Enterprise Application Integration (EAI) in der Praxis - Erfahrungsbericht -

Marian Kuffner

Sun Microsystems GmbH – Sun Java Center
Sonnenallee 1 – 85551 Kirchheim-Heimstetten
Marian.Kuffner@Germany.Sun.COM

Abstract. Durch E-Business haben sich viele Firmen einen vielfältigen Mehrwert erhofft, wie z.B. umfassendere und einfachere Interaktion mit den Kunden, Reduzierung der Kosten usw. Gerade im letzten Jahr hat sich aber gezeigt, dass dazu verschiedenartige Geschäftsabläufe und -einheiten integriert und Informationen dazwischen ausgetauscht werden müssen. Das ist das Kernthema von "Enterprise Application Integration (EAI)". In diesem Beitrag werden zunächst die Bestandteile eines EAI-Systems beschrieben - vor allem das Integrationsmodell und die Basisdienste. Davon ausgehend wird ein Szenario aus dem Bankenwesen vorgestellt und diskutiert. Dabei wird detailliert auf die Vorgehensweise bei der Entwicklung einer EAI-Lösung eingegangen. Danach wird ein Framework vorgestellt, das die Entwicklung von Adapter zur Kopplung von Legacy-Systemen vereinfacht.

1 Einleitung

Während auf Hardware- und Netzwerkebene die Integration der Systeme weit fortgeschritten ist, sollte dies auf Softwareebene über Enterprise Application Integration (EAI) ermöglicht werden. Die Begriffe EAI und E-Business werden häufig unsachgemäß vermischt. Tatsächlich sind beide nicht verwandt oder voneinander abhängig. Allerdings spielt EAI eine wesentliche Rolle im elektronischen Handel, weil nur durch die Integration existierender Systeme ein wirklicher Mehrwert erzielt werden kann. Nur durch die Integration von diversen Geschäftsabläufen und -einheiten ergeben sich Informationen, mit denen z.B. die Kundeninteraktion einfacher und umfassender wird. Weiterhin ermöglicht ein weitergehender Informationsaustausch zwischen den vielfältigen IT-Systemen, dass Abläufe vereinfacht und Kosten eingespart werden können.

Verallgemeinernd kann man sagen, dass EAI immer dann zum Einsatz kommt, wenn monolithische, kundenspezifische Systeme (Legacy Systems) in die IT-Landschaft des Unternehmens integriert werden sollen. Das Ziel der EAI ist immer gleich: die beteiligten Systeme sollen möglichst lose und idealerweise ohne gegenseitige Abhängigkeiten miteinander gekoppelt werden.

EAI bezeichnet also den Integrationsprozess von mehreren, unterschiedlichen, voneinander unabhängig entwickelten Softwaresystemen, die meist auf unterschiedlichen und zum Teil inkompatiblen Technologien basieren. EAI soll die gemeinsame, uneingeschränkte Nutzung von Daten und Datenquellen ermöglichen und zwar über beliebige Anwendungen und Geschäftsprozesse hinweg.

In Kapitel 2 werden die Bestandteile eines EAI-Systems beschrieben: die Middleware als die grundlegende Infrastruktur, das Kommunikationsmodell für die Art des Informationsaustauschs, das Integrationsmodell für die Kopplung der einzelnen IT-Systeme und die Basisdienste mit vorgefertigten Komponenten. Anhand eines Beispiels aus dem Bankenwesen wird in Kapitel 3 ein Szenario dargestellt, wie man eine EAI-Lösung stufenweise konzipiert und realisiert.

2 Bestandteile und Anforderungen eines EAI-Systems

Um EAI besser zu verstehen, werden zunächst die Grundlagen und Bestandteile eines EAI-Systems vorgestellt. Dazu gehören:

- Die Middleware stellt die Infrastruktur für die Kommunikation in einem verteilten System bereit.
- Das Kommunikationsmodell legt fest, wie der Informationsaustausch über die Middleware erfolgt (z.B. synchron oder asynchron).
- Das Integrationsmodell beschreibt, wie die zu integrierenden Systeme gekoppelt werden. Dazu gibt es drei Möglichkeiten: auf der Ebene der Datenhaltung, über Nachrichten-Warteschlangen und auf der Basis von Geschäftsprozessen.
- Dienste stellen vorgefertigte Komponenten bereit, die im Rahmen der Integration wiederverwendet werden können.

2.1 Middleware

Eine einheitlich anerkannte Definition für Middleware gibt es noch nicht [ZB00]. Als Middleware könnte man jedoch eine Software bezeichnen, die Kommunikation zwischen zwei oder mehreren Systemen ermöglicht.

