DR. SIEGERT & PARTNER

Anwendungsintegration mit XML in Finanzinstituten

Dr. Helmut Siegert

Juni 2001

Vorwort

Vor dem Hintergrund des technologischen wie auch organisatorischen Wandels sehen sich die Banken und deren Rechenzentralen einigen großen Herausforderungen gegenüber:

- Anwendungen und/oder Daten müssen – dem „New Product Process“ entsprechend schnell - integriert werden
- Abläufe sind zu standardisieren
- die „Multikanalstrategie“ verlangt nach einer konsistenten Informationsversorgung über alle Vertriebswege und Prozesse hinweg.

Gleichzeitig zeichnet sich unter dem Stichwort „XML“ (eXtensible Markup Language) eine Basistechnologie ab, die

- an technologischer Reife und Stabilisierung ständig zunimmt
- durch die Entwicklung von Standardwerkzeugen begleitet wird
- ihr Anwendungspotenzial ständig vergrößert.

Dies mag entscheidend dazu beigetragen haben, dass z. B. das SIZ-Direktorium am 19. Oktober 2000 einen Grundsatzbeschluss dahingehend getroffen hat, XML als geeignete Basistechnologie für Anwendungen und für die Definition von Datenstandards und Austauschformaten für die DV in der Sparkassen-Finanzgruppe zu empfehlen.

Die besondere Bedeutung von XML hat wesentlich damit zu tun, dass XML durch das WorldWideWeb-Consortium (W3C) zu einem echten Standard erhoben wurde. Damit hat XML a priori das Potenzial zur Vereinheitlichung der Datenverarbeitung. Zu prüfen ist, inwiefern sich dieses Potenzial, unter Berücksichtigung der gegebenen technologischen Restriktionen seitens einer Rechenzentrale (u.a. hoher Datentransfer, Batch-Betrieb, Großrechner-Einsatz) realisieren lässt.

Das „Thema“ XML wird im Schrifttum dem Leser häufig durch die Darstellung (i) XML als „Daten“, (ii) darstellbare „Dokumente“ und (iii) „Gestaltungsmittel“ für Dokumente näher gebracht.[1] Der Focus der vorliegenden Arbeit liegt primär auf erstere – d.h. auf Anwendungen von XML als Schnittstelle zwischen verschiedenen Programmen und Systemen.

Der Autor der Studie hat sich frühzeitig mit dieser neuen Basistechnologie beschäftigt; seine Projekterfahrungen sind, zusammen mit den Ergebnissen aus Literatur-Recherchen und aus Gesprächen mit Lösungsanbietern, in die vorliegende Studie eingeflossen.

[1] so auch R. Anderson u.a.: XML professionell, Bonn 2000

Inhaltsverzeichnis

Abbildungsverzeichnis

Glossar

In der XML-Welt haben sich einige Grundbegriffe verfestigt, die im folgenden erläutert werden.

Grundbegriffe	Erläuterung
eXtensible Markup Language - XML	XML ist eine Metasprache; die Festlegung des unabhängigen Standards findet auf 2 Ebenen statt: • der XML-Standard selbst wird vom World Wide Web Consortium (W3C) betreut und weiterentwickelt • die jeweiligen Anwendungen von XML werden von freien Benutzergruppen erstellt XML-Dokumente bestehen hauptsächlich aus Text und Tags; die Tags erzeugen eine Baumstruktur im Dokument. Wenn eine ordnungsgemäße Verschachtelung der Tags vorliegt, wird es als „well-formed" bezeichnet. Sofern zusätzlich eine DTD für das Dokument existiert, wird es als „valid" deklariert.

Document Type Definition - DTD	in einer DTD werden Strukturmerkmale (Tag-Strukturen) für Dokumente festgelegt: für den Verfasser stellen sie eine Vorgabe dar, für den Programmierer legen sie fest, welche Strukturen verarbeitet werden müssen. Dadurch, dass DTD's lesbar sind, ist auch dem Leser transparent, welche Datenstrukturen es in den Dokumenten gibt. DTD steuern XML-Werkzeuge und prüfen XML-Dokumente auf strukturelle Gültigkeit – für das Verständnis eines XML-Dokumentes sind sie nicht erforderlich.
	
	DTD am Beispiel eines FRA (aus der FpML-Bibliothek:
	`<!-- Copyright (c) 1999 by J.P.Morgan and Pricewaterhouse Coopers. PricewaterhouseCoopers refers to the individual member firms of the world wide PricewaterhouseCoopers organization. All rights reserved.-->`
	`<!-- version 1.0b2 : August 6, 1999 -->`
	`<!ENTITY % ftfra.dtd SYSTEM "ftfra.dtd">`
	`%ftfra.dtd;`
	`<!ELEMENT fpfra:ForwardRateAgreement (fpfra:productID , fpfra:productType , fpfra:forwardRateAgreement )>`
	`<!ATTLIST fpfra:ForwardRateAgreement xmlns:fpfra NMTOKEN #FIXED 'urn:fpml-Product-Fra' >`
	`<!ELEMENT fpfra:productID (#PCDATA )>`
	`<!ATTLIST fpfra:productID e-dtype NMTOKEN #FIXED 'string' >`
	`<!ELEMENT fpfra:productType (#PCDATA )>`
	`<!ATTLIST fpfra:productType e-dtype NMTOKEN #FIXED 'string' >`
	`<!ELEMENT fpfra:forwardRateAgreement (ftfra:ForwardRateAgreementTemplate )>`
	
	DTDs werden in Zukunft durch die neue XML-Schema-Methode ersetzt werden. XML-Schemata sind (im Gegensatz zu DTDs) in XML geschrieben und haben erweiterte Funktionalitäten wie Datentypisierung, Vererbung und Präsentationsregeln – auf diese Weise haben XML-Anwendungen, die strukturierte Daten aus Datenbanken/Geschäftsanwendungen verarbeiten, wesentlich mehr Informationen über die einzelnen Elemente

XML Schema	das „XML-Schema" Proposed Recommendation fokussiert eines der gravierendsten Probleme in XML, das Nichtvorhandensein von Datentypen. DTDs erlauben zwar die Tags und die Struktur einer Dokumentklasse festzulegen, jedoch sind der Inhalt von Dokumentelement und die Werte von Attributen schlichter Text. XML-Schema führen nun Typen wie Zahl, Datum, Zeit, usw. in XML ein und erlauben auch benutzerdefinierte Datentypen. Daneben wird die Modularisierbarkeit von XML-Schemata unterstützt, was die Wiederverwendbarkeit verbessert. Im Gegensatz zu DTD's sind XML-Schemata in XML geschrieben.

Grundbegriffe	Erläuterung
	XML-Schema eignet sich insbesondere für den Datenaustausch und die Transaktionsverarbeitung (Datenorientierung) - und weniger für dokumentenorientierte Umgebungen. Die zahlreichen Features missfällt denn auch (aufgrund der höheren Komplexität gegenüber DTDs) dokumentenorientierten Anwendern
Document Object Model – DOM	DOM ist ein hierarchisches API (Application Programming Interface) für HTML- und XML-Dokumente. Es erlaubt Anwendungsprogrammen, in der Struktur von Dokumenten zu navigieren, einzelne Elemente oder Attribute zu lesen, hinzuzufügen, zu verändern, oder zu löschen. DOM ist eine plattform- und sprachneutrale Schnittstelle, die ein Standardmodell dafür bietet, (a) wie die Objekte in einem XML-Dokument zusammengesetzt werden, und (b) wie der Zugriff auf diese Objekte bzw. das Manipulieren ihrer Beziehungen erfolgen soll
SimpleAccess API for XML - SAX	wenn DOM verwendet wird, erfährt die Anwendung, was im XML-Dokument steht, indem sie Referenzen auf Objekte, die sich im Speicher befinden, verfolgt. Bei der Verwendung von SAX teilt der Parser der Anwendung mit, was sie zu tun hat, indem er das Dokument als einen Strom von Ereignissen interpretiert
eXtensible Style Language – XSL	das Erscheinungsbild eines XML-Dokuments wird in einem Style Sheet festgelegt, das mit XSL erstellt wird – es gehört zu den Grundgedanken von XML, Inhalt und Darstellung zu trennen. Damit ist es möglich, Darstellungsvorgaben für Dokumente zu ändern, ohne dass die Daten selbst bearbeitet werden müssen. Es kann mehrere XSL-Sheets für dasselbe Dokument geben – so können die Darstellungen an die Formatierungsmöglichkeiten des jeweiligen Ausgabegerätes angepasst werden (Bildschirm, Drucker, pdf-Format, ...). XSL-Style-Sheets sind selbst wieder XML-Dokumente – aus diesem Grund lassen sie sich mit den gleichen Werkzeugen wie XML bearbeiten
eXtensible Style Language Transformations - XSLT	XSLT ist eine Untermenge von XSL (i.S. einer Transformationssprache), die darauf ausgelegt wurde, XML-Dokumente in andere XML-Dokumente umzuwandeln. XSLT wurde als Teil von XSL, einer Stylesheet-Sprache für XML, entwickelt. XSL verfügt über den Umfang von XSLT hinaus über ein XML-Vokabular für die Angabe von Formatierungsinformationen
XPointer und XLink	XLink und XPointer legen die Konventionen für das Verweisen/die Definition von Verweiszielen auf Objekte (Linking) in XML-Dokumenten fest
XPath	Syntax zur Adressierung von Positionen in XML-Dokumenten. Grundlage von XSLT und XPointer
XML Query Language – XQL	XQL ist ein Vorschlag für eine Abfragesprache von XML-Dokumenten. Ähnlich wie SQL dient XQL dazu, XML-Dokumente von einer XML-Datenquelle abzurufen. Queries werden auf einzelnen Dokumenten oder auf festen Collections von Dokumenten operieren. Sie können ganze Dokumente oder definierte Teilbäume durchsuchen und neue Ergebnisdokumente auf Basis der Suchergebnisse konstruieren
XML-Pseudostandards	in der Praxis gibt es bereits zahlreiche Initiativen zur Einführung von XML als Standard für den Datenaustausch. In der Finanzbranche sind u.a. folgende XML-Anwendungen in der Entwicklung: FRML – EXtensible Financial Reporting Markup Language FpML – Financial Product Markup Language FinXML - XML for Capital Markets
XML-Parser	ein XML-Parser ist ein Verarbeitungsprogramm, das ein XML-Dokument liest und die Struktur sowie die Eigenschaften der Daten ermittelt, indem es die ‚tags' von den Daten trennt. So kann ein Anwendungsprogramm ein Element identifizieren und seinen Inhalt verarbeiten. Es akzeptiert Dokumente, die den Status „well formed" (syntaktisch richtig) oder „valid" (strukturkonform bez. einer DTD) haben. „Gültigkeit" setzt „Wohlgeformtheit" voraus
Standardisierung	• W3C ist ein vom WorldWideWeb Consortium empfohlener Industriestandard. Seit Februar 1998 ist XML 1.0 eine „Recommendati-

