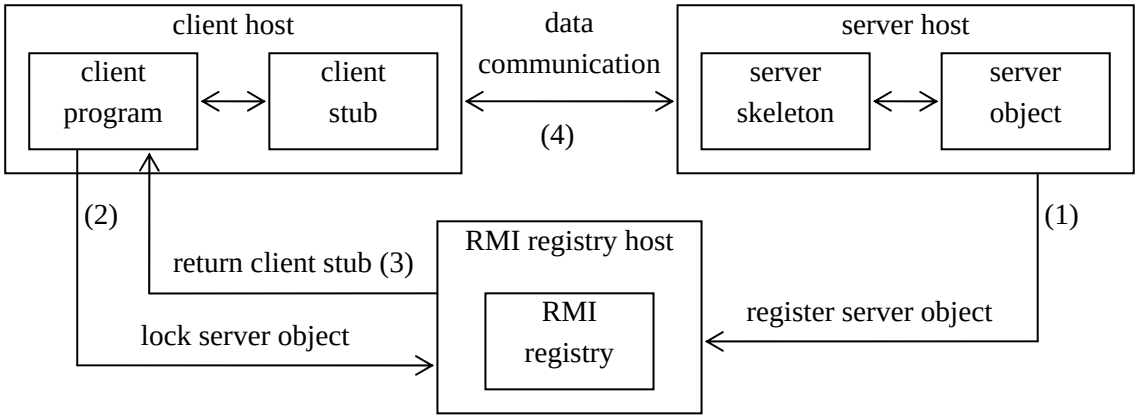


Abbildung 23: RMI Kommunikation [Sc08]

5.4.3.3 Besonderheiten bei Methoden-Parametern

Methoden-Parameter können in RMI auf zwei Arten übergeben werden: mittels „call by reference“ oder „call by value“. Ein entferntes Objekt kann als Remote-Reference transportiert werden. Das Objekt selbst verlässt die JVM nicht, es wird nur eine entfernte Referenz von Typ `java.rmi.Remote` verwaltet. Ein entferntes Objekt kann auch als Kopie übertragen werden. In diesem Fall muss das Objekt die Schnittstelle `java.lang.Serializable` implementieren.

Bei Verwendung von APIs über RMI muss berücksichtigt werden, dass alle Transportobjekte, die als Kopie übertragen werden, die Schnittstelle `java.lang.Serializable` implementieren. Ist dies nicht der Fall, kann die API über RMI nicht benutzt werden. Eine API muss folglich für die entfernte Benutzung entwickelt oder erweitert werden.

In [Su08e] werden die Gründe für die Notwendigkeit der Implementierung der Schnittstelle `java.lang.Serializable` erläutert. Ein Hauptargument dafür ist die Sicherheit. Wird ein Objekt über das Netzwerk serialisiert, so kann es durch Abhören des Streams wieder rekonstruiert werden und alle Variablen einer Klasse, die als private oder protected deklariert sind, können auch gelesen werden, was innerhalb der JVM nicht möglich wäre. Anwendungsentwickler werden durch die Markierung der Transportobjekte als serialisierbar auf diese potenzialen Risiken beim Design von APIs Aufmerksam gemacht. Auf diese Weise lassen sich nur APIs über RMI nutzen, die speziell für den entfernten Aufruf konzipiert sind, verwenden.

5.4.3.4 Umsetzung der Kommunikation

RMI überträgt den Zustand eines Objekts, nicht aber dessen Implementierung. Dies bedeutet, dass die verwendeten Klassen auch beim Client verfügbar sein sollen. Eine API wird als eine Funktionsbibliothek für den Client ausgeliefert, die alle für die Kommunikation benötigten Klassen enthält. Diese Funktionsbibliothek wird meistens als eine jar-Datei komprimiert und muss entweder lokal auf das System verfügbar sein oder dynamisch aus dem Netz von einer so genannten Codebase nachgeladen werden [Su08f].

Damit die Kommunikation zwischen dem Client und dem Server erfolgen kann, muss noch berücksichtigt werden, das beide über die gleiche Version der Funktionsbibliothek verfügen, damit die serialisierung der Objekte stattfinden kann.

5.4.4 Web Services

Das World Wide Web Consortium (W3C) definiert folgendermaßen einen Web Service:

> “A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.” [W308]

Das Ziel von Web Services ist somit die Kommunikation zwischen Anwendungen über ein Netzwerk mithilfe von wohldefinierten Schnittstellen. Web Services beruhen auf folgenden standardisierten Technologien:

- UDDI (Universal Description, Discover, and Integration): ist ein Verzeichnisdienst zur Registrierung und dynamisches Finden von Webservices [Oa04].
- WSDL (Web Service Description Language): ist eine formale Beschreibung des Services. Definition der unterstützten Methoden und deren Parametern [W301].
- SOAP (Simple Object Access Protocol): ist das verwendete Protokoll für die Kommunikation [W307].

- XML (Extensible Markup Language): ist der universale Datenformat [W306].

5.4.4.1 Protokoll-Stapel

Abbildung 24 zeigt den Protokoll Stapel bei Web Services:

- *Communications:* Auf dieser Ebene findet der Nachrichtenaustausch zwischen den Prozessen. Für die Kommunikation können anwendungsorientierte Transportprotokolle eingesetzt werden: HTTP, SMTP [Wi08n], FTP oder auch JMS [Wi08o].
- *Messages:* Zur Übermittlung von Nachrichten zwischen Web Services wird das SOAP-Protokoll eingesetzt. SOAP ist ein leichtgewichtiges XML basiertes Protokoll, das regelt, wie Daten in der Nachricht abzubilden und zu interpretieren sind, und gibt eine Konvention für entfernte Prozeduraufrufe mittels SOAP-Nachrichten vor. Eine SOAP-Nachricht besteht aus einem Envelope, der zwei weitere Komponenten umfasst: einen SOAP Header, der Metadaten enthält, und einen SOAP Body, in dem die eigentliche Nachricht kodiert ist. Weiterhin kann der Envelope einen Verweis auf eine XML-Schema-Datei enthalten, gegen die die Nachricht validiert werden kann.
- *Descriptions:* In dieser Schicht geht es um die Beschreibung von Web Services. WSDL ist eine Sprache zur einheitlichen und unabhängigen Beschreibung der Schnittstelle. Mit dieser genaueren Spezifikation ist es möglich Web Services interoperabel zu machen und die entsprechende Software bei Dienstbringer und Nutzer automatisch zu erzeugen.
- *Processes:* In dieser Schicht sind alle Mechanismen zu finden, die das Zusammenfügen von Web Services zu Prozessen beschreiben. Ein wichtiger Bestandteil dieser ebene ist UDDI, ein Verzeinis-Dienst für Suche nach verfügbaren Web Services.

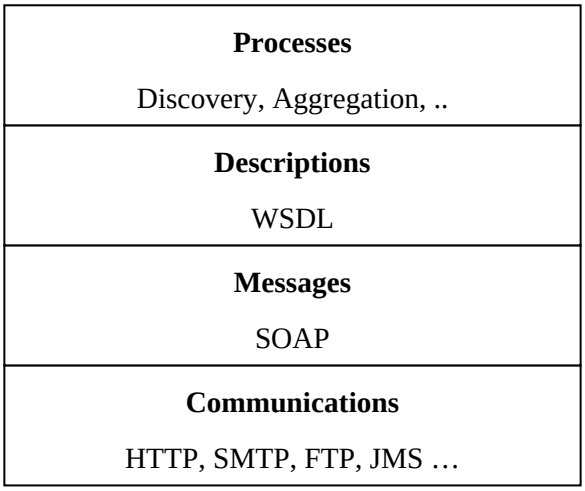


Abbildung 24: Der Protokoll-Stapel von Webservices [W303]

5.4.4.2 Kommunikation

In Abbildung 25 ist die Interaktion bei Web Services dargestellt. Das am häufigste verwendete Protokoll für die Kommunikation ist HTTP. Ein Szenario ist dass der Client eine SOAP-Nachricht mit der HTTP-POST-Methode an dem Server sendet, der wiederum mit einer SOAP-Nachricht mittels HTTP-Response antwortet. Diese Vorgehensweise setzt voraus, dass der Service dem Client bekannt ist. Zusätzlich könnnen UDDI und WSDL benutzt werden. Die WSDL-Datei beschreibt dabei den kompletten Dienst, den ein Web Service zur Verfügung stellt. Damit meldet der Service Provider den bereitgestellten Dienst bei dem Discovery Agent an. Der Client sucht mittels UDDI ein Service beim Discovery Agent und kann anhand der WSDL-Datei die Beschreibung der Schnittstelle erwerben. Danach kann die Semantik der Anfragesoftware erstellt werden. Nach der Entwicklungsphase kann der Client den Dienst nutzen.

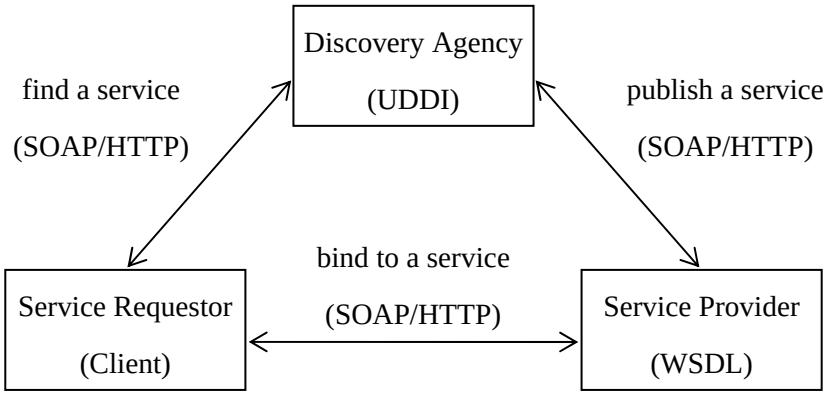


Abbildung 25: Web Services Interaktion [W302]

6 Architekturen führender ECM-Produkte

In diesem Kapitel werden ECM-Produkte vorgestellt, die laut aktueller Analysen von Gartner [Sh07] und Forrester [Mc07] unter den führenden auf dem Markt sind. Bei der Auswahl der Produkte wurde noch berücksichtigt, dass JSR 170 unterstützt wird.

6.1 Alfresco

6.1.1 Unternehmen und Produkt

Alfresco Software Ltd. wurde im Juni 2005 von John Newton, Mitbegründer von Documentum, und John Powell, ehemaliger leitender Geschäftsführer von Business Objects, gegründet. Das Projekt wurde von Mayfield Fund [Ma08] and Accel Partners [Ac08a] finanziell unterstützt. Das Ziel von Alfresco ist eine Open Source Enterprise-Content-Management-Plattform anzubieten, die kommerzielle Produkte in Funktionsumfang, Funktionalität und Nutzen übertrifft [Al08a].