Die **Remote Procedure Call (RPC)** Technik [Lin00] ist die älteste Middleware, es gibt sie seit 1970. Sie erlaubt den synchronen Aufruf von Funktionen auf einem entfernten Computer über ein Netzwerk.

Message Oriented Middleware (MOM) [Lin00] beruht darauf, dass Sender und Empfänger Nachrichten austauschen. Vertreter dieser Kategorie sind Produkte wie z.B. MQSeries von IBM, iPlanet Message Queues (IMQ) und MessageQ von BEA Systems. Die Nachrichten (Message) werden über Warteschlangen ausgetauscht, die meist auch die Nachrichten persistent halten können. Dies ist ein wichtiger Aspekt, der vor allem zu einer höheren Ausfallsicherheit des EAI-Systems beitragen kann.

Distributed Object Technology [ZB00] erweitert das objektorientierte Konzept um die Möglichkeit, Objekte in einem Netzwerk verteilt zu halten. Auf diese Objekte kann man über Schnittstellen zugreifen. In diese Sparte gehören Technologien wie CORBA oder Java 2 Enterprise Edition (J2EE).

Transaction Oriented Middleware [ZB00] beinhaltet vor allem Transaction Processing Monitors (TPM). BEA Tuxedo und CICS von IBM sind die bekanntesten Produkte dieser Middleware-Kategorie. Sie gelten als die erste Generation von Application-Servern. Das Konzept beruht auf dem Transaktionsprinzip und gewährleistet die ACID-Eigenschaften (Atomicity, Consistency, Isolation, Durability).

Application Server [Lin00] enthalten all diese vorgenannten Technologien und besitzen in der Regel Schnittstellen und Protokolle für das Internet (z.B. http). Sie erlauben die Implementierung von Geschäftslogik in modernen Sprachen wie Java. Produkte in dieser Kategorie sind z.B. BEA WebLogic-Server, IBM Websphere, iPlanet Application Server.

2.2 Kommunikationsmodell

Das Kommunikationsmodell legt fest, wie Informationen mit Hilfe der Middleware ausgetauscht werden. Es gibt synchrone und asynchrone Kommunikationsmodelle.

Bekannte Möglichkeiten der *asynchronen* Kommunikation sind Message Passing, Publish/Subscribe und Broadcast. Bei der asynchronen Kommunikation fährt der Sender mit der Verarbeitung fort, ohne auf eine Rückmeldung des Empfängers zu warten. Message Passing wird mit Hilfe von Warteschlangen als Puffer zwischen Sender und Empfänger realisiert. Bei Publish/Subscribe verteilt der Sender seine Nachrichten nur an solche Empfänger, die zuvor ein Abonnement abgesetzt haben. Broadcasting ermöglicht das Senden einer Nachricht an alle möglichen Empfänger, die dann selbst entscheiden, ob diese Nachricht für sie wichtig ist oder nicht.

Die *synchrone* Kommunikation kann über Request/Reply, One-Way und Polling realisiert werden. Bei der synchronen Kommunikation wartet der Sender auf eine Rückmeldung vom Empfänger, bevor er mit der Verarbeitung fortfährt.

Nachrichten können auf zwei Arten vom Sender zum Empfänger gelangen: entweder durch ereignisgesteuertes Messaging oder durch Aufruf von definierten APIs (Application Programming Interfaces).

2.3 Integrationsmodelle

Es gibt drei Integrationsmodelle [Lub01], [Lin00], die hier nur kurz erläutert werden sollen. Dabei werden auch Vor- und Nachteile der einzelnen Modelle aufgezeigt.

Bei **Data-Level EAI** erfolgt die Integration von Systemen über den Austausch von Informationen auf der Ebene der Datenhaltung, wie z.B. bei Datenbanken via JDBC. Dieses stellt gleichzeitig den Einstiegslevel in die EAI dar und wird von vielen Systemen und Tools unterstützt.
Für die Integration von Systemen mittels Data-Level EAI ist eine Anpassung bzw. Erweiterung der einzelnen Systeme meist nicht notwendig, die Integration erfolgt in der Zielanwendung. Ein Austausch von Informationen kann auf diesem Level auch durch definierte Systemschnittstellen erfolgen.