Grundbegriffe	Erläuterung
	on“ des W3C • OASIS ist 1993 gegründet worden (Organization for the Advancement of Structured Information Standards), um den Austausch von Dokumenten zu fördern. Ursprünglich war OASIS auf SGML fokussiert, es konzentriert sich jedoch immer stärker auf XML • XML.ORG ist ein Web-Portal, das von OASIS betrieben wird. Dort können aktuelle Informationen/Referenzen zum Einsatz von XML bezogen werden • BIZTALK ist eine Microsoft-Initiative, um die breitbandige Einführung und Implementierung von XML für Electronic Commerce und Applikationsintegration zu unterstützen. Im BizTalk Framework werden auf Basis des W3C-Standards Richtlinien zur Anwendung und Publikation von XML-Schemata festgelegt
Unicode	weltweit verwendeter Zeichencodierungsstandard auf der Basis einer 16-Bit-Codierungseinheit, die von der Unicode, Inc. entwickelt wurde. Unicode benötigt zwar den doppelten Speicherplatz, kann dafür aber außer lateinischen Buchstaben und arabischen Zahlen auch andere Alphabete und Schriften darstellen. Zudem können mathematische und technische Sonderzeichen codiert werden. Unicode wird von der ISO genormt und liegt in der Version 3.0 vor
SOAP Simple Object Access Protocol	SOAP ist ein Protokoll, das dabei hilft, Daten zu serialisieren und in einen XML-Umschlag einzupacken, sodass ein Transport zwischen Endpunkten realisiert werden kann (entsprechende W3C-Aktivitäten laufen unter dem Stichwort XML Protocol)
Resource Description Framework - RDF	RDF legt ein abstraktes Verfahren fest, um die Zusammenarbeit zwischen Entitäten und deren Merkmale zu beschreiben (Entity-Relationship-Modellierung). Es ist per se an keine bestimmte Syntax gebunden, andererseits setzt der Austausch von konkreten Metadaten eine solche aber voraus. Innerhalb des W3C wurde deshalb eine RDF-Spezifikation auf Basis von XML entwickelt. Ergebnis dieser Standardisierungsbemühungen („semantisches Web“) ist u.a. RDF Schema, womit sich Grammatiken für bestimmte Arten von Metadaten entwickeln lassen. Dazu müssen Klassen von Ressourcen und Eigenschaften spezifiziert sowie deren Beziehungen untereinander dargelegt werden, wodurch ein konsistener Bauplan für eine umfassende Metadaten-Architektur entsteht. Damit kommt RDF Schema bei der Definition von RDF-Anwendungen die gleiche Aufgabe zu, die XML Schema für XML-Applikationen übernimmt. Mit RDF kann jeder seine individuell beschriebenen RDF-Datenmodelle mit den Modellen der Außenwelt kombinieren und nutzen. So können RDF Queries und Applications auf einer gemeinsamen Basis operieren

0 Management Summary

0.1 Problemstellung: Integration von Anwendungen

Die Integration von Anwendungen (bspw. im Umfeld einer Gesamtbanksteuerung) verlangt ....

1. Konsistenz hinsichtlich der Sicherstellung, dass
   - ein bestimmtes Ergebnis in unterschiedlichen Anwendungen identisch ist
   - unterschiedliche Ergebnisse sich ineinander überführen lassen.

   Damit sind folgende Herausforderungen verbunden:
   - konsistente Definition aller Finanzprodukte in einem Finanzinstitut
   - konsistente Methoden über alle Geschäftseinheiten
   - konsistentes Reporting i.S. einer einheitlichen Ergebnisdarstellung.
2. Integration der Bankgeschäfte über alle Risikotypen hinweg und Überführbarkeit der Risikoergebnisse in die interne Ergebnisrechnung und das externe Rechnungswesen
3. Offenheit i.S. einer schnellen Integrationsfähigkeit neuer Produkte, Methoden und/oder Prozesse durch Standardschnittstellen und generische Prozeduren
4. Effizienz der Prozesse, d.h.
   - Nutzung von Hilfsmitteln in mehreren Anwendungen
   - eine Schnittstelle zu vielen Anwendungen
   - Reduktion des Reportings auf das Wesentliche – aber Nachvollziehbarkeit bis ins Detail.

0.2 Lösungspotenzial via XML

Bevor die Frage beantwortet werden kann, *wie* XML helfen kann, Anwendungsintegration in dem o.g. Sinne zu unterstützen, ist es unabdingbar, eine Vorstellung von dem zu haben, *was* XML grundsätzlich zu leisten vermag:

- Die Syntax von XML ist Grundlage für alle möglichen Anwendungsbereiche. Wohlgeformte XML-Dokumente entsprechen in ihrer Syntax den Anforderungen der W3C-Spezifikation von XML 1.0. Diese „Wohlgeformtheit“ kann durch einen Parser überprüft werden; ein sog. validierender Parser ist dagegen in der Lage, ein Dokument auf Gültigkeit im Hinblick auf seinen Bauplan zu überprüfen.
- Die Entwicklung einer eigenen Auszeichnungssprache verlangt die Definition von Regeln. Baupläne (DTDs/Schemata) enthalten alle Definitionen von neuen Elementen und die Regeln für deren Anwendung. Daher können über sie Dokumente auf ihre Gültigkeit hin überprüft und die Struktur von XML-Dokumenten zur Laufzeit erkannt werden.
- Ein wichtiger Schlüssel zum Erfolg einer XML-Anwendung ist die Wahl der richtigen Auszeichnungssprache, d.h. einer Menge von Elementen, ihren Attributen und den Regeln für die Anwendung der Elemente – und ihre problemadäquate Modellierung. Hierbei genügt es nicht, nur das Wesen einer Problemstellung („operative Sicht“) einzufangen, die Beschreibung muss auch dem angestrebten (Daten-) Verarbeitungsprozess („dispositive Sicht“) entgegenkommen. Bei dieser Datenmodellierung gibt es keine festen Regeln und keine Standard-Algorithmen, die man anwenden könnte.
- Liegt die Problemstellung in geeigneter Sprache vor, gibt es weitere Aspekte, die zu beachten sind: Für die Manipulation von XML-Elementen gibt es zwei API – das Document Object Model (DOM), das eine baumartige Sicht auf ein Dokument entwirft, und das Simple API for XML (SAX), das verschiedene Arten von Ereignissen während der Abarbeitung eines Dokuments erzeugt (in der Form:

„hier ist ein Start-Tag, hier beginnt der Inhalt eines Dokuments; hier ist ein End-Tag“). Während ein DOM-Parser ein komplettes Dokument einliest, benachrichtigt ein SAX-Parser eine andere Anwendung, sobald ein Teil eines Dokuments abgearbeitet ist. Was bei dem Eintritt eines Ereignisses passiert, hängt von dem Programm ab, das den Parser steuert. Dies behält die volle Verantwortung für den Bearbeitungsprozess; es braucht nur so viele Informationen wie nötig vorzuhalten, nicht aber das ganze Dokument. SAX-Parser sind daher sehr kompakt und stellen nur geringe Anforderungen an das ausführende System – das macht SAX interessant für die Verarbeitung sehr großer Datenmengen.