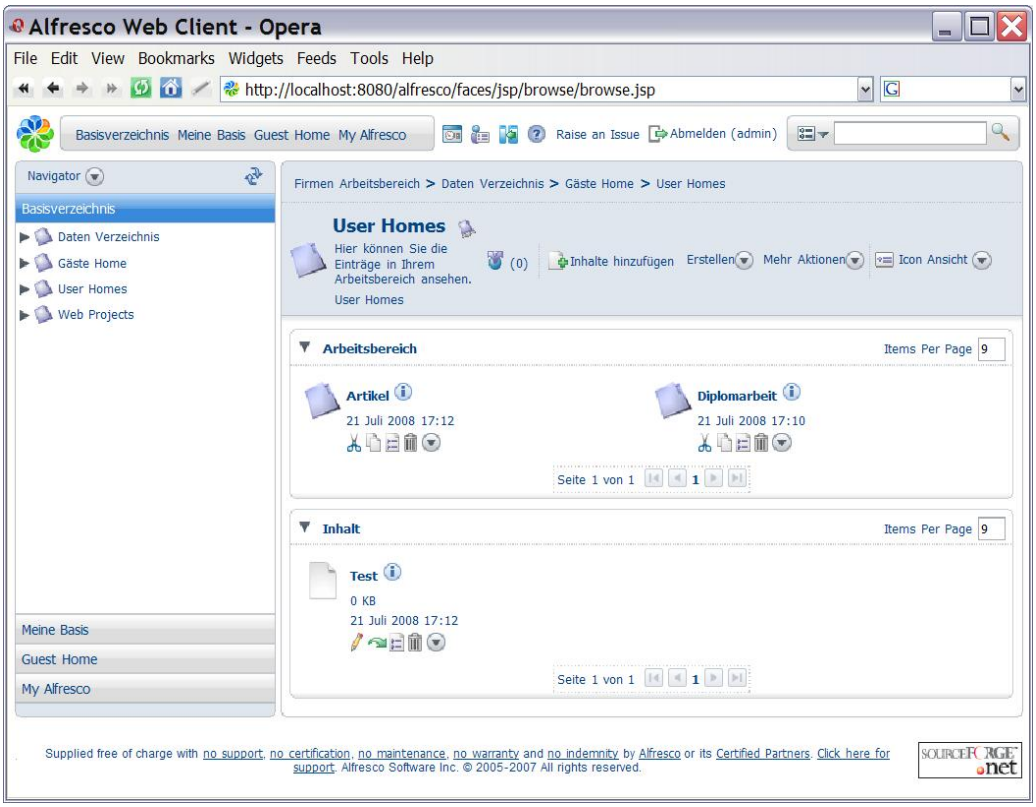


Abbildung 26: Bildschirmfoto der Benutzeroberfläche von Alfresco Community 2.1

Alfresco ist die führende Open-Source-Alternative für Enterprise-Content-Management. Das Produkt nutzt die Stärken von Open-Source-Komponenten zur Schaffung von Content-

Management-Lösungen der Enterprise-Klasse mit dem Anspruch auf Skalierbarkeit und niedrigere Gesamtkosten als vergleichbare proprietäre Lösungen.

Neben der kostenlosen Version *Alfresco Community*, die unter GPL benutzt werden kann, wird auch die kommerzielle Version *Alfresco Enterprise*, die einer kommerziellen kostenpflichtigen Lizenz unterliegt, angeboten. Die beiden Versionen unterscheiden sich nicht in der Funktionalität, sondern in der Form des angebotenen Supports durch Alfresco Software Ltd.

Die Benutzeroberfläche von Alfresco ist vollständig webbasiert und in Abbildung 26 zu sehen. Alfresco bietet ECM-Lösungen in den Bereichen Dokumenten-Management, Web-Content-Management, Records-Management, und Imaging.

6.1.2 Architektur

Alfresco zeichnet sich durch eine besonders gut strukturierte und flexible Architektur aus. Es werden neue Lösungsansätze wie aspektorientierte Programmierung [Wi08p] verwendet. Die Architektur von Alfresco, die auf *Java* [Su08g], *Spring* [Sp08] und *Hibernate* [Hi08] basiert, kann skaliert und verteilt werden. Alfresco kann auf jedem J2SE 5.0 Java-Anwendungsserver wie *Apache Tomcat* [Ap08c] oder *JBoss Application Server* [Jb08b] deployed werden und erlaubt so den Betrieb unter Hochverfügbarkeitsanforderungen. Indexieren und Suchen wurden auf Basis von *Lucene* [Ap08d] implementiert. Das Zugriffskonzept ist mit *ACEGI* [Al08b] umgesetzt und erlaubt granulare Zugriffsregeln. Weitere eingesetzte Open-Source-Technologien sind *MyFaces* [Ap08e] für die browserbasierte Benutzeroberfläche und *JBoss jBPM* [Jb08b] für Workflows.

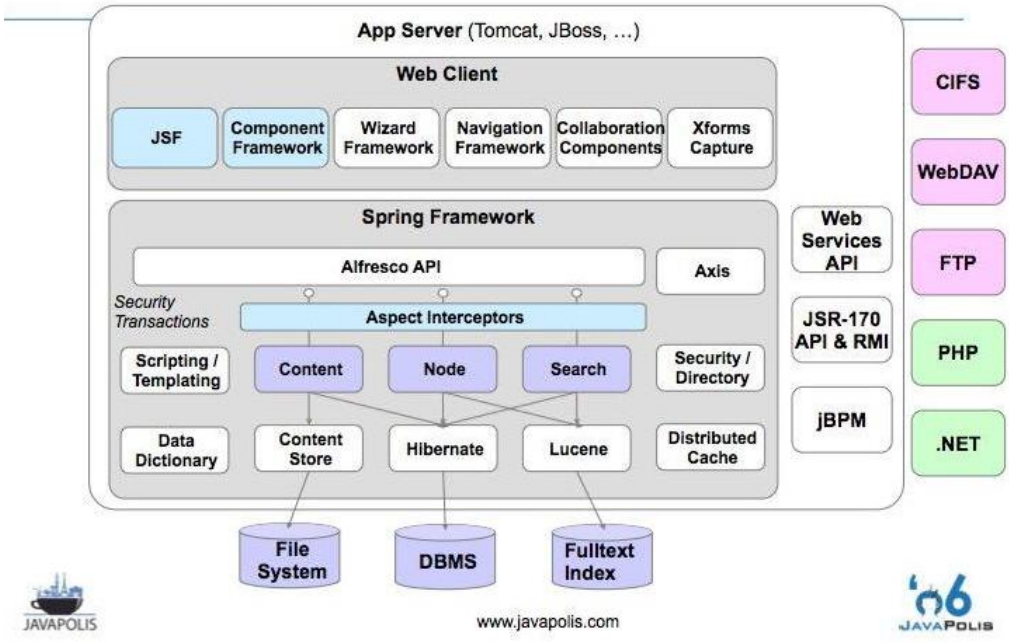


Abbildung 27: Alfresco Architekturdiagramm [Ne06]

Alfresco hat eine Schichten-Architektur (Abbildung 27) und besteht aus dem Alfresco Content-Repository und der Alfresco ECM-Anwendungen, die über Web-Clients benutzt werden können.

Der Einsatz von dem Alfresco-Repository als föderierendes System ist noch nicht implementiert und wird in zukünftigen Versionen ermöglicht.

Für weitere Details über die Alfresco Architektur wird an dieser Stelle auf die Alfresco Wiki [Al08b] verwiesen.

6.1.3 Content-API

Das Alfresco Repository bietet eine Reihe von standardisierten Protokoll-Schnittstellen wie WebDAV, CIFS, FTP an. Die Alfresco Java Foundation API ist die native Bibliothek für Zugriff auf das Repository, die aber nur in Prozess und nicht über RMI benutzt werden kann.

Das Alfresco Repository bietet drei Arten von Remote-Schnittstellen an: Web Services API, RESTfull HTTP API und JSR 170 RMI API. JSR 170 wird auf Level 1 und 2 unterstützt.

Alle Remote-APIs bauen auf die Alfresco Java Foundation API auf. Eine Übersicht ist in Abbildung 28 dargestellt. Die Spezifikation der Schnittstellen kann auf der Alfresco Wiki [Al08c] gefunden werden.

Als Implementierung für die JSR RMI 170 wird die Apache Jackrabbit Bibliothek JCR-RMI [Al08b] verwendet. Zurzeit kann die JCR-RMI API wegen auftretender Laufzeitfehler bis deren Behebung nicht produktiv eingesetzt werden. In [Ap08d] wird die Problematik beschrieben und eine temporäre Lösung ist in [Al08e] zu finden.

Abbildung 28: Alfresco Remote APIs

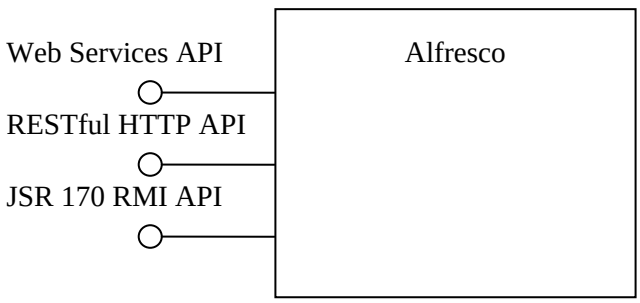


Abbildung 29: Symbolisches Modell Alfresco

Abbildung 29 zeigt das Symbolische Modell von Alfresco. Für die Integration sind die Web Services API, die RESTfull HTTP API und die JSR 170 RMI API von Bedeutung.

6.2 Day Communiqué

6.2.1 Unternehmen und Produkt

Day Software Holding AG wurde 1993 in Basel (Schweiz) von Michael Moppert gegründet und ist seit April 2000 an der SWX Swiss Exchange (SWX: DAYN) notiert [Da08b]. Zu Days Kunden gehören weltweit führende Unternehmen. Day bietet ECM-Lösungen in den Bereichen Web-Content-Management, Digital-Asset-Management, Social-Collaboration und Content-Infrastructure.