Message-Level EAI unterstützt die Integration auf der Basis von Nachrichten. Informationen zwischen unterschiedlichen Systemen werden mittels Nachrichten ausgetauscht. Der Unterschied zum Data-Level EAI ist, dass der Nachrichtenaustausch direkt von den zu integrierenden Systemen unterstützt wird. Dies ermöglicht einen besser kontrollierbaren Informationsaustausch.
Dieser Ansatz ist mit mehr Aufwand in der Umsetzung verbunden, da existierende Systeme um eine neue Schnittstelle erweitert werden müssen, die das Senden und das Empfangen von Nachrichten ermöglicht. Hierdurch ergibt sich eine *enge* Kopplung zwischen den einzelnen Systemen.

Process-Level EAI geht einen Schritt weiter und ermöglicht die Integration auf der Basis von Geschäftsprozessen. Process-Level EAI ist die Erweiterung der Message-Level EAI, weil der Informationsaustausch zwischen den Systemen weiterhin über Nachrichten erfolgt. Die EAI-Middleware agiert dabei als eine Workflow-Engine, die das Routing der Informationen anhand von definierten Geschäftsprozessen (Workflow) im Gesamtsystem übernimmt.
Dieser Ansatz repräsentiert die beste Möglichkeit bei der Umsetzung eines EAI-Konzepts: es bietet eine *lose* Kopplung der Systeme und gewährleistet damit eine leichte Erweiterbarkeit bzw. Anpassbarkeit der Systeme. Durch den Einsatz einer Workflow-Engine kann die Kopplung leichter an das Geschäftsmodell angepasst werden. Dabei können auch ganze Komponenten (Schichten) ausgetauscht werden.
Dieses Integrationsmodell wird im EAI-Szenario in Kapitel 3.2 verwendet.

2.4 Services

Die Ausprägung der zur Verfügung stehenden Services ist vom Einsatzgebiet der EAI-Lösung abhängig. Meist werden in dieser Kategorie Services bereitgestellt, die sich für alle EAI-Implementierungen als nützlich erweisen können. Ein Beispiel für einen solchen Service ist die Fehler- und Ausnahmebehandlung. Diese schreibt z.B. beim Auftreten eines Fehlers entsprechende Informationen in eine Log-Datei und

kann z.B. über eMail einen Administrator alarmieren. Weitere mögliche Dienste wären die Konvertierung und Umwandlung von Datenformaten und die Verschlüsselung von Daten, die jedoch abhängig vom EAI-Umfeld sind.

2.5 Weitere Anforderungen an EAI

Die wichtigsten Bewertungskriterien lassen sich, wie bei allen Softwareprojekten, in die Kategorien Performance/Betrieb, Wirtschaftlichkeit und Technik unterteilen. In der nachfolgenden Übersicht sind die wichtigsten Anforderungen kurz aufgelistet:

Performance/Betrieb

- Netzlast und Datendurchsatz der einzelnen Systemkomponenten (Middleware, Adapter, Proxy, Workflow-Engine, etc. - siehe Kapitel 3) und des Gesamtsystems
- Anzahl und Verteilung der Benutzer (insgesamt und konkurrierend)
- Anzahl und Art der Transaktionen
- Skalierbarkeit und Verteilbarkeit der Komponenten (siehe Kapitel 3)
- Ausfallsicherheit und Recovery

Wirtschaftlichkeit

- Entwicklungsaufwand
- Administrationsaufwand
- Flexibilität bei Änderungen und Erweiterungen, wie z.B. bei der Integration neuer Produkte

Technik

- Art der Middleware
- Kommunikations- und Integrationsmodell
- Reifegrad der Technik
- Einsatz von Standardarchitekturen und Herstellerunabhängigkeit
- Möglichkeiten für Daten- und Schnittstellenkonvertierung

Bei der Auswahl eines geeigneten Produkts für die EAI müssen darüberhinaus die Anforderungen berücksichtigt werden, welche Middleware, Kommunikationsmodell, Integrationsmodell und Services betreffen.

Dabei ist die Analyse der zu integrierenden Systeme (Geschäftsprozesse, Schnittstellen, etc.) Grundvoraussetzung für die Erstellung eines Anforderungskatalogs für die Auswahl eines EAI-Produkts.

3 Umsetzung in der Praxis

In diesem Abschnitt wird zunächst ein Szenario vorgestellt, das Bestandteil eines aktuellen EAI-Projekts bei einer großen deutschen Bank ist. Das Projekt befindet sich noch in Entwicklung. Danach wird die Vorgehensweise zur Integration beschrieben, die sich über mehrere Stufen erstreckt. Ausgangspunkt ist die Analyse der zu integrierenden Systeme, um anschließend die Anforderungen an die künftige EAI-Lösung zu spezifizieren. Damit ist es möglich, einerseits die Produktauswahl vorzunehmen und andererseits die Gesamtarchitektur zu konzipieren. Abschließend wird auf die Architektur eines Frameworks eingegangen, das die Erstellung der Adapter für die individuellen (Teil-) Systeme wesentlich erleichtert.