- Die automatische Erkennung von Dokument-Strukturen gelingt nicht mit DTDs, hierfür sind Schemata und Namensräume geeignetere XML-Technologien. Derzeit besteht. bereits eine Vielzahl von Branchen-Vokabularen, die durchaus Teilaspekte einer Problemstellung abdecken könnten. Auch wenn nur Teile davon ‚brauchbar' sind, so könnte doch auf erprobte (Teil-)Lösungen aufgebaut werden. Mittels Namensräumen (Menge von Bezeichnern) können durch die Angabe der Quelle Teile anderer Vokabeln genutzt werden, ohne Namenskonflikte durch Mehrdeutigkeiten befürchten zu müssen. Vorstellbar ist auch, in umfangreicheren Projekten für jedes Teilproblem eine geeignete Auszeichnungssprache zu entwickeln und dann Teile der verschiedenen Sprachen miteinander zu vermischen.
- Jeder Mechanismus einer robusten Anwendung muss in irgendeiner Form die Möglichkeit zur Verknüpfung von Daten vorsehen (in HTML geschieht dies über Links, in relationalen Datenbanken werden ‚foreign keys' für die Verknüpfung von Tabellen benutzt). Für die Verknüpfung von XML-Dokumenten oder anderen Inhalten als XML (Grafiken, binäre Daten) gibt es Vorschläge, die beim W3C eingereicht wurden (bspw. Xlink und Xpointer; XML Parser bieten von sich aus keine Unterstützung für das Linking an). Wünschenswert wäre es auch, Suchkriterien an einen Parser zu übergeben, der dann das Dokument nach Elementen, die diese Kritieren erfüllen, durchsucht und diese zurückliefert. Hierfür wurden ebenfalls Vorschläge eingereicht (bspw. Xquery).
- Die Transformation von XML-Dokumenten wird dann angewandt, wenn Dokumente zwischen zwei verwandten XML-Dialekten übersetzt oder aber Dokumente in ein anderes Textformat (wie die sog. ‚komma-separierten' Werte) zu transformieren sind. Das Interessante daran ist, dass die Mapping-Regeln ebenfalls in einem Dokument stehen und nicht programmiert werden. Erst zur Laufzeit wird bestimmt, welche Transformation nötigt ist – dann wird ‚einfach' das entsprechende Regelwerk angewandt. Hierfür ist XSLT (Bestandteil von XSL – eXtensible Style Language) vorgesehen; diese Skript-Sprache erlaubt viele Transformations-Operationen, das Sortieren von Daten und die Ausführung organisatorischer Aufgaben in XML ohne eine einzige Zeile einer konventionellen Programmiersprache. Stattdessen werden Regeln für die Transformation von Elementen in Abhängigkeit vom Kontext des Auftretens eines Elements formuliert.
- Anwendungen beziehen ihre Daten i.d.R. aus relationalen Datenbanken, während XML die Daten hierarchisch organisiert. Folglich wird eine Schnittstelle zwischen XML und Datenbanken benötigt. Viele Datenbankhersteller haben diesen Bedarf erkannt und bieten entsprechende Schnittstellen an. Was bleibt, ist die Entwicklung effizienter Strategien zur Überführung von Daten aus XML-Dokumenten in das Modell von relationalen Tabellen und wieder zurück (es ist bspw. ein Skript zu schreiben, mit dem Tabellen erzeugt werden können, die eigenen XML-Schemata entsprechen).

- XML-fähige Anwendungen können zu komplexen Systemen verbunden werden. Dahinter steht die Idee, verschiedene Server zur Lösung einer Aufgabe kooperieren zu lassen. Da in solchen Systemen oft die unterschiedlichsten Methoden zusammenarbeiten müssen, wird XML als Abstraktionsschicht benötigt, um die Systeme zu integrieren. Einige Methoden für ein solches Vorgehen wurden bereits entwickelt, bspw. XML-RPC, das als Konvention zur Ausführung entfernter Prozeduren auf einem Server dient. Es erlaubt den Namen der aufzurufenden Prozedur und deren Parameter zu spezifizieren; da XML als Format benutzt wird, wird von Plattform-spezifischen Problemen abstrahiert. SOAP verwendet XML, um den Zugriff auf Methoden lokaler Objekte über HTTP zu erlauben. XML dient hier der Beschreibung der aufzurufenden Methoden und der zu übergebenden Parameter. So werden Abhängigkeiten von irgendwelchen Techniken für verteilte Objekte vermieden. Beide Verfahren werden demnach von XML geschickt gebraucht, um Probleme der Kommunikation über Netzwerke zu lösen.
- XML ist ‚ideal' für den Austausch von Daten zwischen Applikationen; daher ist es nicht verwunderlich, dass der Bereich E-Commerce an der Spitze der Anwendung von XML steht und in Mitbewerb zu konventionellen Formaten (wie bspw. EDI) tritt. Der Nachteil von EDI liegt darin, dass proprietäre Netzwerke und Formate verwendet werden, was wiederum die Implementierung zeitaufwändig und teuer macht (was sich insbesonders kleinere Häuser nicht leisten können). Mittlerweile werden XML-basierende Auszeichnungssprachen entwickelt (z.B. XML-EDI), die den ursprünglichen EDI-Strukturen entsprechen, aber nun mit Werkzeugen und Parsern von Drittanbietern genutzt werden können. Das Besondere liegt hier darin, dass XML nicht nur zwischen unterschiedlichen Formaten, sondern auch zwischen physikalisch getrennten Servern vermittelt. Bei einem solchen Datenaustausch, der i.d.R. über einen Produktkatalog ‚geregelt' wird, ist nicht damit zu rechnen, dass es ‚den' Standard geben wird. Benötigt werden also Mechanismen zur Abwicklung von Geschäftstransaktionen, die auch mit proprietären Produktbeschreibungen umgehen können – mithin standardisierte Werkzeuge zur Erkennung von Datenstrukturen und zur Transformation von Daten.
- Obwohl XML seine Stärke insbesonders in der Kommunikation zwischen Anwendungen ausspielt, sind XML-Dokumente am Ende des Datenstroms dem menschlichen Leser via Browser zu präsentieren. Für die Wandlung von XML in formatierte Dokumente gibt es verschiedene, datengesteuerte Ansätze, die im Gegensatz zu dem konventionellen Vorgehen der fest eingefügten Formatisierungsanweisungen im Dokument stehen. XSL sorgt bspw. dafür, dass XML-Daten für Benutzer aufbereitet werden und dass viele verschiedene Arten der Präsentation aus ein und demselben Datensatz generiert werden können.

0.3 Ergebnis

Für die Zusammenführung der u.U. weit verstreuten Daten ist es notwendig, eine konkrete Schnittstelle zu etablieren, der ein abstraktes Datenmodell beigelegt wird. Eine solche „generische" Schnittstelle beschreibt die Strukturen der Produkt- und Geschäftsdaten in einer standardisierten, homogenen und konsistenten Form: solange für den Datenaustausch zwischen den Systemen eine Vielfalt von Schnittstellen notwendig ist, erfordert die Integration neuer Anwendungen immer aufwändigere Wartungs- und Anpassungsarbeiten. Die „generische" Schnittstelle ist hingegen offen für die Aufnahme neuer technischer Systeme bzw. fachlicher Geschäftsprozesse, ohne dass das Datenmodell strukturell geändert werden müsste.

Als zentrales Bindeglied zwischen operativen und dispositiven Systemen beinhaltet eine solche Schnittstelle Funktionen einer Middleware, die alle Geschäfte vereinheitlicht und vergleichbar abbildet. Die damit verbundene Standardisierung der Geschäftsvorfälle sowie die Existenz semantischer Schnittstellen bilden die Grundlage für ein effizientes, strategisch ausgerichtetets Schnittstellenmanagement. Es umfasst die Homogenisierung aller verfügbaren Daten mittels eines abstrakten (damit flexiblen) Datenmodells und Infrastruktureinrichtungen mit Konvertierungs- und Qualitätsfunktionalitäten.

Ein XML-basierter Datenaustausch verlangt hierfür den Entwurf einer eigenen Auszeichnungssprache, die für das Bankwesen typische Metadaten enthält. Solch eine Sprache kann u.a. als Basis für strukturierten Informationsaustausch über Transaktionen zwischen operativen und dispositiven Systemen, die Suche nach bestimmten Produktmerkmalen, die Erstellung von Report-Listen, die Abwicklung von Geschäftsprozessen, die Überwachung von Positionen dienen. Das resultierende XML-Dokument hält in jedem Fall Informationen, die an jeder Stelle des Geschäftsablaufs relevant sein können, in einem plattformunabhängigen Format bereit.