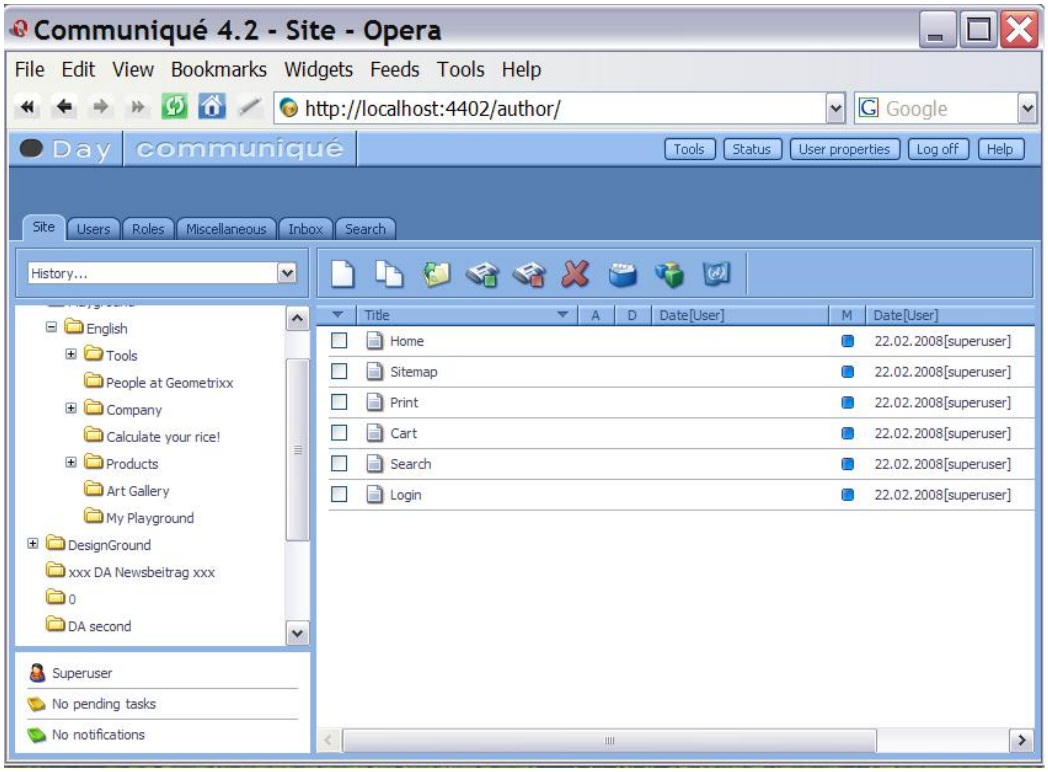


Abbildung 30: Bildschirmfoto der Benutzeroberfläche von Day Communiqué 4.2

Unter der Leitung von Day Software wurde der Standard JSR 170 (Kapitel 2.4.1) entwickelt. Day Communiqué ist die erste kommerziell erhältliche Enterprise-Content-Management-Lösung, bei der die ECM-Anwendung und das Repository voneinander getrennt sind und nativ über die JSR 170 API kommunizieren. Day Communiqué wurde von einem Kernentwicklerteam in enger Zusammenarbeit mit Kunden entworfen, womit das hinzukaufen von Technologien von anderen Anbietern vermieden wurde.

Day Communiqué ist ein Content-Management-System, das die Verwaltung und Veröffentlichung von Content aus unterschiedlichen Systemen ermöglicht. Abbildung 30

zeigt die Webbrowser-basierte Benutzeroberfläche der Autorenumgebung von Day Communiqué 4.2.

6.2.2 Architektur

Day Communiqué zeichnet sich durch eine effiziente, skalierbare und einfach integrierbare Architektur aus. Das System baut auf die Java-Plattform auf. Die ECM-Anwendungen selbst sind auf Basis der eigenen ContentBus™-Technologie entwickelt worden [Da06], die in älteren Versionen von Day Communiqué auch die API für Zugriff auf das Repository zur Verfügung gestellt hat. Ab Version 4.2 wurde der ContentBus™ erweitert, so dass das Repository über die JSR 170 API benutzt werden kann. Abbildung 31 zeigt eine Übersicht der Produktreihe von Day Software, wobei bei allen Produkten die Kommunikation mit dem Repository über die JSR 170 API erfolgt.

Das Content-Repository CRX (Content Repository Extreme) stellt einen separaten Produkt dar und wird als eine Lösung für Content-Infrastruktur vermarktet. Abbildung 32 zeigt die

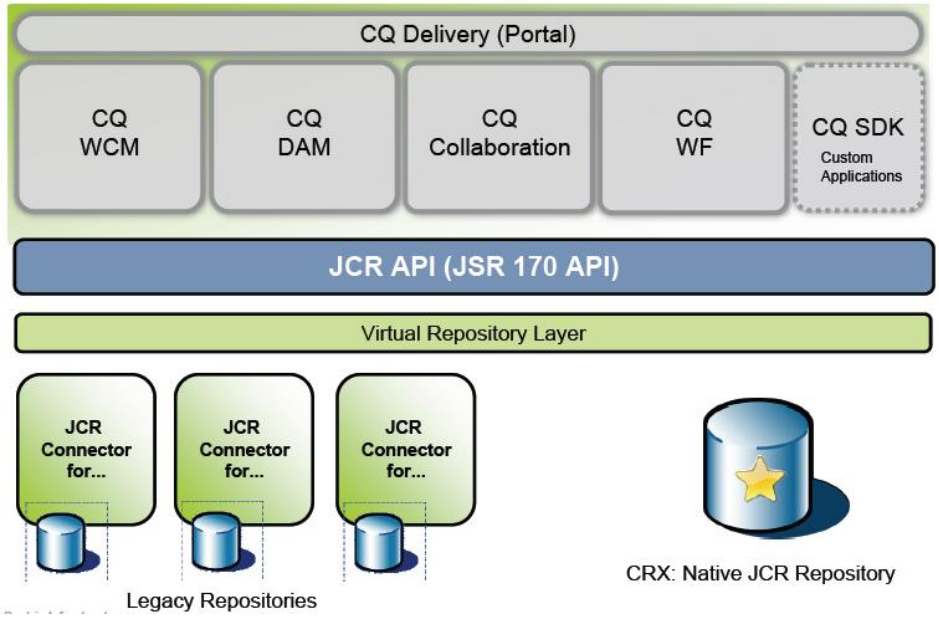


Abbildung 31: Day Communiqué Architekturdiagramm [Da08c]

Architektur von dem CRX-Repository. Ein Virtual-Repository-Layer ermöglicht, dass unterschiedliche Systeme mittels Konnektoren mit CRX verbunden werden. Auf diese Weise lassen sich auch Systeme über die JSR 170 API benutzten, die nativ den Standard nicht unterstützen. Die Systeme werden als JCR-Workspaces dargestellt. Der Zugriff auf die externen Repositories unterscheidet sich somit nicht von dem Zugriff auf einen Standard-JCR-Workspace und erfolgt über die JSR 170 API. In Kapitel 6.3.4 wird der Konnektor für Documentum näher beschrieben.

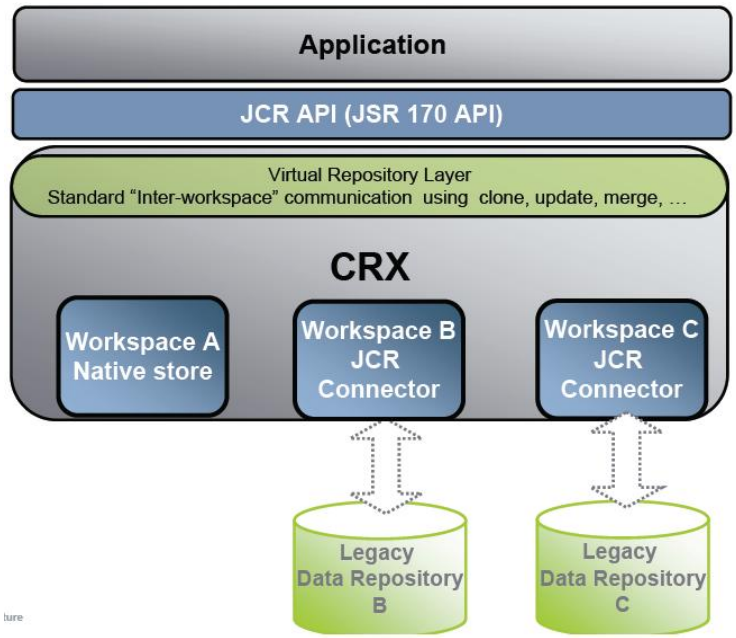


Abbildung 32: CRX Repository Architekturdiagramm [Da08c]

Im Sinne des in Kapitel 4.3.4 eingeführten Konzepts kann CRX als föderiertes System bezeichnet werden, dass einen einheitlichen Zugriff auf unterschiedliche Systeme über eine standardisierte API ermöglicht.

6.2.3 Content-API

Der Zugriff auf CRX kann über die Protokoll-Schnittstelle WebDAV erfolgen. Das CRX Repository bietet ferner als Remote-APIs eine RESTfull HTTP API und die JSR 170 RMI API an. CRX implementiert vollständig den JSR 170 Standard.

Die Remote-APIs bauen auf die native JSR 170 API des Repositorys auf. Eine Übersicht ist in Abbildung 33 dargestellt. Die RESTfull API basiert auf Apache Sling [Ap08f]. Als Implementierung für die JSR RMI 170 API wird die Apache Jackrabbit Bibliothek JCR-RMI [Ap08b] verwendet. Die Spezifikation der Schnittstellen ist entsprechend unter [Ap08g] und [Nü05] zu finden.

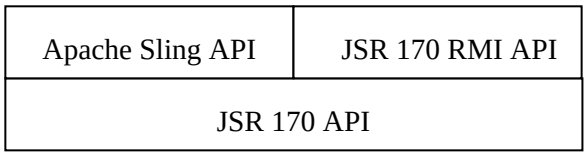


Abbildung 33: Day Communiqué Remote APIs

Abbildung 34 zeigt das symbolische Modell von Day Communiqué. Für die Integration sind die RESTfull HTTP API und die JSR 170 RMI API von Bedeutung.

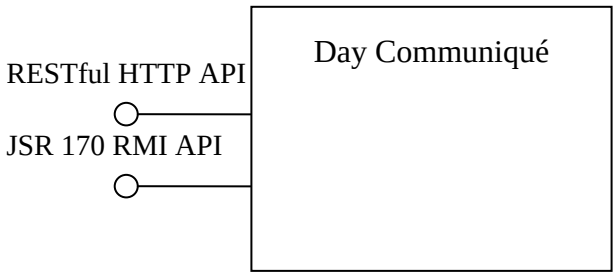


Abbildung 34: Symbolisches Modell Day Communiqué

6.3 EMC Documentum

6.3.1 Unternehmen und Produkt

EMC Corporation mit Hauptsitz in Hopkinton, Massachusetts (USA) ist ein führender Anbieter von Technologien und Lösungen für Informationsinfrastrukturen und an der New York Stock Exchange (NYSE: EMC) notiert. EMC wurde im Jahr 1979 von Richard Egan und Roger Marino gegründet [Em08a].