3.1 EAI-Szenario

Die Handelssysteme einer Bank sollen mit einem Risikoanalysesystem gekoppelt werden. Wird ein Handel (z.B. Aktienkauf) über ein Handelssystem getätigt, sollen die Handelsdaten dem Risikoanalysesystem für die Berechnung einer Risikoanalyse zeitnah zugeführt werden. Nach der Berechnung werden die Ergebnisse nicht an das Handelssystem zurückgeschickt, sondern verbleiben für Reporting-Zwecke im Risikoanalysesystem.

Im Rahmen dieses Artikels wird die Kopplung von zwei Handelssystemen betrachtet. Die Handelssysteme übernehmen die Rolle der prozessauslösenden Systeme. Zwei weitere Systeme – die so genannten Datenlieferanten – dienen der Anreicherung der Handelsdaten mit zusätzlichen Geschäftsinformationen wie Geschäftspartner- und Marktdaten. Letztere werden von Systemen wie z.B. Teledata, Reuters, Bloomberg geliefert. Das Risikoanalysesystem übernimmt in diesem Szenario die Rolle des Datenkonsumenten. Die Grobarchitektur des Gesamtsystems ist in der nachfolgenden Abbildung skizziert.

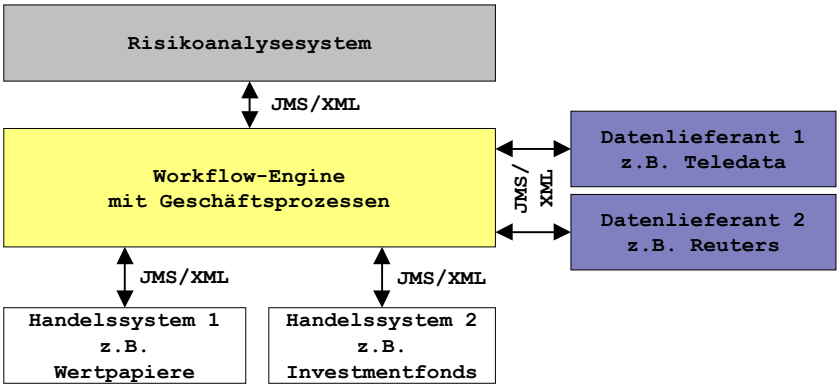


Abbildung 1: Grobarchitektur des Gesamtsystems

Bei den zu integrierenden Systemen handelt es sich um Legacy-Systeme, die auf unterschiedlichen Hardware- und Softwareplattformen im heterogenen Netzwerk der Bank laufen. Sie verfügen derzeit über keine Schnittstellen, die die Übermittlung der notwendigen Daten (z.B. in Form von XML-Dokumenten) ermöglichen. Jedes dieser Systeme hat aus softwaretechnischer Sicht eine eigene Datenhaltung.

Eine weitere Anforderung an das EAI-System ist die Verarbeitung von XML-Daten als zukunftssicheres Austauschformat.

3.2 Vorgehensweise

Bei der Analyse und Definition der Architektur des Gesamtsystems kann eine Trennung zwischen fachlichem Teil (Geschäftprozess) und technischem Teil (Schnittstellen, Middleware) erfolgen. Das führt zu einer besseren Rollenverteilung im Projekt. Der Fachbereich definiert, welche Daten aus den Handelssystemen und aus den Datenlieferanten dem Risikoanalysesystem übermittelt werden, und welche Semantik sie haben. Im technischen Bereich wird dann das Datenformat und die notwendige Funktionalität der Infrastruktur (Kopplung der zu integrierenden Systeme über Adapter, EAI-Produkt, etc.) für die Übermittlung der Daten definiert.

Bei der Konzeption der Architektur kann auf eine Reihe von Patterns [Lut01] zugegriffen werden. Diese sind analog zu den bekannten Design-Patterns [GHJV95] und zeigen Lösungswege für verschiedene Problemstellungen im EAI-Umfeld.

Als Middleware wird MQSeries von IBM eingesetzt, weil diese Middleware in der IT-Landschaft der Bank bereits eingesetzt wird. Weiterhin bietet sie einige Vorteile gegenüber anderer Middleware. MOM erlaubt eine lose Kopplung der Systeme. Ferner bietet MQSeries eine persistente Datenhaltung der Nachrichten, so dass im Fehlerfall keine Nachrichten verloren gehen können.