Das entscheidende Kriterium, das letzlich für XML spricht ist, dass XML eine standardisierte Metasprache ist, die es erlaubt, problemorientierte Sprachen zu erzeugen – und dennoch durch allgemeine APIs mittels Standard-Werkzeugen programmiert werden kann:

- XML bietet eine einfache Möglichkeit, Inhalte durch Tags zu beschreiben und so Informationen zu übermitteln. Tags kapseln einzelne Inhalte und erlauben so, Strukturen beliebiger Komplexität zu erzeugen
- darüber hinaus sind Mechanismen zur Erweiterung der Tags fester Bestandteil von XML. Weil dieser Mechanismus ein Standard ist, besteht eine formal festgeschriebene Methode, diese neuen Tags auch anderen Anwendungen mitzuteilen. Dies gilt auch für die Attribute von Tags. Da auch Schemata wiederum XML zur Beschreibung von Metadaten nutzen, sind die Metadaten selbstbeschreibend (i.S.e. ‚Beipackzettels')
- Obwohl XML technisch ohne weiteres dazu geeignet ist, eigene Tag-Familien für seinen eigenen Bedarf zu schaffen, liegt die eigentliche Stärke von XML darin, solche Vokabulare mit anderen Anwendern gemeinsam zu nutzen. Dies macht das Durchsuchen von Dokumenten und den Austausch von Daten wesentlich einfacher
- Unabhängig vom gewählten Wortschatz können aber stets dieselben Parser und andere Werkzeuge verwendet werden, da allen Tag-Familien eine gemeinsame Syntax zu Grunde liegt.

1 Problemstellung

Der Veränderungsgeschwindigkeit in den Kreditinstituten steigt zunehmend. So finden ständig Verbesserungen in den Bereichen Risiko-Controlling und Risiko-Management statt. Zur Zeit wird das Thema Kreditrisiko auf dasselbe intellektuelle Niveau gehoben wie das Marktpreisrisiko, um am Schluss der Diskussionen beide Risiken zu integrieren. Deutlich erkennbar ist auch der Trend, alle Risiko-Aspekte zentriert einer Abteilung verantwortlich zu übertragen – einschließlich der entsprechenden Techniklösungen. Ein solches „Enterprise Wide Risk Management" benötigt hierzu eine Vielzahl von Werkzeugen und Methoden, insbesonders nimmt der Bedarf nach Integration ständig zu, wobei eine inhaltliche Konvertierung von Daten und Nachrichten unabdingbar wird. Produktfamilien[2] für die „Enterprise Application Integration" (EAI) versprechen diese Aufgabe zu vereinfachen.

XML (eXtensible Markup Language) ist ein Format für strukturierte ‚Dokumente' und komplexe Daten. Ihr Einsatzgebiet hat sich aber in den letzten Monaten enorm erweitert, was an den zahlreichen Veröffentlichungen zu beobachten ist.[3] In der vorliegenden Studie wird XML primär in der Verwaltung von strukturierten Daten analysiert – damit wird XML selbst zum Betrachtungsobjekt, etwa bei der Frage, wie solche XML-Dokumente geeignet in einer Großrechner-Umgebung eingesetzt werden können.

Rein formal unterscheidet XML zwar nicht zwischen Dokumenten und Daten[4], dennoch ist diese Unterscheidung nicht ganz unwichtig, wenn es um die Frage der maschinellen Verarbeitung geht (die hat in einer reinen Dokumenten-Welt eine anderen Stellenwert als dort, wo (Massen-)Daten ausgetauscht werden). Nicht unwesentlich ist die Abgrenzung auch dort, wo es gilt, die Frage nach geeigneten Werkzeugen (zwecks Verarbeitung dieser Daten) zu beantworten.

Angesichts der fachlichen Problemstellung – Geschäftsdaten zwischen Applikationen auszutauschen bzw. diese entsprechend zu versorgen – wird in der vorliegenden Studie XML ausschließlich als Vehikel zum Management dieser „Nutz-Daten" gesehen. Typische „Nutz-Daten" in dem hier abzubildenden Kontext sind Daten von Finanzprodukten („Geschäftsgegenstände"), Finanztransaktionen („Geschäfte"), Finanzbeständen (z.B. an Wertpapieren) sowie Gattungs-, Markt- und Kundendaten.

(Geschäfts-)Daten treten (im Gegensatz zu Dokumenten) regelmässig auf, sind formalisiert und strukturiert (an dieser Stelle wird bereits der enge Zusammenhang zur Datenmodellierung deutlich). Für die maschinelle Verarbeitung wird die Struktur der Daten mittels sog. DTDs (Document Type Definitions) dokumentiert. Da DTDs den weitergehenden Anforderungen im Datenmanagement[5] nicht genügen, werden zu-

[2] EAI-Lösungen übernehmen die notwendige Datentransformation, verfügen über regelbasierende Module, unterstützen bei der Workflow-Steuerung und stellen Adapter/Konnektoren zur Anbindung etablierter Anwendungen bereit

[3] vgl. bspw. die Verwendung einer XML-Datei als Buildfile (in M. Marr: Fleißige Ameise, in iX 2/2001, S. 57) sowie zur Kommunikation von Komponenten über XML (A. Siffring: SOAP soll das Rückgrat von Web-Services bilden, in Computerwoche 50/2000, S. 73ff). Die Einsatzszenarien reichen demnach von der Beschreibung von Metadaten bis zum Serialisierungsformat in der Kommunikation. „Das ist das Faszinierende an XML, wie ein Chamäleon passt es sich der Einsatzumgebung an, aber das macht es auch schwer zu fassen" (N. Hranitzky: Editorial XMLspektrum, in JavaSPEKTRUM 1/2001, S. 59)

[4] ein XML-Dokument ist immer wohlgeformtes XML – egal ob es Dokumente (bspw. Bücher) oder Daten (Geschäftsdaten) enthält. XML-Dokumente sind so gesehen immer „Daten mit Auszeichnungen". Während XML zwar aus einer dokumentenorientierten Welt stammt, verschiebt sich der Schwerpunkt dennoch in die daten-orientierte Welt - i.S. einer Metasprache für die Beschreibung beliebiger Datenstrukturen

[5] ‚Datenmanagement' und ‚Schnittstellenmanagement' werden in der vorliegenden Studie als synonyme Begriffe verwendet

künftig XML-Schemata an ihrer Stelle treten (bzw. sind bereits getreten) – erlauben sie doch erst die effiziente Verarbeitung von Daten durch ihre Typisierung.

Aufgrund der Erfahrungen des Autors in konkreten Projekten scheint XML bestens geeignet als Format für komplexe Daten. Um diese Meinung zu ‚belegen', wird XML hier sowohl auf einer eher abstrakt-technischen Ebene (Beispiele hierfür sind die beiden API-Standards DOM und SAX), als auch auf einer konkret-betriebswirtschaftlichen Ebene betrachtet (wo Strukturdefinitionen, d.h. Meta-Daten von Bedeutung sind) – wobei der konkrete Ansatz stärker gewichtet wurde: Die Erfahrung zeigt, dass nicht die Syntax das Problem des Datenmanagements ausmacht, sondern die Semantik der Daten.

2 Lösungspotenzial via XML-Techniken

Mit der eXtensible Markup Language (XML) gibt es – seit 1998 in der Version 1.0 – zunächst lediglich eine abgespeckte Variante der umfangreicheren SGML - Spezifikation[6]. Formal handelt es sich mithin um eine Teilmenge von SGML unter Berücksichtigung der „Web SGML Adaptions“ vom Dezember 1997. Obwohl XML daher unmöglich mehr leisten kann als SGML, boomt die neue Sprache und lässt SGML in der Anwendungsbreite weit hinter sich zurück. Der Grund dafür ist vor allen Dingen in ihrer Rolle als universelle Schnittstellensprache zu sehen. Primär hinsichtlich dieser Rolle werden die XML-Techniken in der vorliegenden Studie denn auch betrachtet.

Nic Fulton[7] machte auf der „XML 2000-Messe“[8] mittels einer historischen Analogie deutlich, welchen Platz XML in der heutigen Welt einnimmt: Die Einführung des Standard-Containers für die Verpackung und den Transport von Gütern führte zu einer Revolution bei der Verteilung von Waren auf der ganzen Welt. So wurden Häfen mit speziellen Anlagen (sprich: proprietären Schnittstellen) überflüssig, Transportzeiten verkürzt, das allgemeine Handelsvolumen gesteigert. XML ist so gesehen, nichts anderes als ‚der’ Standardcontainer mit der Ware Information – und der Versuch, das Datenchaos zu managen, das durch die Verknüpfung der zahlreichen hochspezialisierten Systeme über die Jahre hinweg entstanden ist.

Datenmanagement bedeutet in diesem Zusammenhang die Implementierung eines ‚daten-zentrierten-Ansatzes’, der auf die Geschäftsprozesse ausgerichtet ist. „Datastream management is the process of taking data from the diverse, heterogeneous business systems and platforms found in today’s enterprise, and combining and enhancing them to create valuable, intelligent workstreams that any intended destination can use.“[9] Das so beschriebene Management von Datenströmen beinhaltet den Datentransfer, die Datentransformation und die Datenverteilung.