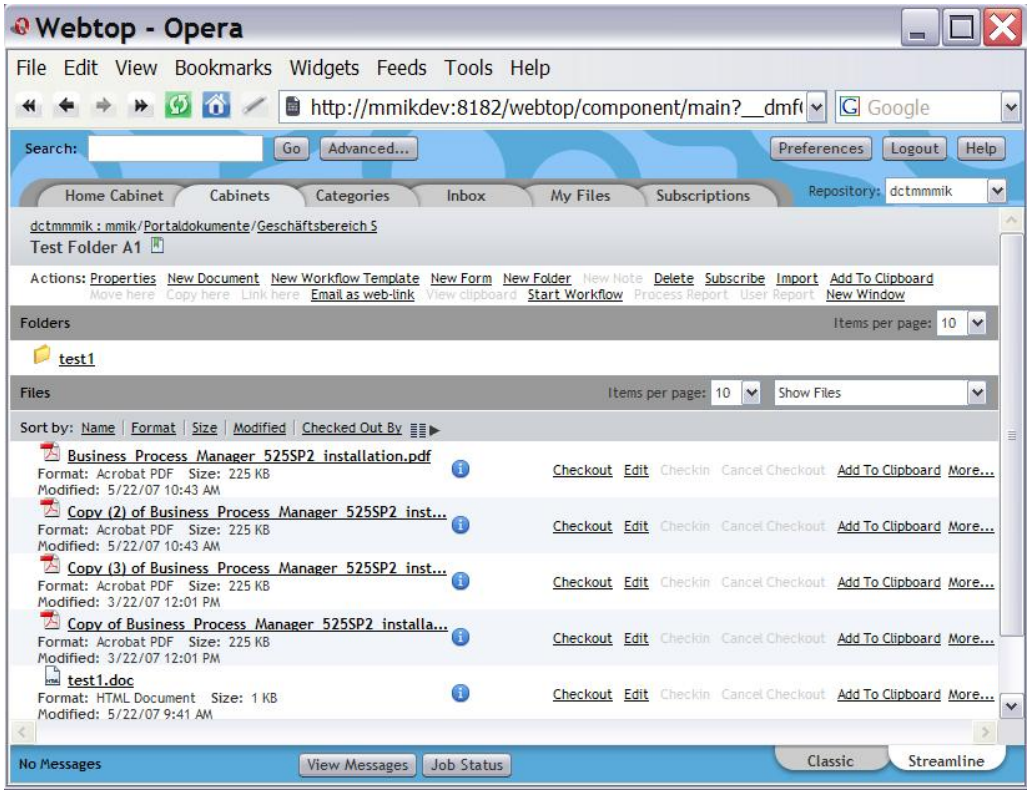


Abbildung 35: Bildschirmfoto der Benutzeroberfläche von Documentum 5.3 Webtop

Documentum wurde von Howard Shao und John Newton im Jahr 1990 als eigenständige Firma gegründet [Wi08r] und im Jahr 2003 von EMC Corporation übernommen und als Produkt erfolgreich weiterentwickelt.

Documentum ist ein Enterprise-Content-Management-System, das eine umfassende Software-Suite in den Bereich Dokumenten-Management, Rekords-Management, Digital-Asset-Management, Web-Content-Management anbietet. Auf [Em08b] ist die vollständige Documentum-Produktreihe vorgestellt.

In Abbildung 35 ist die Webbrowser-basierte Benutzeroberfläche der Webanwendung Documentum Webtop zu sehen.

6.3.2 Architektur

In [Em03a] wird die Documentum-Architektur als eine Vier-Schichten-Architektur beschrieben:

- Die Services-Schicht besteht aus dem Documentum Content Server und erweiterten Content-Services, die als Basis für Content-Management-Lösungen verwendet werden.
- Die Interfaces-Schicht besteht aus der DFC API und der darauf aufbauenden APIs und ermöglicht die Kommunikation zwischen der Services- und der Clients-Schicht.
- Die Clients-Schicht besteht aus Produkten für den Endbenutzer, Entwickler-Tools und Integration mit anderen Systemen
- Die Applications-Schicht besteht aus Produkten von Documentum oder Partnerunternehmen, oder auch Eigenentwicklungen, die die Content-Management-Funktionalität als Teil deren Business-Lösungen nutzen.

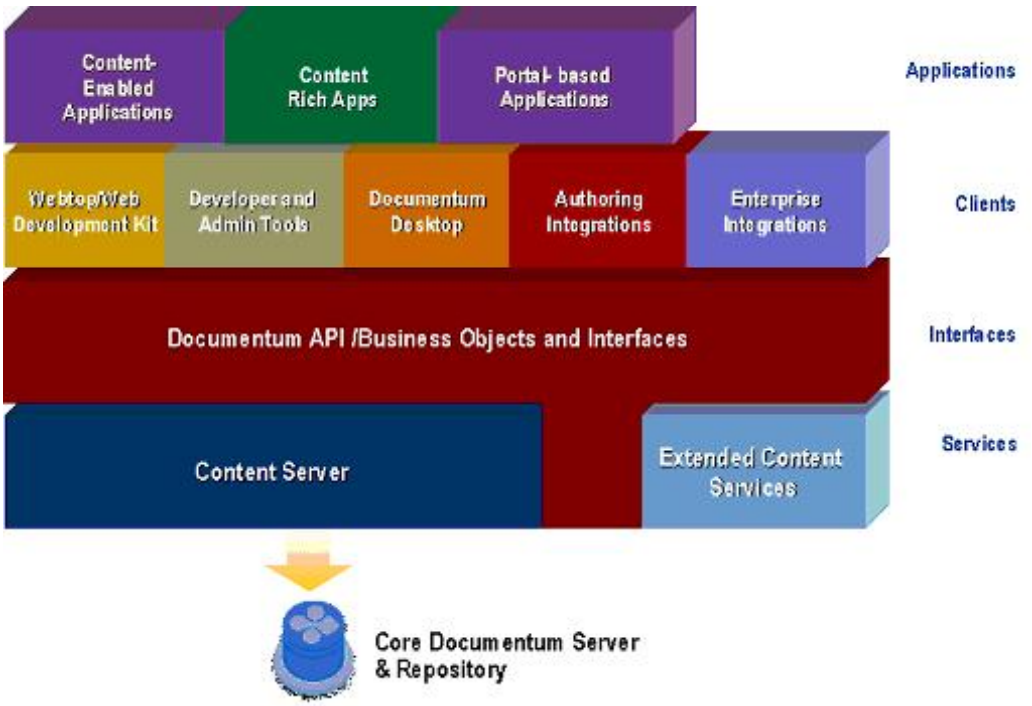


Abbildung 36: Documentum 5.3 Architekturdiagramm [Em03b]

Abbildung 36 zeigt eine Übersicht der Architektur. Die ersten drei Schichten stellen die Documentum-Plattform dar, die die Applications-Schicht unterstützt.

Der Documentum Content Server verwaltet die Repositories, die auch Docbases genannt werden. Im Sinne des in Kapitel 4.3.4 eingeführten Konzepts kann Documentum als föderiertes System eingesetzt werden. Die Föderation von Repositories ist in [Em05d] beschrieben. Eine Föderation besteht aus zwei oder mehreren Repositories, die das Verwalten von globalen Anwender und Gruppen in eine verteilte Konfiguration ermöglichen. In der Föderation wird ein Repository als führendes und alle anderen als Mitglied-Repositories definiert. Es existieren zwei Strategien für das Verwalten von Content. Inhalte von verschiedenen Repositories können mitbenutzt oder auch repliziert werden.

6.3.3 Content-API

Der Zugriff auf Documentum kann über die standardisierten Protokoll-Schnittstellen WebDav und FTP erfolgen. Documentum bietet noch eine Reihe von Remote-Schnittstellen. DMCL (Documentum Client Libary) ist die native API für Kommunikation mit dem Content Server. DMCL ist C++ basiert und wird zusammen mit der daraufbauenden DFC API auf dem Client installiert. Die DMCL API ist verantwortlich für die Client-Server-Kommunikation.

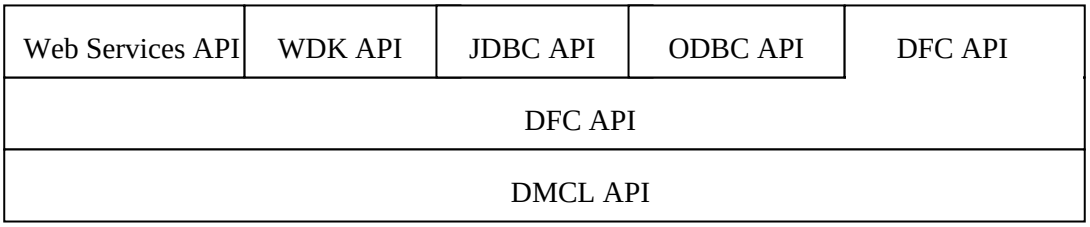


Abbildung 37: Documentum 5.3 Remote APIs

Die DFC API (Documentum Foundation Classes API) ist ein objektorientiertes Java Framework für Zugriff und Erweiterung der Funktionalität des Content Servers. Mittels DFC können auch DQL-Abfragen gegen den Content Server ausgeführt werden. DQL (Documentum Query Language) ist eine Erweiterung von SQL. Weitere von Documentum bereitgestellte APIs sind die standardisierten JDBC API [Su08h] und ODBC API [Mi08]. Die WDK API (Web Development Kit) ist eine API für Erstellung von Java-basierten Web-Anwendungen. Die Webservices API baut auf das Documentum Business Objects Framework auf, mit deren Hilfe wiederverwendbaren Komponenten, die Business-Logik kapseln, implementiert werden können. Abbildung 37 zeigt die Struktur der Documentum APIs. Weitere Informationen über die APIs können von [Em05a], [Em05b], [Em05c] und [Em06] bezogen werden. Alle erwähnten Remote APIs können für Integrationslösungen verwendet werden. Abbildung 39 zeigt das symbolische Modell von Documentum.

6.3.4 JCR-Documentum-Konnektor

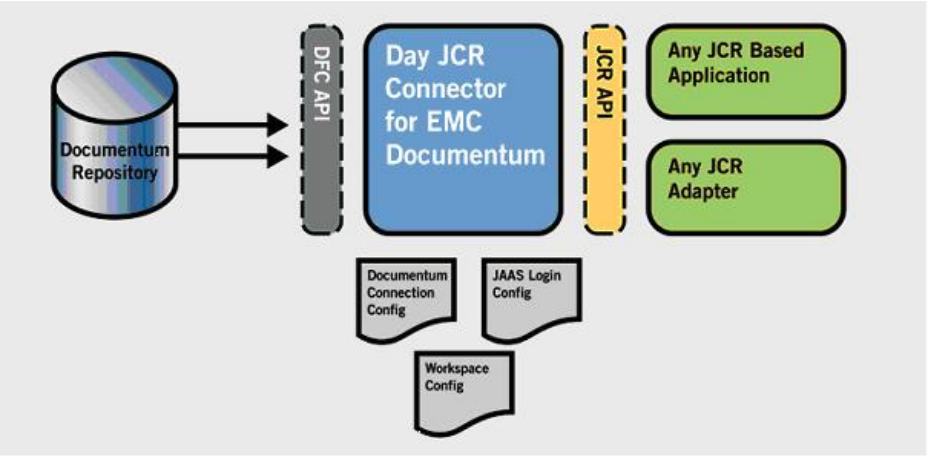


Abbildung 38: JCR Konnektor für Documentum 5.3 [Da08d]

Im Kapitel 6.2.2 wurde die Architektur des Content-Repositorys CRX vorgestellt. Wie jedes Java-Content-Repository besteht CRX aus Workspaces. Ein JCR-Workspace hat ein Interface *PersistenceManager* [Ap08h], das für die persistente Speicherung der Nodes und Properties in Dateisysteme oder Datenbanken verantwortlich ist. Der CRX2Documentum-Konnektor ist eine Implementierung dieses Interfaces von der Firma Day Software AG, mit dessen Hilfe Lesezugriff auf Documentum verschafft wird.