Als Kommunikationsmodell (siehe Kapitel 2) wurde die asynchrone Kommunikation gewählt, um eine bessere Entkopplung der einzelnen Systeme zu erreichen. Als Integrationsmodell wurde das Process-Level EAI gewählt, damit eine leichtere Erweiterung und Anpassung der EAI-Lösung möglich ist. Nach der Evaluierung mehrerer Produkte anhand der definierten Anforderungen, fiel die Wahl auf iPlanet Integration Server (iIS, vormals Forté Fusion) aus folgenden Gründen (siehe auch Abbildung 2):

- Dieses Produkt verfügt über eine *Workflow-Engine*, die das Ausführen von vordefinierten Geschäftsprozessen ermöglicht. Diese können mit Hilfe der mitgelieferten grafischen Entwicklungsumgebung modelliert werden.
- Ferner bietet das System die Möglichkeit, über *Java Messaging Service* (JMS) [JMS99] mit der ausgewählten Middleware zu kommunizieren. Als alternative Protokolle werden http und IIOP unterstützt.

- Die *native Unterstützung von XML* [XML00] durch die iIS-Komponenten Workflow-Engine und Proxy war ein weiteres Kriterium, das zur Entscheidung für dieses EAI-Produkt beitrug.
- Die Workflow-Engine erlaubt die Verarbeitung von XML-Dokumenten und dient gleichzeitig zur Verwaltung der Prozesse, die auf Grund von Statusübergängen in den Handelssystemen initiiert werden. Den einzelnen Prozessschritten (Aktivitäten) sind ein oder mehrere Proxies zur Kommunikation mit den angebundenen Systemen über ihre jeweiligen Adapter zugeordnet. Die Workflow-Engine, Proxies und Adapter können in vielfältigen Kombinationen instanziiert werden, um das EAI-System zu skalieren. Die Kommunikation zwischen der Workflow-Engine und den Proxies wird vom iIS Message Broker gesteuert.

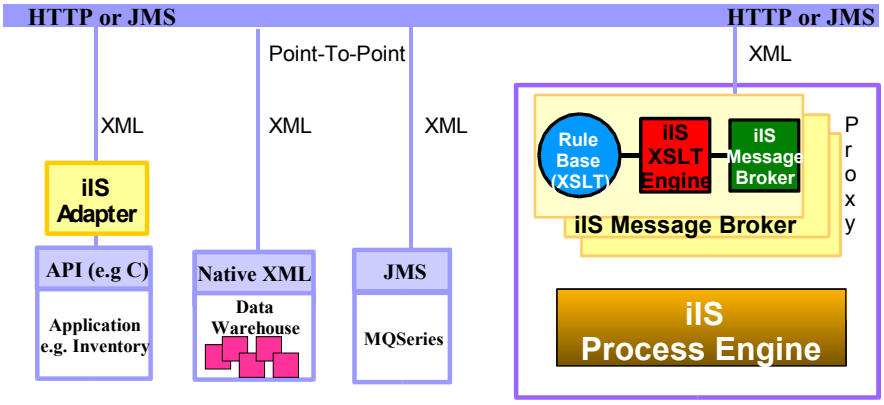


Abbildung 2: iIS Funktionalität

Um die Wartbarkeit zu verbessern, die Weiterentwicklung des Gesamtsystems zu erleichtern und das Process-Level EAI Integrationsmodell umzusetzen, wurden einige Dienste (Services) definiert, die in den Workflow des EAI-Systems integriert wurden. Dazu zählen Dienste wie Plausibilitätsprüfung, Fehlerbehandlung und Logging. Aus Sicht der Workflow-Engine werden die Systeme, die diese Dienste implementieren, quasi als Legacy-Systeme betrachtet, die mittels Proxies über eine Point-to-Point JMS-Verbindung, d.h. ohne Adapter, angesprochen werden.

3.3 Adapter

Da keines der integrierenden Legacy-Systeme über eine passende Schnittstelle verfügt, muss die Kopplung über so genannte Adapter erfolgen (siehe Abbildung 2). Die Legacy-Systeme haben zwar klassische APIs, sie werden aber durch ihren jeweiligen Adapter „XML-fähig“ gemacht, d.h. ein Adapter transformiert empfangene XML-Dokumente bzw. Nachrichten in eine proprietäre Datenstruktur des Legacy-

Systems (z.B. ein C-struct). In der umgekehrten Richtung transformiert der Adapter proprietäre Datenstrukturen in zu versendende XML-Dokumente (siehe Abbildung 3). Die XML-Schnittstelle, die ein solcher Adapter quasi stellvertretend für das zugehörige Legacy-System anbietet, ist dienstorientiert und bietet eine Reihe von Diensten mit den zugehörigen Ein- und Ausgabeparametern an. Diese Dienste können dann von einem anderen Legacy-System aufgerufen werden (service requester) oder von einem Adapter erbracht werden (service provider). In Abbildung 3 wird beispielhaft skizziert, wie ein zu integrierendes System mit der Workflow-Engine kommuniziert.