2.1 Management von Datenströmen

In Kreditinstituten besteht der Wunsch nach einem geordneten, standardisierten Datenaustauschverfahren. Ein solches Verfahren umfasst Techniken des Transfers, der Transformation und der Kommunikation. XML-Techniken dienen hierbei als ‚semantische Middleware[10]’, in der geschäftsprozessbezogene Meta-Daten zwischen den Nutz-Daten der Quell- und Zielsysteme vermitteln:

[6] das WordWideWebConsortium (W3C) hat im Februar 1998 den St

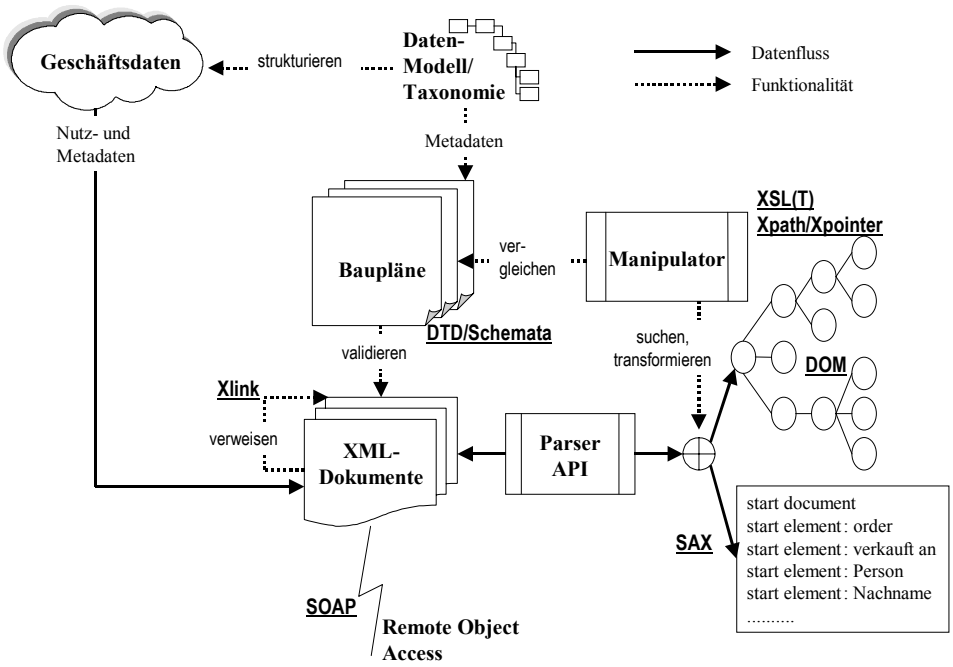


ABBILDUNG 1: XML-TECHNOLOGIEN IM DATENFLUS

Die obige Abbildung ordnet die wesentlichen „XML-Technologien“ ihrer Bestimmung im Management eines Datenstroms zu. Die genannten Standards werden nachfolgend skizziert.

2.1.1 Datentransfer

I.d.R. definieren Datenstandards Formate, in denen Daten in einer Datei gespeichert werden. XML erlaubt hingegen dem Anwender die Beschreibung seiner Daten in einer den „Nutz-Daten“ entsprechenden Form („Meta-Daten“, die durch sog. „Tags“ als solche gekennzeichnet wurden) beizulegen. Nutz- und Metadaten werden zusammen in einer Datei gespeichert; sie werden als „XML-Dokumente“ bezeichnet.

Die Tags einer XML-Sprache[11] sowie deren hierarchische Struktur und deren Attribute sind in einer speziellen Datei, der sog. „Document-Type-Definition“[12] (DTD) festgelegt. Deren Syntax ähnelt der von XML, da auch hier wieder Tags benutzt werden. Die Tags können mittels verschiedener Mechanismen (s.u.) bearbeitet werden. Während HTML immer mit demselben Satz an Tags auskommen muss, um Dokumente auszuzeichnen, erlaubt XML dagegen beliebig viele solcher Bezeichner zu verwenden.[13] Ziel ist es letztlich, alle relevanten Daten und ihre Eigenschaften aus der Realwelt über geeignete semantische Abbildungen auf ein mit den XML-Notationsregeln konformes Norm-Dokument zurückzuführen:

[11] die Folge von Zeichen zwischen den Elementbegrenzern ‚<’ und ‚>’ werden als Tags bezeichnet und bestimmen die semantische Bedeutung der Daten
[12] solche Normierungen legen die Struktur von XML-Dokumenten und die Verarbeitungsregeln der Nutz-Daten verbindlich fest
[13] insofern wird XML auch als Metasprache bezeichnet, die lediglich eine abstrakte Grammatik spezifiziert, mit der der Anwender eigene „Markup Languages“ bauen kann

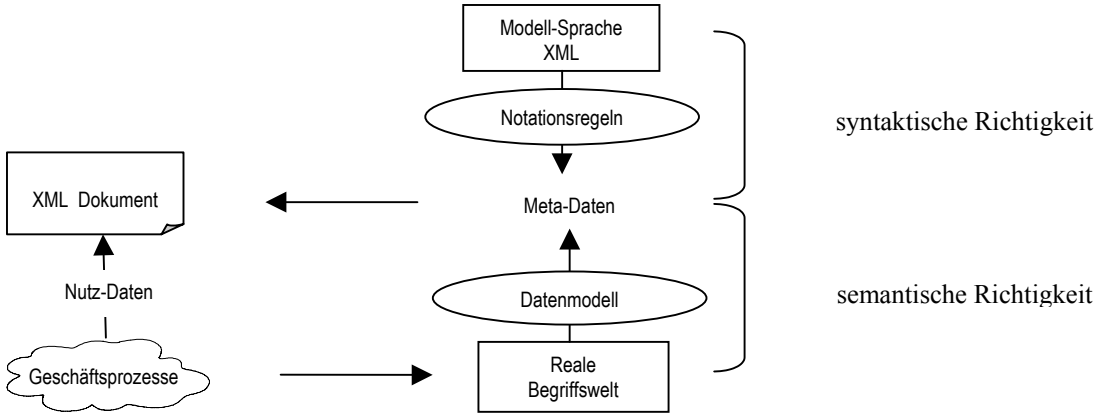


ABBILDUNG 2: REALITÄT, MODELL, XML-DOKUMENT

Da beliebige Tag-Sets definiert werden können, ist es nicht verwunderlich, dass für die Spezifikation von Datenaustausch-Schnittstellen von verschiedenen Initiativen bereits „einheitliche Standards“ festgelegt wurden.[14] Für den hier interessierenden Finanzbereich sind folgende Modellierungs-Standards von Bedeutung:[15]

- **FpML** [www.fpml.org] (Financial Products Markup Language; Initiatoren: J.P.Morgan und PricewaterhouseCoopers), welche auf Finanzderivate[16] (derzeit Zins- und Devisenprodukte) einschließlich Portfolio- und Risiko-Management zielt. J.P.Morgan will mit FpML die Geschäftsprozesse „Abschluss“, „Bestätigung“, „Bewertung“, „Risiko-Analyse“ und „Zugriff auf Marktdaten“ abbilden, während PricewaterhouseCoopers mit FpML das Management für Markt- und Betriebsrisiken outsourcen will.
- **FinXML** [www.finxml.org] (Finance eXtensible Markup Language; Initiator: INTEGRAL), das sich (ebenfalls) als Standard-Austauschformat im Finanzmarkt etablieren soll. Die gegenwärtigen Definitionen beschreiben u.a. Elemente und Attribute für Transaktionen, Finanzinstrumente, Marktdaten, Zahlungen, Settlements und Bestätigungen. Unterstützt werden zahlreiche Nicht-Derivate und Derivate: Zins-, Devisen-, Edelmetall-Derivate, Bonds, Geldmarkt-Produkte, Kredite, Einlagen, börsengehandelte Optionen und Futures. FinXMLs Spezifikation wurde entwickelt in Zusammenarbeit mit der ISDA (International Swaps and Derivatives Association), deren Standards im Markt weit verbreitet sind.
- **NTM** [www.risk.sungard.com/ntm] (XML-basiertes „Network Trade Model“ (NTM); Initiator: SunGard) wurde von SunGard als Schlüssel-Komponente für ihre Integration Strategie eingeführt. Um die Gefahr einer Fragmentierung mit XML zu verhindern, ist SunGard seit Juli 2000 im „technischen Kommittee“ von FpML vertreten: SunGard sieht NTM komplementär zu FpML und plant, FpML und NTM zu gegebener Zeit zusammenzuführen.[17] Auf Grund der Implementierungsreife/-umfangs wird NTM im Folgenden detaillierter beschrieben:

[14] diese ‚Standards‘ legen Definitionen innerhalb spezifischer Fachgebiete fest. Sie werden technisch zur Zeit (noch) in entsprechenden DTDs abgelegt, wo die Struktur und die Syntax der tags (und damit die definierte Sprache mit den erlaubten Schachtelungen, die Tag-Attritute und die erlaubten Attributwerte) deklariert werden. Beispielsweise ebXML (electronic business XML): es geht hier bei um einen technischen Rahmen zum weltweiten Austausch von elektronischer Geschäftsdaten

[15] in dieser Vielfalt liegt zugleich das (temporäres) Dilemma: ohne Unterstützung von ‚großen’ Softwareanbietern bleibt das Integrationsproblem letztlich bestehen; vgl. auch St. Folan: Avoiding Babel, in Risk Professional 2/2001, S. 25f

[16] Weather Risk Advisory (Cambridge, England) führt eine Initiative an, die ein Standardprotokoll für Wetterderivate, basierend auf XML, plant; siehe www.weatherml.org

[17] vgl. o.Verf.: The SunGard Network Trade Model, o.J.