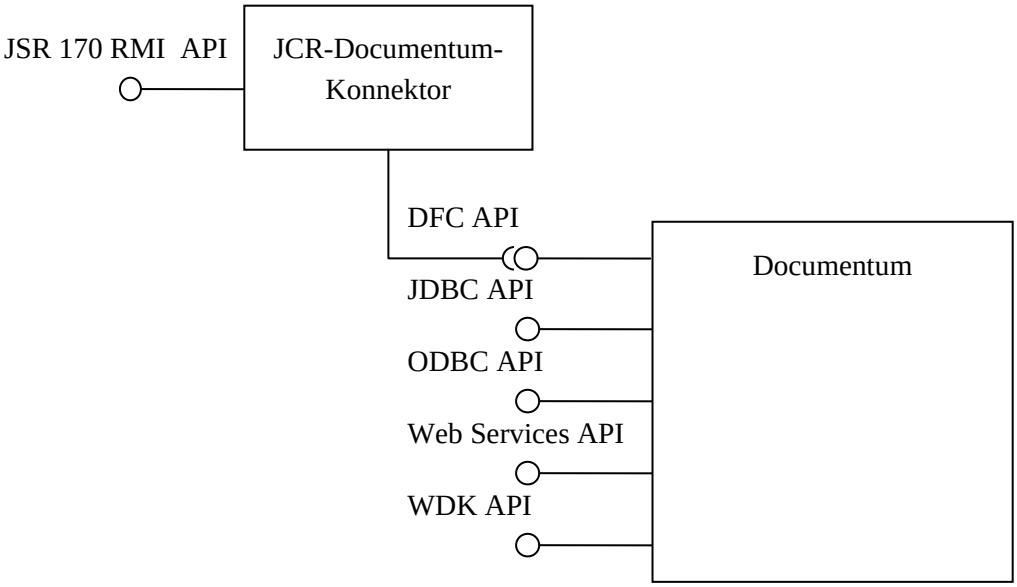


Abbildung 39: Symbolisches Modell Documentum 5.3

Somit wird ermöglicht, die JSR 170 Schnittstelle von CRX für Methodenaufrufe gegen Documentum zu nutzen. Insofern wird nur Level 1 des Standards unterstützt. In [Da08e] ist die Installation und Konfiguration des Konnektors dargestellt. In der vorliegenden Arbeit wird

CRX zusammen mit dem CRX2Documentum Konnektor im Sinne der in Kapitel 5.2.2.2 definierten Integrationskomponente als JCR-Documentum-Konnektor bezeichnet, der die JSR 170 API Methodenaufrufe in DFC API Methodenaufrufe übersetzt.

7 Analyse und Bewertung der technischen Integrationsszenarien

In diesem Kapitel wird die Analyse und Bewertung der in Kapitel 4.4 abgeleiteten Integrationsszenarien mittels JSR 170 beschrieben. Zu diesem Zweck werden typische Vorgehensweisen für die Realisierung der definierten Szenarios vorgestellt und der konkrete Nutzen von JSR 170 für die Bewältigung der Integrationsaufgaben wird bewertet.

Eine explizite Differenzierung nach Produkten wird hier nicht unternommen. Der Grund dafür ist, dass das Ziel dieser Arbeit die szenariobasierte Analyse der Integrationsmöglichkeiten von ECM-Systemen und die Bewertung der Realisierung dieser Szenarien mittels JSR 170 ist. Somit wird eine Bewertung der vorgestellten Produkte zur Realisierung der Szenarien unterlassen. Die Ergebnisse von Kapitel 6 zeigen, inwieweit führende Produkte den JSR 170 Standard unterstützen und welche alternative APIs bereitgestellt werden. Für die Betrachtung der Szenarien aus der JSR 170 Sicht ist weiterhin nur von Bedeutung, ob Level 1, Level 2 oder die optionale Funktionalität des Standards von den zu integrierenden Systemen unterstützt wird.

7.1 Datenmigration

7.1.1 Typische Vorgehensweisen

In Tabelle 16 wurde die Datenmigration definiert. Datenmigration bedeutet programmatisches Kopieren von Inhalten von einem System in ein anderes System unter Nutzung der von den Systemen bereitgestellten datenorientierten Schnittstellen. Dabei ist es möglich, dass auf das Zielsystem Content von mehreren Quellsystemen migriert wird.

Die Datenmigration kann grundsätzlich auf zwei Arten durchgeführt werden:

- Programmatische Migration – eine Migrationskomponente steuert die Migration der Daten durch direkte Verbindungen zu Quell- und Zielsystem.
- Export/Import – alle Daten eines Systems werden mit den zugehörigen Metadaten und Binärdateien in einer Datei exportiert. Die Datei wird danach in das Zielsystem importiert.

Für die Durchführung der programmatischen Migration wird in der Regel eine *Migrationskomponente* (vgl. Kapitel 5.2.2.3) entwickelt. Dabei können RMI-APIs, Webservices APIs und RESTfull APIs benutzt werden. Die angebotenen APIs und deren Funktionalität hängen insofern von dem jeweiligen Produkt ab.

In Kapitel 0 ist ein Quellcode-Beispiel für eine Migrationskomponente vorgestellt, die mit den Schnittstellen, die von den zu integrierenden Systemen bereitgestellt werden, verbunden wird. Sie hat die Aufgabe die Verbindungen aufzubauen, die Migration der Daten

durchzuführen und darauf folgend die Verbindungen zu schließen. Wie schon in Kapitel 5.3.4 gezeigt, wird eine Transformation der Metadaten benötigt, auch wenn die Systeme APIs vom gleichen Typ bereitstellen, da jede Anwendung ein eigenes Datenmodell hat.

Bei dem Export/Import der Daten wird vorausgesetzt, dass sowohl das Quellsystem als auch das Zielsystem das gleiche Format der Migrationsdatei, in der alle Daten des Systems exportiert werden, unterstützen. Ist dies nicht der Fall, soll die Datei nachträglich in das Format des Zielsystems transformiert werden, was z.B. bei Verwendung von XML-Dateien möglich wäre.

7.1.2 Einsatzmöglichkeiten von JSR 170

Bei einer programmatischen Datenmigration werden ein lesender Zugriff zum Quellsystem und ein schreibender Zugriff zum Zielsystem benötigt. Die gegebenenfalls notwendige Transformation der Daten muss immer programmiert werden, weil das Datenmodell, wie in Kapitel 5.3.4 gezeigt, anwendungsspezifisch ist.

JSR 170 kann bei der Datenmigration als RMI-API verwendet werden. Level 2 des Standards muss vom Zielsystem unterstützt werden, weil Schreibeoperationen benötigt werden. Für das Quellsystem ist der Level 1 ausreichend um die Inhalte durch Leseoperationen zu beziehen.

Wenn Produkte JSR 170 nicht nativ unterstützen, können zusätzliche Komponenten - *Konnektoren* (vgl. Kapitel 5.2.2.2), die proprietäre APIs auf JSR 170 abbilden, eingesetzt werden. Ein Beispiel dafür ist der JCR-Dokumentum-Konnektor (vgl. Kapitel 6.3.4).

Die Existenz solcher Standartkomponenten ist produktspezifisch. Außerdem besteht die Möglichkeit, dass ein Konnektor nicht den gewünschten Level des Standards unterstützt. Der im Kapitel 6.3.4 beschriebene Konnektor unterstützt nur Level 1 des Standards und kann somit nur mit einem Quellsystem, aus dem Inhalte gelesen werden, verbunden werden.

Darüber hinaus ist Datenmigration ein einmaliger Prozess. In diesem Fall sollen die Lizenzkosten für die Nutzung einer solchen Komponente mit den Kosten, die durch den Aufwand für die Nutzung der nativen APIs entstehen (vgl. Kapitel 5.3.2.4), verglichen werden. Diese Entscheidung ist für jedes Integrationsprojekt individuell zu treffen.

Wie in Kapitel 5.3.2 gezeigt, hängt der Aufwand bei der Realisierung einer Integrationslösung von den APIs ab, die von den Systemen bereitgestellt werden. JSR170 muss sich dabei mit den Produkt APIs messen, die immer zur Verfügung stehen. Im Gegenteil, nicht jedes System unterstützt nativ den JSR 170 Standard.

Ein weiterer Vorteil von JSR 170 ist die Standardisierung der Methoden, die für das Extrahieren des Datenmodels eines Systems verwendet werden. JSR 170 bietet für diesen Zweck das Interface `NodeTypeManger` an. Wie in Kapitel 5.3.4 gezeigt, ist das Kenntnis der Datenmodelle und die Transformation der Daten von dem Modell des Quellsystems in das Modell des Zielsystems notwendig. Bei der Datenmigration erweist sich diese Eigenschaft der JSR 170 API als sehr vorteilhaft, weil diese wichtige Aufgabe auf eine standardisierte Art und

Weise gelöst werden kann. Bei der Nutzung von proprietären APIs ist das Extrahieren der Datenmodelle, wenn überhaupt möglich, produktspezifisch zu lösen.

Weiterhin unterstützt JSR 170 Export/Import-Operationen. In [Nü05] ist das System View XML Mapping dargestellt. Alle Daten von einem Java-Content-Repository können als eine XML-Datei eindeutig serialisiert werden. Auf diese Weise kann die Migration zwischen Systemen vom gleichen Typ verlustfrei durchgeführt werden. Die XML-Datei ist in dem Content-Modell vom Quellsystem strukturiert. Die Datei kann auch in das Modell eines Zielsystems transformiert werden. Danach kann der Import ins Zielsystem erfolgen.

Im Allgemeinen liegt der Nutzen von JSR 170 bei der Datenmigration in der Standardisierung. Daraus resultiert die Nutzung einer einheitlichen API für Realisierung der Migrationskomponente.

Die hier vorgestellten Vorgehensweisen für die Durchführung der Datenmigration können mit JSR 170 für eine größere Anzahl von Systemen, soweit der Standard unterstützt wird, einheitlich angewendet werden. Bei Verwendung von proprietären APIs wird bei jedem Integrationsprojekt individuell vorgegangen.

7.2 Punkt-zu-Punkt Funktionsintegration

7.2.1 Typische Vorgehensweisen

Die *Punkt-zu-Punkt Funktionsintegration* wurde in Tabelle 17 definiert. Bei dieser Art von Integration wird ein System explizit für die Nutzung der datenorientierten Schnittstellen anderer Systeme entworfen. Dieses System wird hier als führendes System bezeichnet. Somit kann das führende System Inhalte von mehreren anderen Systemen beziehen, ohne dass eine Datenmigration (Kapitel 7.1) auf das führende System benötigt wird. Die Inhalte werden weiterhin in den Quellsystemen verwaltet. Das führende System muss in diesem Fall die Verbindung zu den zu integrierenden Quellsystemen jederzeit aufbauen können, damit der Datentransfer erfolgen kann.