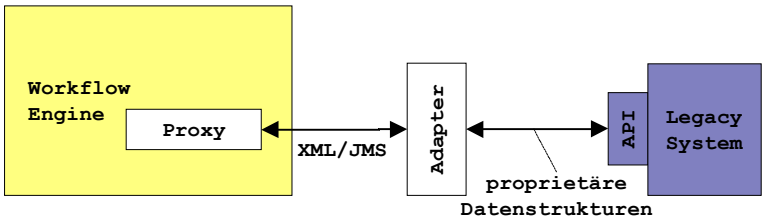


Abbildung 3: Anbindung eines Systems mittels Adapter an die Workflow-Engine

Innerhalb der Workflow-Engine befindet sich ein so genannter Proxy, der das Legacy- System innerhalb der Workflow-Engine repräsentiert. Durch den Proxy kann aus dem Workflow heraus über den dienstorientierten Adapter auf das Legacy-System zugegriffen werden. Im iIS unterstützt jeder Proxy die Verarbeitung von XML-Dokumenten mittels XSLT (XML Stylesheet Language Transformation) [XSLT99], wie es in Abbildung 2 dargestellt ist. Dabei können unterschiedliche XSLT-Regeln jeweils für ein- und ausgehende XML-Dokumente definiert werden. Auf der Proxy-Seite können über diesen Mechanismus Daten aus dem XML-Dokument extrahiert und Variablen zugewiesen werden, die zur Ablaufsteuerung des Geschäftsprozesses dienen. In der umgekehrten Richtung können über ein Proxy Variablen in ein XML-Dokument geschrieben werden.

3.4 Java Adapter Framework

Architektur und Überblick

Um die Entwicklung von Adaptern zu unterstützen, wurde ein Java Adapter Framework (JAF) realisiert, das in diesem Abschnitt vorgestellt wird. Das Hauptziel des JAF ist die Vereinfachung der Realisierung von systemspezifischen Adaptern. Dieses wird erreicht durch die Implementierung und Bereitstellung wiederverwendbarer Komponenten, die die Hauptaufgaben eines Adapters abbilden.

Folgende Aufgaben des Adapters wurden bei der Definition des JAF berücksichtigt:

- Der Adapter empfängt und verarbeitet XML-Dokumente als Nachrichten über MQSeries.
- Der Adapter transformiert empfangene XML-Dokumente in eine systemspezifische Datenstruktur. Dabei verarbeitet er diese Daten nach einer nur ihm bekannten Logik.
- Der Adapter leitet die transformierten Daten an das Zielsystem über den Aufruf einer Funktion des System-APIs weiter.
- Außerdem kann der Adapter auch neue XML-Dokumente erzeugen, um ggf. Rückmeldungen des Legacy-Systems an den Aufrufer weiterzuleiten.

Die Architektur des JAF ist in Abbildung 4 dargestellt.

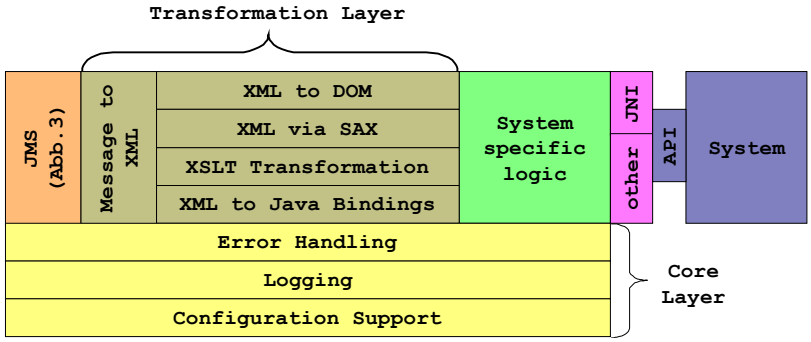


Abbildung 4: JAF-Architektur

Die Datenübertragung zum und vom Adapter erfolgt mit JMS. Das nachgelagerte Modul *Message to XML* übernimmt die Konvertierung zwischen XML und dem Nachrichtenformat des MOM-Produkts, also MQ Series.