NTM basiert im Kern auf FpML wurde aber erweitert in Richtung[18]

- eigener XML-Baupläne (DTDs/Schemas) für neue Produkte, Prozesse und Referenz-Daten
- externer XML-Baupläne (mittels eines Konverters), um die Kompatibilität zwischen NTM Dokumente und externen Standards wie FpML herzustellen bzw. beizubehalten.

Sog. ‚NTM Adapter' wurden für eine Vielzahl von SunGard-Systemen und Middleware-Produkte entwickelt. Darüber hinaus wird das NTM Modell selbst und werden Werkzeuge für den Datenaustausch mitgeliefert, um auch non-SunGard-Systeme zu integrieren:[19]

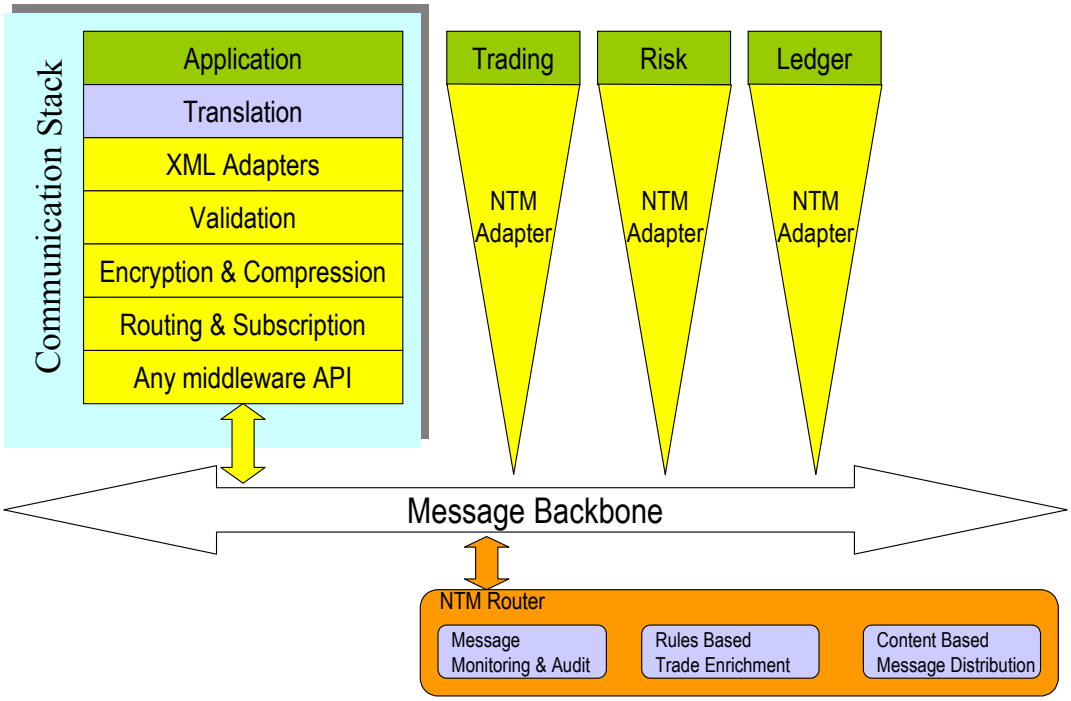


ABBILDUNG 3: NTM - COMMUNICATION STACK

Quelle: in Anlehnung an: o.Verf.: The SunGard Network Trade Model, o.J. S. 6/7

NTM Definitionen erlauben die Abbildung zahlreicher Handelsaktivitäten. Die derzeit verfügbaren Produkte werden ständig erweitert. Jede Definition lässt darüber hinaus (mittels Kombination bestehender Beschreibungen), auch die Abbildung strukturierter Produkte zu. Die 'NTM Trade Definitions' umfassen derzeit (Mai 2001) u.a.:

[18] vgl. bspw. SunGard's middleware: www.integration.sungard.com
[19] o.Verf.: The SunGard Networt Trade Model, o.J., S. 6f

1. NTM General Model Outline
    1. The Master Message: <NTMMessage>
    2. The Message Header: <NTM Header>
    3. The Master Trade Message: <NTM Trade>
    4. The Trade Header: <TradeHeader>
    5. Extensibility: <Extensions>
    6. Trade Tags: <TradeTags>
    7. Transaction Costs: <TradeTransactionCosts>
    8. Future Agreements: <FutureInfo>
    9. Options on underlying trades: <OptionInfo>
    10. Cashflow Strips: <TradeLeg>
        1. Trade Leg Schematic
        2. Cashflow Generation Details: <TradeLegDetails>
        3. Cashflows: <Cashflows>
        4. Cashflow: <Cashflow>
        5. Cashflows Example: Fixed Interest
        6. Cashflows Example: Vanilla Floating Interest
        7. Cashflows Example: Compounding and Averaging Floating Interest
2. Trades
    1. Simple Trades
        1. FX: <FX>
        2. FX Swaps: <FXSwaps>
        3. FX Options: <FXOption>
        4. FX Futures: <FXFuture>
        5. Deposit Futures: <DepositFuture>
        6. Options on Deposit Futures: <DepositFutureOption>
        7. Discount Bills: <DiscountBill>
        8. Forward Rate Agreements: <FRA>
        9. Payments: <Payment>
    2. Securities
        1. Bonds: <Bonds>
        2. Options on Bonds: <BondOption>
        3. Bond Repo: <BondRepo>
        4. Certificate Of Deposit: <CertificateOfDeposit>
        5. Floating Rate Note: <FRN>
        6. Bond Futures
        7. Options on Bond Futures: <BondFutureOption>
    3. Equity and Derivatives
        1. Equity: <Equitiy>
        2. Options on Equity: <EquityOption>
        3. Equity Futures: <EquityFuture>
        4. Options on Equity Futures: <EquityFutureOptions>
        5. Equity Repo
    4. Commodity Derivatives
        1. Commodity Futures: <CommodityFuture>
        2. Options on Commodity Futures: <CommodityFutureOption>
        3. Commodity Forwards: <CommodityForward>
        4. Options on Commodity Forwards: <CommodityForwardOption>
    5. Interest Rate Derivatives
        1. Loans and Deposits: <Cash>
        2. Caps, Collars and Floors
        3. Interest Rate Swaps: <IRSwap>
        4. Options on Interest Rate Swaps: <IRSwaption>
3. DTD Style Guide
4. Data Dictionary

ABBILDUNG 4: NTM - TRADE DEFINITIONS

Quelle: SunGard (Hrsg.): NTM Trade Definitions User Guide, o.O., Mai 2001, S. 3f

NTM ist letztlich eine Modell-Sammlung, beschrieben durch DTDs (Schemas werden voll unterstützt); auch die Messages sind in XML beschrieben. Die NTM DTDs basieren auf normalisierte Datenstrukturen; die NTM Messages beginnen mit einem NTMHeader, die Geschäftsinformationen beginnen mit einem NTMTradeHeader. Gemeinsame Datenstrukturen (wie „cashflows“) werden von allen Geschäftstypen benutzt.

Zur Illustration von XML-Dokument, DTD und XML-Schema wird nachfolgend das einfache Dokument „Visitenkarte“ genutzt:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <!DOCTYPE Visitenkarte (View Source for full doctype...)>
- <Visitenkarte>
   - <Name>
       <Vor_Name>Helmut</Vor_Name>
       <Nach_Name>Siegert</Nach_Name>
     </Name>
   - <Firma>
       <Firma_Name>Dr. Siegert & Partner</Firma_Name>
       <Position>Geschäftsführer</Position>
     </Firma>
   - <Kontakt>
       <email>Helmut.Siegert@Siegert-Partner.de</email>
       <phone type="direkt">06027-465702</phone>
       <phone type="Zentrale">06027-465700</phone>
     </Kontakt>
 </Visitenkarte>
```

ABBILDUNG 5: XML-DOKUMENT „VISISTENKARTE“

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XML Spy v3.0 (http://www.xmlspy.com)-->
<!ELEMENT Firma (Firma_Name, Position)>
<!ELEMENT Firma_Name (#PCDATA)>
<!ELEMENT Kontakt (email, phone+)>
<!ELEMENT Nach_Name (#PCDATA)>
<!ELEMENT Name (Vor_Name, Nach_Name)>
<!ELEMENT Position (#PCDATA)>
<!ELEMENT Visitenkarte (Name, Firma, Kontakt)>
<!ELEMENT Vor_Name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ATTLIST phone type (Zentrale | direkt) #REQUIRED>
```