Als Vorgehensweise für die Realisierung der *Punkt-zu-Punkt Funktionsintegration* wird die *direkte Verbindung* (Kapitel 5.2.2.1) angewendet. Es können RMI-APIs, Webservices APIs und RESTfull APIs benutzt werden, wobei die angebotenen APIs und deren Funktionalität von dem jeweiligen Produkt abhängen.

Im Folgenden werden einige typische Beispiele für die Anwendung der *Punkt-zu-Punkt Funktionsintegration* in Projekten dargestellt:

- Ein Web-Content-Management-System kann während der Webseitenerstellungs-prozess Inhalte von anderen Systemen z.B. Dokumenten-Management-Systemen beziehen, wobei die Integration transparent für den Endbenutzer des Systems realisiert wird. Der Endbenutzer lädt über einen Browser eine Webseite. Auf dieser Webseite können Inhalte dargestellt werden und Download-Links für Inhalte angeboten werden.

Die Webseite selbst wird im System mit Hilfe einer JSP erstellt. Im JSP Code werden Informationen aus dem DMS ausgelesen und in die Ausgabe geschrieben.

- Redakteure von Web-Content-Management-System bauen Webseiten aus vorgefertigten Komponenten zusammen. Häufig wird eine Komponente bereitgestellt, mit der verschiedene Inhalte wie Bilder oder Dokumente unterschiedlicher Formate auf die zu erstellende Webseite hinzugefügt werden können. Diese Komponente zeigt ein Verzeichnis zur Auswahl der Inhalte. Die Inhalte selbst können von einem anderen System mittels einer direkten Verbindung bezogen werden.
- Ein Dienst wie z.B. Suche kann von einem System angeboten werden. Das System hat dann die Aufgabe weitere Systeme zu durchsuchen und die Suchergebnisse darzustellen. Dabei verbindet sich das führende System mit jedem anderen System über die entsprechenden bereitgestellten APIs und führt die Suche aus. Das führende System stellt die Suchergebnisse auf eine einheitlich Art und Weise dar, so dass die Integration für den Endbenutzer transparent bleibt.

7.2.2 Einsatzmöglichkeiten von JSR170

Bei der Funktionsintegration spielt die Qualität und Mächtigkeit der eingesetzten APIs eine entscheidende Rolle.

Typischerweise gibt es Mindestanforderungen an die API, um ein konkretes Szenario überhaupt realisieren zu können. Wenn zum Beispiel eine Suche auf einem entfernten System effizient ausgeführt werden soll, dann muss eine Suchfunktion in der implementierten API enthalten sein.

JSR 170 enthält alle typischen Grundfunktionen, die von ECM-Systemen bereitgestellt werden, wie zum Beispiel Lesen und Schreiben von Inhalten, Authentifizierung, Suche, Benachrichtigung bei Update des Contents. JSR 170 kann somit in vielen Fällen für eine Funktionsintegration genutzt werden. Die Möglichkeit JSR 170 über RMI anzusprechen ermöglicht zudem auch die Nutzung dieser Schnittstelle beim Zugriff auf entfernte Systeme. Weiterhin wird eine Metadatentransformation (vgl. Kapitel 5.3.4) bei verschiedenen Datenmodellen der Anwendungen benötigt.

Der Aufwand für die Realisierung einer Funktionsintegration in einem konkreten Anwendungsfall entsteht üblicherweise durch das Programmieren von Logik, welche durch die API nicht abgedeckt ist. Je näher die Funktionen der API an eine gewünschte Integrationslösung heranreichen, desto geringer ist der verbleibende Aufwand für die Implementierung. Der Idealfall für ein Funktionsintegrationsszenario wäre ein einziger Funktionsaufruf der API, der die gewünschten Ergebnisse direkt liefert.

Der Nutzen einer API bezüglich Funktionsintegration allgemein ist daher proportional zur Anzahl der bereitgestellten Funktionen, sowie der Möglichkeiten, die die Methoden anhand ihrer Parametern bieten. Die produktspezifischen APIs aller in dieser Arbeit vorgestellten ECM-Systeme stellen jeweils die größtmögliche Zahl von Funktionen und Möglichkeiten zur

Verfügung, weil es sich um „maßgeschneiderte“, systemeigene APIs handelt. JSR 170 ist dagegen eine standardisierte, systemunabhängige API und ist als solche auf eine gemeinsame Schnittmenge von standardisierten Funktionen beschränkt.

Aufgrund der erweiterten Möglichkeiten sind produktspezifische APIs folglich besser geeignet als JSR 170 um den Aufwand für die Realisierung von Funktions-integrationsszenarien zu minimieren.

Demgegenüber bietet JSR 170 jedoch den Vorteil der Standardisierung: Eine Anwendung, die mit JSR 170 arbeitet, kann leichter auf andere Produkte übertragen werden als solche, die mit einer produktspezifischen API arbeiten. Durch den Einsatz von JSR 170 kann also die Wiederverwendbarkeit der Anwendung bzw. des Quellcodes erhöht werden.

Zusätzlich ergeben sich Vorteile durch die verkürzte Einarbeitungszeit von Entwicklern, die mit JSR 170 bereits vertraut sind.

Der Nutzen von JSR 170 für Funktionsintegration ist somit Abhängig vom Anwendungsfall und kann nicht allgemein beurteilt werden. Wenn die Vorteile durch Standardisierung überwiegen, dann kann JSR 170 empfehlenswert sein. Es kann jedoch auch der Fall eintreten, dass eine produktspezifische API aufgrund der Mächtigkeit ihres Funktionsumfangs klare Vorteile bietet.

7.3 Funktionsintegration mittels eines föderierenden Systems

7.3.1 Typische Vorgehensweisen

Tabelle 18 zeigt die *Funktionsintegration mittels eines föderierenden Systems.* Ein föderierendes System ist ein System, dass den Zugriff auf eine Menge von Systemen transparent ermöglicht. Somit muss ein führendes System, das Inhalte von einem föderierenden System bezieht, im Unterschied zu der *Punkt-zu-Punkt Funktionsintegration* die Quellsysteme nicht kennen. Es wird nur eine Verbindung zu dem föderierenden System aufgebaut, dass für den Login, den Session-Management und anschließend den Datentransfer auf den jeweiligen Quellsystemen verantwortlich ist.

Mittels eines föderierenden Systems kann eine einheitliche Sicht auf eine Menge von Repositories realisiert werden. Der Unterschied zu der Datenmigration liegt darin, dass die Inhalte auf den Quellsystemen verwaltet werden. Ein föderierendes System wird als ein virtuelles Repository bezeichnet [Si05]. In diesem Repository können virtuelle Verzeichnisse angelegt werden, dessen Inhalte nicht in dem virtuellen Repository selbst, sondern aus mehreren anderen Systemen aggregiert werden. Weiterhin werden die Inhalte in diesen virtuellen Verzeichnissen bei Änderungen in den Quellsystemen mittels eines Überwachungs-mechanismus aktualisiert. Somit wird eine einheitliche Suche über alle Quellsysteme bereitgestellt.

Ein föderierendes System muss zuerst mit den Quellsystemen über deren angebotenen APIs integriert werden. Dies kann mittels einer *direkten Verbindung* (Kapitel 5.2.2.1) realisiert

werden. Ein führendes System wird danach mit der angebotenen Schnittstelle des föderierenden Systems durch eine *direkte Verbindung* integriert.

Die in Kapitel 7.2 angegebenen Beispielen, die mit *Punkt-zu-Punkt Funktionsintegration* gelöst worden sind, können auch mit einem Virtuellen Repository mit weniger Aufwand gelöst werden. Die Anzahl der zu realisierenden direkten Verbindungen wird reduziert, womit auch der Aufwand.

Von den in Kapitel 6 vorgestellten Produkten können zurzeit das CRX-Repository (Kapitel 6.2) und Documentum (Kapitel 6.3) als föderierende Systeme eingesetzt werden. Bei CRX wird die Verbindung zu verschiedenen Quellsystemen ermöglicht. Als Standardschnittstelle für Zugriff auf CRX wird die JSR 170 API über RMI bereitgestellt. Weiterhin bietet Dokumentum die Möglichkeit zur Föderation von Dokumentum-Repositories. Als Schnittstelle können die Documentum Content APIs, oder auch der JCR-Documentum-Konnektor, bereitgestellt werden.

7.3.2 Einsatzmöglichkeiten von JSR170

Ein föderierendes System kann mittels einer standardisierten Schnittstelle wie JSR 170 einen einheitlichen Zugriff auf mehrere Systeme ermöglichen. Da jedes System eine eigene API bereitstellt, ist der Nutzen bei der Verwendung von einer standardisierten API in diesem Szenario daher entscheidend. Dadurch lassen sich die in Kapitel 5.3.2 beschriebenen Schwierigkeiten bei der Nutzung verschiedener APIs vermeiden.

Einige Produkte wie Documentum dagegen bieten die Möglichkeit zur Nutzung eines föderierenden Systems über die eigene proprietäre API. Werden die Funktionalitäten der Systeme gebraucht, die durch die Standard API JSR 170 nicht abgedeckt sind, bleibt der Einsatz der produktspezifischen APIs überlegen.

Ein föderierendes System dagegen, dass JSR 170 als eine übergreifende standardisierte Schnittstelle anbietet, erlaubt den Zugriff auf Repositories unterschiedlichen Alters, differenter Struktur und verschiedener Hersteller. Alte DMS Anwendungen, die bisherige isolierte Informationsinseln dargestellt haben, werden auf reine Speichersysteme reduziert, die ihre Daten als nachgeordneten Dienst unterschiedlichen Anwendungen zur Verfügung stellen können [Ka03j]. Dementsprechend kann ein föderierendes System mit weniger Aufwand realisiert werden, wenn die zu integrierenden Systeme JSR 170 als datenorientierte Remote-Schnittestelle bereitstellen.

In dieser Hinsicht ist der content-zentrische Ansatz von JSR 170 ein klarer Vorteil. Content-zentrische Schnittstellen erleichtern die Integration, in dem der Fokus auf dem einheitlichen Zugriff auf Content und nicht auf der spezifischen Funktionalität der Anwendung liegt. Der Funktionsumfang der JSR 170API ist im Vergleich zu proprietären Produkt APIs begrenzt, ermöglicht aber einer Reihe von Anwendungen einen standardisierten Zugriff auf Content [Fi05].

Im Allgemeinen ergeben sich auch hier die Vorteile bei der Nutzund der JSR 170 API aus der Standardisierung. Wie bei der *Punkt-zu-Punkt Funktionsintegration* (Kapitel 7.2) ist der Einsatz von JSR 170 vom Anwendungsfall abhängig.

8 Zusammenfassung

Ziel der vorliegenden Arbeit ist die Analyse und Bewertung der Integrationsmöglichkeiten von ECM-Systemen mittels JSR 170.