Im Transformation Layer stehen weitere Transformationsmodule zur Verfügung. Das XML-Dokument kann z.B. mit Hilfe des *XML to DOM*-Moduls in ein DOM-Modell [DOM00], [Mor00] transformiert werden. Ebenso kann ein XML-Dokument durch das Modul *XSLT Transformation* in einen Stringbuffer überführt werden. Eine Alternative dazu stellt das Modul *XML to Java Bindings* dar, in dem mit Hilfe eines Compilers bzw. Generators die Abbildung des XML-Dokuments auf Java-Klassen (statischen Code) erfolgt. Hier besteht außerdem die Möglichkeit, einen SAX-Parser [SAX00], [Mor00] zu verwenden, um die Daten aus dem XML-Dokument direkt in eine systemspezifische Datenstruktur zu transformieren. Diese Module basieren auf dem Java XML Parsing API (JAXP) [JAXP01]. Damit ist der Austausch sowohl des XML-Parsers als auch des XSLT-Prozessors jederzeit ohne Änderungen/ Erweiterungen an dem Framework möglich.

Das Modul *System Specific Logic* ist für die Implementierung der systemspezifischen Funktionalität zuständig, wie z.B. fachliche Plausibilitätsprüfungen.

Typische Schnittstellen eines zu integrierenden Systems sind in der Sprache C oder in Skriptsprachen realisiert. Für eine C-Schnittstelle wird das *JNI*-Modul (Java Native Interface [JNI97]) verwendet.

Im *Core Layer* stehen Basisfunktionalitäten zur Verfügung, die von allen Modulen genutzt werden: Error Handling, Logging und Configuration Support.

Auf der Basis des hier vorgestellten Frameworks können Adapter schneller und effektiver implementiert werden. Die Auswahl eines geeigneten Transformationsmoduls bleibt dabei dem Entwickler überlassen. Für das Modul *XML to Java Bindings* wird noch nach einem Produkt auf Basis von JAXB (Java API for XML/Java Binding) [JAXB01] gesucht.

Die einzelnen Komponenten und ihr Einsatz in der Praxis
Im Projekt werden hauptsächlich Adapter auf der Basis des Moduls *XSLT Transformation* realisiert. Zu dieser Entscheidung führen vor allem folgende Überlegungen:

- Die Anpassung und/oder Erweiterung der Transformationslogik ist beim Einsatz von XSLT-Regeln leichter möglich. Der Code des Adapters bleibt unverändert, es werden lediglich die XSLT-Regeln, die in Textform vorliegen, angepasst und/oder erweitert. Das führt zu einem verkürzten Releasezyklus.
- Im konkreten Einsatzfall sind die XSLT-Regeln statisch, d.h. sie können sich zur Laufzeit nicht ändern. Diese Randbedingung stellt gleichzeitig eine Optimierungsmöglichkeit dar: der XSLT-Prozessor wird beim Start des Adapters mit den entsprechenden Regeln initialisiert und steht dann als wiederverwendbares Objekt zur Laufzeit zur Verfügung. Eine erneute Initialisierung entfällt.
- Falls bestimmte Funktionalität mit den Standard-XSLT-Sprachelementen nicht abgebildet werden kann, stehen hier standardisierte Erweiterungsmöglichkeiten zur Verfügung, und zwar durch die Definition von eigenen Sprachelementen und/oder Erweiterungsfunktionen. Diese können ebenfalls in Java implementiert und in den XSLT-Prozessor bzw. in die XSLT-Regeln integriert werden.

Eine weitere Randbedingung war, dass abhängig von bestimmten Inhalten des XML-Dokuments, unterschiedliche XSLT-Regeln angewandt werden sollten. Bei der Umsetzung dieser Anforderung stand der Performance-Aspekt im Vordergrund. Dies führte zu folgender Lösung: anstatt den XSLT-Prozessor ständig mit neuen Regeln zur Laufzeit neu zu initialisieren, wird ein Pool von vorinitialisierten XSLT-Prozessoren implementiert. Der Pool wird beim Starten des Adapters initialisiert. Anhand der definierten Kriterien wird von der Adapterlogik zur Laufzeit der vorinitialisierte Prozessor aus dem Pool ausgewählt. Dieser Lösungsansatz führt zwar

zu einem höheren Speicherbedarf, ermöglicht jedoch die performante Ausführung der notwendigen Transformation. Die Dauer der Transformation von XML-Dokumenten mit gleichen Unterscheidungsmerkmalen bleibt nahezu konstant.