ABBILDUNG 6: XML-DTD „VISITENKARTE“

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.0 (http://www.xmlspy.com) by Helmut Siegert (Dr. Siegert & Partner) -->
<!--W3C Schema generated by XML Spy v3.0 (http://www.xmlspy.com)-->
<!DOCTYPE xsd:schema PUBLIC "-//W3C//DTD XMLSCHEMA 19991216//EN" "" [
    <!ENTITY % p 'xsd:'>
    <!ENTITY % s ':xsd'>
]>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <xsd:complexType name="FirmaType" content="elementOnly">
        <xsd:sequence>
            <xsd:element name="Firma_Name" type="xsd:string"/>
            <xsd:element name="Position" type="xsd:QName"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="KontaktType" content="elementOnly">
        <xsd:sequence>
            <xsd:element name="email" type="xsd:string"/>
            <xsd:element name="phone" type="phoneType" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
```

```
    </xsd:complexType>
    <xsd:complexType name="NameType" content="elementOnly">
        <xsd:sequence>
            <xsd:element name="Vor_Name" type="xsd:uriReference"/>
            <xsd:element name="Nach_Name" type="xsd:uriReference"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="Visitenkarte">
        <xsd:complexType content="elementOnly">
            <xsd:sequence>
                <xsd:element name="Name" type="NameType"/>
                <xsd:element name="Firma" type="FirmaType"/>
                <xsd:element name="Kontakt" type="KontaktType"/>
            </xsd:sequence>
            <xsd:attribute name="xmlns:xsi" type="xsd:uriReference" use="default" va-
lue="http://www.w3.org/1999/XMLSchema-instance"/>
            <xsd:attribute name="xsi:noNamespaceSchemaLocation" type="xsd:string"/>
            <xsd:attribute name="xsi:schemaLocation" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="phoneType" base="xsd:string">
        <xsd:enumeration value="06027-465700"/>
        <xsd:enumeration value="06027-465702"/>
        <xsd:attribute name="type" use="required">
            <xsd:simpleType base="xsd:NMTOKEN">
                <xsd:enumeration value="Zentrale"/>
                <xsd:enumeration value="direkt"/>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
</xsd:schema>
```

ABBILDUNG 7: XML-SCHEMA „VISITENKARTE“

An diesem simplen Beispiel wird bereits deutlich:

- Tags definieren (nur) das Beginn und Ende eines Elements; der Inhalt eines Elementes steht zwischen den Tags
- die Elemente erscheinen in einer strengen Hierarchie
- jedes Element kann ein oder mehrere Attribute haben (siehe z.B. `type="direkt"`)
- neue Elemente werden über neue Tags identifiziert: Was sich semantisch hinter den Tags verbirgt, definiert der Anwender. Anstelle der „sprechenden“ Bezeichnung <Vor_Name> könnte auch <VN> verwendet werden, ohne dass ein XML-Parser[20] dies beanstanden würde.[21] Durch die Verwendung ‚kryptischer‘ Steuerzeichen geht jedoch ein wichtiger Vorzug von XML verloren, nämlich „verstehbare“ Tags (wie sie auch strukturiert in Datenbanken abgelegt sind) und damit lesbare Dokumente abzufassen.

Unter Umständen ist es notwendig, ein komplexes Dokument durch *mehrere* Tag-Familien zu strukturieren[22]. Das setzt zunächst voraus, dass die Tag-Familien keine „Namensvetter“ haben. Mit sog. „Namespaces“ ist aber auch ein solches Problem

[20] der Parser untersucht die Verwendung von Elementen und Attributen in einem XML-Dokument mit den Regeln, die im Norm-Dokument festgelegt sind. Geprüft wird nur die syntaktische Korrektheit – die Bedeutung der Daten bzw. des Dokumentes zu interpretieren, ist Aufgabe der zu verarbeitenden Anwendung (sobald aber ein XML-Schema vom Browser angewendet wird, wird auch der (ggfs. sehr detaillierte) Datentyp vom Parser geprüft). Parser sind keine speziellen Werkzeuge, vielmehr können sie für alle XML-Dokumente verwendet werden

[21] Parser rufen beim Einlesen des Dokuments an vorher vereinbarten Stellen (Start-Tag, End-Tag, etc.) Skript-Prozeduren auf, denen die entsprechenden Dokumentteile übergeben werden. Mithin liegt „nur“ in der Programmierung der aufgerufenen Prozeduren die eigentliche Arbeit

[22] neben ‚freien‘ gibt es auch ‚feste‘ Bezeichner aus anderen W3C-Standards wie HTML und ‚offiziellen‘ DTDs wie FinXML und FpML

lösbar, indem jede Familie ihre eigenen Tags unter einem „Familiennamen“ zusammenfasst. Mittels eines Prefix kann ein Parser deren Herkunft feststellen.

Namespaces erlauben, mehrere (standardisierte) Tag-Familien, die sich auf bestimmte Domänen spezialisiert haben, in einem XML-Dokument parallel zu berücksichtigen. Darüber hinaus kann ein eigener Standard entwickelt werden, der ggfs. auf die genannten aufsetzt und um individuelle Elemente modifiziert wird. Darüber hinaus kann XSL(T) kann hierbei als ‚Translator’ zwischen verschiedenen ‚Standards’ fungieren. Dort wo es möglich ist, sollten desweiteren etablierte Codes (z.B. Länder- und Währungscodes des ISO-Standards, WM-Gattung, etc.) genutzt werden.[23]

In XML unterscheidet man zwischen wohlgeformten (well-formed) und gültigen (valid) Dokumenten: sie sind dann „wohlgeformt“, wenn ein XML-Dokument in sich schlüssig ist, d.h. alle Elemente sind korrekt geschachtelt und in XML-Syntax vorhanden (syntaktische Richtigkeit). Bei „Gültigkeit“ liegt zusätzlich zum Wohlgeformten eine Strukturierung des XML-Dokumentes analog zur Struktur einer DTD vor (semantische Richtigkeit).[24]

Für den automatischen Datenaustausch zwischen Applikationen stößt die DTD (die bislang den einzigen standardisierten Mechanismus für Meta-Daten darstellt) an Grenzen: Ein besonderer Mangel für den kontrollierten Datentransfer ist, dass in DTDs Elementen oder Attributen keine Datentypen zugeordnet werden können. So kann bspw. das XML-Element „phone“ an der Quelle eine Zeichenkette (string) auszeichnen, am Ziel wird aber ein numerischer Wert erwartet. Eine Validierung gegen eine DTD könnte den unerwünschten Versand von Zeichenketten nicht verhindern.

Diese Schwächen sollen mit XML-Schemata beseitigt werden[25]. Auch sie legen fest, wie Dokumente zu strukturieren sind, die sich als Instanz dieses Dokumenttyps sehen. Die Beschreibung des Vokabulars und seiner Verwendungsregeln erfolgt in Schemata-Dateien (mittels XML-Syntax); die Schemaspezifikation legt eine Vielzahl von Datentypen fest und erlaubt deren eindeutige Zuordnung zu Elementen.

XML Schemata erlangte Ende Oktober 2000 den Status einer „Candidate Recommendation“, die Verabschiedung als W3C „Proposed Recommendation“ erfolgte im März 2001.[26]

2.1.2 Datentransformation

Während es bisher „nur“ darum ging, Daten auszugeben, zu transportieren und einzulesen, kann eine Veränderung der Daten zwischen Ausgeben und Einlesen bspw. dann notwendig werden, wenn als Schnittstelle ein anderes als das intern verwendete XML-Format vereinbart wurde. Die Transformation lässt sich zwar durch eigene Programmierung regeln, da Transformationen aber häufig benötigt werden, gibt es dafür eine massgeschneiderte Sprache, die „eXtensible Stylesheet Language Transformation“ (XSLT).

[23] an dieser Stelle wird bereits deutlich, dass der Entwurf maßgeschneiderter „Bauvorschriften“ die Arbeit von Fachexperten (’Content’) und Software-Entwicklern (’Architecture’) ist: „XML technology still requires the disciplines of analysis and development of a set of requirements to produce the best result“ (St. Folan: Avoiding Babel, in Risk Professional, 2/2001, S. 26)
[24] das Norm-Dokument kann folglich auch als Vertrag zwischen dem Lieferanten und dem Abnehmer der Information hinsichtlich der Einhaltung auf Syntax und Semantik der Schnittstellendefinition gesehen werden (siehe hierzu Kapitel 4)
[25] vgl. o.Verf.: Schemata machen XML für den Datenaustausch tauglich, in Computerwoche 47/2000, S. 17f
[26] vgl. hinsichtlich konkurrierender Ansätze o.Verf.: Bei XML-Schemasprachen ist kein Standard in Sicht, in Computerwoche 13/2001, S. 24