Im Kapitel 2 wurden die grundlegenden Begriffe und Konzepte eingeführt, sowie JSR 170 beschrieben.

Anhand von Dokumentationen aus ECM-Projekten, Fachliteratur und Expertengesprächen wurden die Gründe für Integration in Unternehmen im ECM-Bereich in Kapitel 3 zusammengefasst.

Im Kapitel 4 wurde ein theoretisches formales Modell für Integration eingeführt, mit dessen Hilfe alle möglichen Integrationsmöglichkeiten abgeleitet wurden. Nach der Anwendung dieses Modells auf die Gründe für Integration wurden *Datenmigration*, *Punkt-zu-Punkt Funktionsintegration* und *Funktionsintegration mittels eines föderierenden Systems* als Szenarien identifiziert, die die Integration mittels datenorientierten Schnittstellen im ECM-Umfeld vollständig charakterisieren.

In Kapitel 5 wurde auf die technische Realisierung von Integration und die dabei entstehenden Schwierigkeiten näher eingegangen. Verschiedene Programmiersprachen, Verschiedene APIs, verschiedene Versionen, Verschiedene Datenmodelle und der Transport erwiesen sich als die wesentlichen Herausforderungen einer Integrationslösung. Die *direkte Verbindung* (Kapitel 5.2.2.1), der *Konnektor* (Kapitel 5.2.2.2) und die *Migrationskomponente* (Kapitel 5.2.2.3) wurden als Vorgehensweisen für die Realisierung von Integration identifiziert. Es wurde näher auf Kommunikationsorientierte Middleware eingegangen und die Technologien RESTfull API, RMI, Web Services wurden vorgestellt. Mit Hilfe dieser Kommunikationsmechanismen ermöglichen die im Kapitel 6 vorgestellten ECM-Systemen den Zugriff auf die verwalteten Inhalte.

Anschließend wurde in Kapitel 7 die Realisierung von den identifizierten Integrationsszenarien diskutiert und der Nutzen von JSR 170 bewertet.

Bei allen betrachteten Szenarien wurde festgestellt, dass der wesentliche Vorteil der JSR170 Spezifikation in der Standardisierung liegt. Die Nutzung einer einheitlichen API erleichtert die Realisierung von Integrationslösungen. Aufgrund der internationalen Etablierung der Schnittstelle wird die Zugreifbarkeit der Inhalte langfristig sichergestellt. Zudem ist die Einarbeitungszeit von Entwicklern kürzer, wenn diese mit der API vertraut sind.

Demgegenüber ist der Funktionsumfang der JSR170 API nicht so mächtig wie der der maßgeschneiderten, produkteigenen APIs der proprietären ECM-Systeme. JSR 170 ist als standardisierte, systemunabhängige API auf eine gemeinsame Schnittmenge von standardisierten Funktionen zur Verwaltung von Content beschränkt. Nachteilig ist außerdem dass der JSR170 Standard derzeit nur von wenigen Produkten implementiert wird.

Als Ergebnis der Analyse hat sich herausgestellt, das der Nutzen von JSR 170 vom Anwendungsfall abhängig ist und nicht allgemein beurteilt werden kann. Wenn die Vorteile durch Standardisierung überwiegen, dann kann JSR 170 empfehlenswert sein. Es kann jedoch auch der Fall eintreten, dass eine produktspezifische API klare Vorteile bietet.

Darüber hinaus ist eine wesentliche Erkentniss der Arbeit, dass die anwendungsspezifischen Datenmodelle der Systeme eine beachtliche Hürde bei der Integration darstellen und eine Transformation der Daten erzwingen. In dieser Hinsicht ist die Standardisierung der Methoden, die für das Extrahieren des Datenmodels eines Systems verwendet werden, ein großer Vorteil.

Standardisierung ist ein allgemein anerkannter Trend. In diesem Sinne wird JSR 170 weiterentwickelt und die nächste Version der Spezifikation wird als JSR 283 [Ja07] veröffentlicht. Die wesentlichen Erweiterungen sind die neuen Methoden zur Registrierung von Node-Typen und zur Verwaltung von Workspaces.

Falls der Standard eine breite Verwendung in angebotenen Produkten findet, wird seine Bedeutung in den nächsten Jahren entsprechend zunehmen.

Literaturverzeichnis

Ac08a Accel Partners. http://www.accel.com/, 2008

Ac08b Acegi Security. http://www.acegisecurity.org/, 2008

Ai08 AIIM-The ECM Association: *What is ECM.* http://www.aiim.org/ResourceCenter/AboutECM.aspx, 2008

Al08a Alfresco Software Ltd.: *Über Alfresco.* http://www.alfresco.com/de/about/, 2008

Al08b Alfresco Software Ltd.: *Alfresco_Repository_Architecture.* http://wiki.alfresco.com/wiki/Alfresco_Repository_Architecture, 2008

Al08c Alfresco Software Ltd.: *Developer Guide.* http://wiki.alfresco.com/wiki/Developer_Guide, 2008

Al08d Alfresco Software Ltd.: *JCR-RMI Extension.* http://wiki.alfresco.com/wiki/JCR-RMI_Extension, 2008

Al08e Alfresco Software Ltd.: *Extension JCR Thread Bound.* http://forge.alfresco.com/frs/shownotes.php?release_id=298, 2008

Ap08a Apache Jackrabbit: *Apache Jackrabbit.* http://jackrabbit.apache.org/, 2008

Ap08b Apache Jackrabbit: *Jackrabbit JCR-RMI.* http://jackrabbit.apache.org/jackrabbit-jcr-rmi.html, 2008

Ap08c Appache Tomcat. http://tomcat.apache.org/, 2008

Ap08d Apache Lucene. http://lucene.apache.org/, 2008

Ap08e Apache MyFaces. http://myfaces.apache.org/, 2008

Ap08f Apache Sling: *Apache Sling.* http://incubator.apache.org/sling/site/index.html, 2008

Ap08g Apache Sling: *Documentation.* http://incubator.apache.org/sling/site/documentation.html, 2008

Ap08h Apache Jackrabbit: *PersistenceManagerFAQ.* http://wiki.apache.org/jackrabbit/PersistenceManagerFAQ, 2008

Bu03 Bussler, C.: *B2B Integration. Concepts and Architecture.* Berlin: Springer Verlag, 2003

Bü01 Büchner H.; Zschau O.; Traub D.; Zahradka R.: *Web Content Management.* Bonn: Galileo Pess, 2001, S. 45

Co06 Conrad S.; Hasselbring W.; Koschel A.; Trisch R. : *Enterprise Application Integration.* 1. Auflage. München: Spektrum Akademischer Verlag, 2006, S.12

Da06 Day Software Holding AG: *Day Technical Training,* 2006

Da08a Day Software Holding AG: *Standardizing the Content Repository*, 2008, S. 31

Da08b Day Software Holding AG: *Unternehmen.* http://day.com/site/de/index/company/company_overview.html.html

Da08c Day Software Holding AG: *CQ WCM and Connectors*, 2008

Da08d Day Software Holding AG: *Day JCR Connector for EMC Documentum V5.x*, 2008

Da08e Day Software Holding AG: *CRX Connector for EMC Documentum v5.3 and v5.2.5*, 2008

De99 *Deutsches Handelsgesetzbuch (HGB).* München C.H. Beck Verlag, 1999, § 257

Du00 *Duden. Das große Fremdwörterbuch.* 2. Auflage., Mannheim: Dudenredaktion, 2000, S. 628

Du03 Dunkel J.; Holitschke A. : Softwarearchitektur für die Praxis, Springer Verlag, 2003

Em02 EMC Documentum Inc.: *Developing Documentum™ Applications,* 2002, S. 4

Em03a EMC Documentum Inc.: *Documentum 5 Architecture: A Technical Overview,* 2005, S. 25

Em03b EMC Documentum Inc.: *Documentum 5 Architecture: A Technical Overview,* 2005, S. 26

Em05a EMC Documentum Inc.: *Content Server Fundamentals*, 2005

Em05b EMC Documentum Inc.: *Content Server API Reference Manual*, 2005

Em05c EMC Documentum Inc.: *Content Server DQL Reference Manual*, 2005

Em05d EMC Documentum Inc.: *Distributed Configuration Guide*, 2005

Em06 EMC Documentum Inc.: *Web Development Kit and Client Applications Development Guide*, 2006

Em08a EMC Corporation: *Dokumetum-Produktreihe.* http://germany.emc.com/products/family/documentum-family.htm, 2008

Em08b EMC Corporation: *EMC im Überblick.* http://germany.emc.com/about/emc-at-glance/corporate-profile/index.htm, 2008

Fi00 Fielding R. : *Architectural Styles and the Design of Network-based Software Architectures.* University of California, Dissertation, 2000

Fi05 Fielding R. : *JSR 170 Overview*, 2005

Gh06 Ghezzo M.: *Standardsoftware vs. Individualsoftware.* http://www.computerwelt.at/detailArticle.asp?a=105826&n=2, 2006

Hi08 Hibarnate Framework http://www.hibernate.org/, 2008

In07 International Data Corp: The expanding Digital Universe - *A Forecast of Worldwide Information Growth Through 2010*, 2007

Ja05 Java Community Process: *JSR 170: Content Repository for JavaTM technology API.* http://jcp.org/en/jsr/detail?id=170, 2005

Ja07 Java Community Process: JSR 170: Content Repository for JavaTM technology API Version 2.0. http://jcp.org/en/jsr/detail?id=283

Jb08a JBoss Enterprise Application Platform. http://www.jboss.com/products/jbossas, 2008

Jb08b JBoss jBPM, http://www.jboss.com/products/jbpm, 2008

Ka02a Kaib M.: *Enterprise Application Integration.* Dissertation Universität Marburg. Wiesbaden: Deutscher Universitäts-Verlag Gmbh, 2002, S. 10

Ka02b Kaib M.: *Enterprise Application Integration.* Dissertation Universität Marburg. Wiesbaden: Deutscher Universitäts-Verlag Gmbh, 2002, S. 19

Ka02c Kaib M.: *Enterprise Application Integration.* Dissertation Universität Marburg. Wiesbaden: Deutscher Universitäts-Verlag Gmbh, 2002, S. 60-67

Ka02d Kaib M.: *Enterprise Application Integration.* Dissertation Universität Marburg. Wiesbaden: Deutscher Universitäts-Verlag Gmbh, 2002, S.102

Ka03a Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise?.* Project Consult GmbH., 2003, S. 63

Ka03b Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise*?. Project Consult GmbH., 2003, S. 89

Ka03c Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise?.* Project Consult GmbH., 2003, S. 371

Ka03d Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise?.* Project Consult GmbH., 2003, S. 90

Ka03e Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise?.* Project Consult GmbH., 2003, S. 92

Ka03f Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise?.* Project Consult GmbH., 2003, S. 55

Ka03g Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise?.* Project Consult GmbH., 2003, S. 286

Ka03h Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise?.* Project Consult GmbH., 2003, S. 289

Ka03i Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise?.* Project Consult GmbH., 2003, S. 235