Das Framework wurde primär für die Entwicklung von Adaptern entwickelt, im Laufe des Projekts hat sich jedoch herausgestellt, dass das JAF einen sehr hohen Wiederverwendungsgrad besitzt: es diente als Basis auch für die Implementierung weiterer Dienste, wie z.B. Plausibilitätsprüfung. In diesem Dienst werden die XML-Dokumente Prüfungen unterzogen, die sowohl die Struktur als auch die Inhalte eines XML-Dokuments anhand vorgegebener Prüfvorschriften validieren. Hier kommen verstärkt die Module *XML to DOM* und *XML via SAX* zum Einsatz.

Mit *XML via SAX* können sehr effizient kleine Mengen von Elementen aus dem XML-Dokument geparsed und damit auf Existenz geprüft werden.

Die Strukturprüfung eines XML-Elements kann performant mit Hilfe des *XML to DOM* Moduls realisiert werden. Der Vorteil dabei ist, dass das Objektmodell (DOM) nur einmal aufgebaut wird und dann im Speicher zur Verfügung steht. Es können beliebig viele Überprüfungen von Teilstrukturen durchgeführt werden.

4 Fazit und Ausblick

Die vorgestellten EAI-Bestandteile sollten zum besseren Verständnis eines EAI-Systems und zur leichteren Auswahl geeigneter EAI-Produkte beitragen. Das in diesem Beitrag vorgestellte EAI-Szenario auf der Basis des prozessorientierten EAI-Integrationsmodells bildet das Rückgrat der EAI-Lösung zur Datenversorgung in der IT-Landschaft der Bank. Flexibilität, Offenheit des Gesamtsystems und damit auch Zukunftssicherheit wird bei der vorgestellten EAI-Lösung durch den Einsatz der Workflow-Engine auf XML-Basis erreicht.

Die Entwicklung und Wartung der Adapter wurde durch die Bereitstellung des JAF-Frameworks vereinfacht, das auch als Basis für die Implementierung weiterer Dienste dienen kann. Java und XML sind das „Yin und Yang“ [Dim01] des E-Business: die Java-Plattform bringt die Portabilität der Applikationen mit sich, während XML die Portabilität der Daten sicherstellt. Zusammen bringen sie die heute geforderte Unabhängigkeit von der Infrastruktur und die System-Offenheit im Unternehmen.

Um EAI-fähig zu sein, sollten neu entwickelte Applikationen mit einem dienstorientierten, XML-fähigen API ausgestattet werden. Dies ermöglicht eine leichtere Integration ohne Adapter.

5 Literaturverzeichnis

Dim01	Carla Dimond: The e-Business Tao, EAI-Journal Mai 2001, http://www.eaijournal.com
DOM00	W3C Recommendation: Document Object Model (DOM) Level 2 Core Specification Version 1.0, 2000, http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113
GHJV95	Gamma, E., Helm, R., Johnson, R., Vilissides, J.: Design Patterns, Addison-Wesley, 1995
JAXP01	Sun Microsystems Inc., Java API for XML Processing Specification Version 1.1 - Final Version, 2001, http://java.sun.com/xml/jaxp-1_1-spec.pdf
JAXB01	Sun Microsystems Inc., Java API for XML/Java Binding, Java Community Process JSR-031, http://java.sun.com/aboutJava/communityprocess/jsr/jsr_031_xmld.html
JMS99	Sun Microsystems Inc., Java Message Service Version 1.0.2, 1999, http://java.sun.com/products/jms/index.html
JNI97	Sun Microsystems Inc., Java Native Interface Specification Version 1.1,1997, http://java.sun.com/products/jdk/1.2/docs/guide/jni/
Lin00	Linthicum, David, S.: B2B Application Integration – e-Business-Enable Your Enterprise, Addison-Wesley Professional, 2000
Lub01	Boris Lublinsky: Achieving the Ultimate EAI Implementation, EAI-Journal Februar 2001, http://www.eaijournal.com
Lut01	Jeffrey C. Lutz: EAI Architecture Patterns, EAI-Journal März 2001, http://www.eaijournal.com
Mor00	Morgenthal, JP: Enterprise Application Integration with XML and Java, Prentice Hall, 2000
SAX00	Simple API for XML Version 2.0, http://www.megginson.com/SAX/index.html
XML00	W3C Recommendation: Extended Markup Language XML 1.0 Second Edition, 2000, http://www.w3.org/TR/2000/REC-xml-20001006
XSLT99	W3C Recommendation: XSL Transformation, Version 1.0, 1999, http://www.w3.org/TR/1999/REC-xslt-19991116
ZB00	Zimmermann, J., Benecken, G.: Verteilte Komponenten und Datenbankanbindung, Addison-Wesley, 2000