Ka03j Kampffmeyer, U.: *Dokumenten-Technologien: Wohin geht die Reise?.* Project Consult GmbH., 2003, S. 290

Ka07 Kampffmeyer, U.: *ECM - Status Quo und Vision,* 2007

Ka99a Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 16

Ka99b Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 56

Ka99c Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 94

Ka99d Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 228

Ka99e Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 134

Ka99f Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 101

Ka99g Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 20

Ka99h Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 140

Ka99i Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 189-190

Ka99j Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 100

Ka99k Kampffmeyer, U.: *Dokumenten-Management. Grundlagen und Zukunft.* Project Consult GmbH.,1999, S. 113-114

Ke02a Keller W. *Enterprise Applikation Integration*. Heidelberg: dpunkt-Verlag, 2002, S. 63

Ke02b Keller W. *Enterprise Applikation Integration.* Heidelberg: dpunkt-Verlag, 2002, S. 76-68

Ku96 Kurbel K.; Rautenstrauch C.: *Integration Engineering: Konkurenz oder Komplement zum Information Engineering.* München: Oldenbourg-Verlag , 1996, S. 169

Li03 Linthicum D.: *Next Generation Application Integration.* Addison Wesley, 2003

Ma05 Masak D.: *Moderne Enterprise Architekturen.* Berlin: Springer Verlag, 2005, S.40

Ma08 Mayfield Fund. http://www.mayfield.com/, 2008

Mc07 McNabb N.: *The Forrester Wave™: Enterprise Content Management Suites, Q4 2007.*, 2007

Mi08 Microsoft Corporation: *ODBS.* http://msdn.microsoft.com/en-us/library/ms714562(VS.85).aspx, 2008

MM02 Mitchell, R.; McKim J.: *Design by Contact, by Example.* Addison-Wesley, 2002

Mo07 Moppert M.: *Auf die Architektur kommt es an.* http://www.ap-verlag.de/Online-Artikel/20070910/20070910t%20Day%20Content%20Management%20JSR%20170%20Moppert.htm, 2007

Ne05 Network Working Group: *Uniform Resource Identifier (URI): Generic Syntax.* http://tools.ietf.org/html/rfc3986

Ne06 Newton, J.: *Developing Enterprise Content Applications using Open-Source. Online-Präsentation* http://www.parleys.com/display/PARLEYS/Home#title=DevelopingEnterpriseContentApplicationsusingOpen-Source;slide=1;talk=8166,2006

Nü05 Nüscheler, D.: *Java Community Process: JSR 170: Content Repository API for Java Technology Specification.*, 2005 S. 83

Oa04 OASIS Open: *UDDI Version 3.0.2*. http://uddi.org/pubs/uddi_v3.htm, 2004

Pe04 Peltzer D.: *.NET & J2EE Interoperability*. McGraw-Hill, 2004

Re08 RESTWiki: *Rest Triangle.* http://rest.blueoxen.net/cgi-bin/wiki.pl?RestTriangle, 2008

Ru05 Rupp C.; Hahn J.; Queins S. Jeckle M.; Zengler B.: *UML2 glasklar*. München: Carl Hanser Verlag, 2005, S. 119-120

Sc08 Schlichter J.: *Distributed Applications - Verteilte Anwendungen.* Technische Universität München, Vorelusngsunterlagen, 2008

Se99 Serain D.; Craig L.: *Middleware.* Springer-Verlag, 1999

Sh06 Shariff M.: *Alfresco Enterprise Content Management Implementation.* Packt Publishing, 2006

Sh07 Shegda K.; Bell T.;Chin K.; Gilbert M.: *Magic Quadrant for Enterprise Content Management*, 2007

Si05 Silver, B.: Content: *The Other Half of the Integration Problem*, http://www.intelligententerprise.com/showArticle.jhtml?articleID=171000641, 2005

Si06 Siedersleben, J.: *Moderne Softwarearchitekturen.* 1. Auflage. dpunkt.verlag, 2006, S. 44

Sp08 Spring Framework http://www.springframework.org/, 2008

Su08a Sun Microsystems Inc.: *Java Authentication and Authorization Service.* http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html, 2008

Su08b Sun Microsystems. *Java Transaction API (JTA) Specification.* http://java.sun.com/products/jta/index.html, 2008

Su08c Sun Microsystems Inc.: *Java Remote Method Invocation*. http://java.sun.com/javase/6/docs/platform/rmi/spec/rmiTOC.html, 2008

Su08d Sun Microsystems Inc.: *jGuru: Remote Method Invocation (RMI).* http://java.sun.com/developer/onlineTraining/rmi/RMI.html#RMIArchitectureLayers, 2008

Su08e Sun Microsystems Inc.: http://java.sun.com/javase/technologies/core/basic/serializationFAQ.jsp#whyserial, 2008

Su08f Sun Microsystems Inc.: *Dynamic code downloading using JavaTM RMI.* http://java.sun.com/javase/6/docs/technotes/guides/rmi/codebase.html, 2008

Su08g Sun Microsystems Inc.: *Java* http://java.sun.com/javase/, 2008

Su08h Sun Microsystems Inc.: *JDBC.* http://java.sun.com/products/jdbc/overview.html, 2008

SWD03 Schubert P.; Wölfle R.; Dettling W.: *E-Business-Integration: Fallstudien zur Optimierung elektronischer Geschäftsprozesse.*Carl Hanser Verlag, 2003

THB05 Maier, R.; Hädrich T.; Peinl R.: *Enterprise Knowledge Infrastructures.*Springer-Verlag, 2005, S. 204

Vo05 Vogel O.; Arnold I.; Chughtai A.; Ihler E.; Mehling U.; Neumann T.; Völter M.; Zdun U.: *Softwarearchitektur.* München: Spektrum Akademischer Verlag, 2005

W301 *Web Services Description Language (WSDL) 1.1.* http://www.w3.org/TR/wsdl, 2001

W302 W3C: *Web Service Architectural Roles.* http://www.w3.org/2002/ws/arch/2/10/roles_clean.htm, 2002

W303 W3C: *Web Services Architecture.* http://www.w3.org/TR/2003/WD-ws-arch-20030514/, 2003

W306 W3C: *Extensible Markup Language (XML) 1.0.* http://www.w3.org/TR/2006/REC-xml-20060816/, 2006

W307 W3C: *SOAP.* http://www.w3.org/TR/soap/, 2007

W308 W3C: *Web Services Glossary.* http://www.w3.org/TR/ws-gloss/, 2008

W399 W3C: *XML Path Language (XPath)*, 1999

Wi08a Wikipedia: *Enterprise Content Management System.* http://de.wikipedia.org/wiki/Enterprise_Content_Management_System#cite_note-2, 2008

Wi08b Wikipedia: *Integration (Software).* http://de.wikipedia.org/wiki/Integration_(Software), 2008

Wi08c Wikipedia: *Application-To-Application.* http://de.wikipedia.org/wiki/Application-To-Application, 2008

Wi08d Wikipedia: *Universally Unique Identifier.* http://de.wikipedia.org/wiki/Universally_Unique_Identifier, 2008

Wi08e Wikipedia: *Open Source.* http://de.wikipedia.org/wiki/Open_Source, 2008

Wi08f Wikipeida: *Programmirschnitstelle.* http://de.wikipedia.org/wiki/Programmierschnittstelle, 2008

Wi08g Wikipedia: *Protokoll (Informatik).* http://de.wikipedia.org/wiki/Protokoll_(Informatik), 2008

Wi08h Wikipedia: *WebDAV.* http://en.wikipedia.org/wiki/WebDAV, 2008

Wi08i Wikipedia: *FTP.* http://en.wikipedia.org/wiki/Ftp, 2008

Wi08j Wikipedia: *CIFS.* *http://en.wikipedia.org/wiki/Cifs*, 2008

Wi08k Wikipedia: *Middleware.* http://en.wikipedia.org/wiki/Middleware, 2008

Wi08l Wikipedia: *OSI Model.* http://en.wikipedia.org/wiki/OSI_model, 2008

Wi08m Wikipedia: *Representational State Transfer*. http://en.wikipedia.org/wiki/Representational_State_Transfer, 2008

Wi08n Wikipedia: *SMTP*. *http://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol, 2008*

Wi08o Wikipedia: *JMS*. http://en.wikipedia.org/wiki/Java_Message_Service, 2008

Wi08p Wikipedia: *Aspektorientierte Programmierung*. http://de.wikipedia.org/wiki/Aspektorientierte_Programmierung, 2008

Wi08r Wikipedia: *Documentum*. http://en.wikipedia.org/wiki/Documentum, 2008

ZGB05a Zöller, B.: Gulbins, J.; Baumeister, H.: *Dokumenten-Management*. VOI, 2005, S. 113

ZGB05b Zöller, B.: Gulbins, J.; Baumeister, H.: *Dokumenten-Management*. VOI, 2005, S. 122

ZGB05c Zöller, B.: Gulbins, J.; Baumeister, H.: *Dokumenten-Management*. VOI, 2005, S. 441

ZGB05d Zöller, B.: Gulbins, J.; Baumeister, H.: *Dokumenten-Management*. VOI, 2005, S. 461

ZGB05e Zöller, B.: Gulbins, J.; Baumeister, H.: *Dokumenten-Management*. VOI, 2005, S. 578

ZGB05f Zöller, B.: Gulbins, J.; Baumeister, H.: *Dokumenten-Management*. VOI, 2005, S. 469

ZGB05g Zöller, B.: Gulbins, J.; Baumeister, H.: *Dokumenten-Management*. VOI, 2005, S. 544-547

ZGB05h Zöller, B.: Gulbins, J.; Baumeister, H.: *Dokumenten-Management*. VOI, 2005, S. 118

Anhang A

```
<?xml version="1.0" encoding="UTF-8"?>
<nodeTypes xmlns:jcr="http://www.jcp.org/jcr/1.0"
xmlns:da="http://www.da.tum.de">
<nodeTypes>
      <nodeType name="da:da_content"
            hasOrderableChildNodes="false"
            isMixin="false"
            primaryItemName="jcr:data">
            <supertypes>
                  <supertype>nt:resource</supertype>
            </supertypes>
            <propertyDefinition name="da:created"
                  autoCreated="false"
                  mandatory="true"
                  multiple="false"
                  onParentVersion="COPY"
                  protected="false"
                  requiredType="Date" />
            <propertyDefinition name="da:createdBy"
                  autoCreated="false"
                  mandatory="false"
                  multiple="false"
                  onParentVersion="COPY"
                  protected="false"
                  requiredType="String" />
            <propertyDefinition name="da:lastModifiedBy"
                  autoCreated="false"
                  mandatory="true"
                  multiple="false"
                  onParentVersion="COPY"
                  protected="false"
                  requiredType="String" />
            <propertyDefinition name="da:title"
                  autoCreated="false"
                  mandatory="true"
                  multiple="false"
                  onParentVersion="COPY"
                  protected="false"
                  requiredType="String" />
      </nodeType>
</nodeTypes>
```