Neben XML-Dokumenten und DTDs/Schemata gehören daher „eXtensible Stylesheet Languages“ (XSLs) zum XML-Formalismus. Während DTDs/Schemata Daten strukturieren, formatiert XSL die Daten zur Darstellung in unterschiedlichen Medien oder zur Weiterverarbeitung in unterschiedlichen Systemen – unter Berücksichtigung der DTDs/Schemata.[27] Hierzu liest ein Parser die Datei ein, durchsucht sie schrittweise nach DTDs/Schemata-Notationen und erstellt einen Baum mit den Meta-Daten als Knoten. Der Baum wiederum wird von einem Codegenerator durchkämmt, der das gewünschte Ausgabeformat (DOC, PDF, HTML, PS, WAP, etc.) erzeugt.

Es gibt zwei standardisierte Verfahren, um auf diese Daten per API (Application Programming Interface) zuzugreifen: Da ist zum einen das „Simple API for XML (SAX)“[28], zum anderen das „Document Object Model (DOM)“ des Web-Consortiums W3C.[29] In der *DOM*-Spezifikation sind eine Reihe von Methoden spezifiziert, mit denen sich die Baumstruktur des Dokuments auslesen und verändern lässt, wobei Token die einzelnen Knotenpunkte referenzieren. Auf diese Token können Entwickler die entsprechenden Methoden anwenden, z.B. Attribute lesen oder setzen, Kind-Elemente abfragen und erzeugen. Aufgrund dieser Fähigkeiten resultiert der große Einsatzbereich von DOM, der weit über die Dokumentenverarbeitung hinausgeht. Mit Hilfe dieses Modells lassen sich beliebig komplizierte Baumstrukturen erstellen und zur Speicherung oder Übermittlung serialisieren.

Das DOM erfordert, dass ein eingelesener Datensatz zuerst komplett analysiert wird, bevor er in einer Baumstruktur zur Verfügung steht. Für den Entwickler ist es dann relativ leicht, Transformationen mittels XSLT-Prozessoren[30] an diesem Baum vorzunehmen, d.h. Knoten zu entfernen oder hinzuzufügen, um den Datensatz anschließend als transformiertes XML-Dokument wieder auszugeben.[31] Diese Vorgehensweise ist allerdings mit einem großen Speicherbedarf verbunden, da die Anwendung das Dokument vollständig im Speicher halten muss.

Im Vergleich zu DOM geht *SAX* deutlich sparsamer mit dem Speicherplatz um. Bei diesem ereignisorientierten API werden während der Analyse des Datensatzes Callback-Methoden aufgerufen. So führt bspw. das Auftreten von Start- und End-Tags zum Aufruf der Methoden startElement() und endElement(). Die zwischen den Tags stehenden Zeichen (der Inhalt des Elements) verarbeitet die Methode characters().

SAX ist zwar leicht benutzbar, ist aber mit gewissen Einschränkungen verbunden. So kann man bspw. nicht vorausschauen um festzustellen, ob das aktuelle Element nachfolgende Geschwister hat. Die Ursache liegt in der sequenziellen Verarbeitung der Daten: SAX stellt durch das sequenzielle Einlesen keine Datenstruktur zur Verfügung, durch die das Dokument weiter verarbeitbar wäre, während der DOM-Baum sich hingegen beliebig durchlaufen lässt – weil er nach dem Parsen eine komplette Datenstruktur aufbaut, die sich leicht traversieren läßt.

---

[27] insgesamt gliedert sich XSL in 3 Teile: Transformatoren, Selektoren und Gestaltungsaspekte. Vgl. hierzu: A.Oberdörster: XML mit Serviervorschlag, in c’t 6/2000, S. 244ff sowie ders.: XSL im Detail, in c’t 6/2000, S. 250ff

[28] nähere Informationen hierzu unter http://www.megginson.com/SAX/

[29] siehe http://www.w3.org/DOM/

[30] es gibt zahlreiche Open-Source-Software, problematisch erweist sich hier allerdings die mangelnde Ausführungsgeschwindigkeit. Sun hat einen XSLT Compiler, der XSLT-Anweisungen in Java-Programmcode überführt, der dann seinerseits compiliert und ausgeführt werden kann – was insgesamt die Verarbeitungsgeschwindigkeit erhöht.

[31] hierfür definiert DOM spezielle Schnittstellen

SAX eignet sich gut, wenn man lineare Konvertierungen vorzunehmen hat, d.h. jedes Tag genau einem Output zuzuordnen ist. Entdeckt der Parser einen Knoten, so ruft er eine dafür zuständige Callback-Routine auf, die die Konversion durchführt oder eine Aktion auslöst. Sie kann entweder einfachen HTML-Code erzeugen oder Platzhalter, die durch Informationen ersetzt werden, z.B. eingelesene Daten werden so in eine Datenbank geschrieben. Im Gegensatz zur Baumstruktur des DOM liegt hier ein linearer Daten-Strom vor.

Bei der Transformation innerhalb der Grenzen von XML kommt XSLT eine große Bedeutung zu. Mit den verfügbaren Standards (XML 1.0, DTDs oder XML-Schemata, XSL(T)) besitzt man folglich Spezifikationen, um Daten strukturiert zu beschreiben, auf sie zuzugreifen, sie zu lesen[32] und zu verarbeiten. Für die Datenkommunikation über Rechnergrenzen hinweg ist darüber hinaus ihr Transport notwendig.

2.1.3 Datenverteilung

Neben der Verarbeitung von Dokumenten dient XML auch zur Kommunikation zwischen Prozessen. Vorhandene Transport-Protokolle wie HTTP (Hyper Text Transfer Protocol, synchron) oder SMTP (Simple Mail Transfer Protocol, asynchron) lassen sich ohne weiteres dazu nutzen. Im ersten Fall können Daten durch einen URL als Adressangabe nach dem Muster „http://host.domain.com/somepath.xml“ mit einem HTTP-GET angefordert oder per HTTP-POST versendet werden. Durch Angabe eines MIME-Type im HTTP-Head lässt sich der Datensatz als XML identifizieren.

Die Benutzung von Standards führt allerdings noch nicht zwangsläufig zu standardisiertem Vorgehen. Der Einsatz von HTTP als Transportprotokoll und XML zur Beschreibung der Pakete sagt noch nichts darüber aus, *wie* die Pakete aussehen. Mit dem XML-basierten Kommunikationsprotokoll „Simple Object Access Protocol“ (SOAP) gibt es einen entsprechenden – von Microsoft initiierten – Standard: sowohl die gemeinsam von OASIS (Organization for the Advancement of Structured Information Standards) und der UNO getragene ebXML-Initiative als auch das konkurrierende von Microsoft initiierte Biztalk-Konsortium werden SOAP 1.1. nutzen.

SOAP[33] stellt im Wesentlichen ein auf XML basierendes Format dar, das eine Serialisierung von Methodenaufrufen sowie deren Ergebniswerten und Fehlermeldungen darstellt. Die Übertragung kann wiederum synchron per HTTP oder asynchron per SMTP erfolgen. SOAP ist aber kein verteiltes Objektmodell (wie Corba oder COM), vielmehr wurde es unabhängig vom Objektmodell und der Programmiersprache für Methodenaufrufe über das Web konzipiert.

Der Aufruf einer Methode geschieht in der für die jeweilige Programmiersprache üblichen Weise: Noch auf der Client-Seite wird der Methodenaufruf für den Entwickler transparent in XML-Form in einen „soap:Body“ eingebaut, der Teil eines „soap:Envelope“ ist. Dieses Gesamtpaket nimmt via HTTP (POST-Methode) oder SMTP seinen Weg zum Server. Dort läuft der umgekehrte Vorgang ab: Der Methodenaufruf wird deserialisiert und die entsprechende Methode aufgerufen. Hierbei wird entweder ein Rückgabewert oder ein Fehler erzeugt. Auf jeden Fall schickt der Server das Ergebnis in ein SOAP-Antwortpaket verpackt an den Client. Die SOAP-

[32] das Schreiben sollte mit DOM erfolgen, weil sich SAX nicht zum Schreiben von XML-Dateien eignet
[33] vgl. o.Verf.: SOAP soll das Rückgrat von Web-Services bilden, in Computerwoche 50/2000, S. 73f sowie anwendungsbezogen M.Farkus/R.Milward: Working up a lather over SOAP, in RISK 10/2000, S. 42f

Komponente öffnet den „soap:Envelope“ und reicht das Ergebnis als Resultat des ursprünglichen Methodenaufrufs weiter.[34]

Die Einfachheit von Kommunikationsabläufen wie diesem trägt dazu bei, dass die Bedeutung von XML für die Datenkommunikation stetig zunimmt – insbesondere bei der Integration verteilter Anwendungen. XML hat daher in vielen Bereichen der Softwareentwicklung Einzug gehalten; auch im Bereich der komponentenbasierten Middleware spielt XML eine Schlüsseltechnologie mit hohem Potenzial.[35] Interessant ist in diesem Zusammenhang der Versuch, die Strenge in der Typisierung einer Schnittstelle durch die Strenge in der Einigung auf einen Parameternamen (Tag) zu ersetzen (sog. ‚Strong Tagging’)[36]. Der Zugriff erfolgt dann bspw. per DOM – und ist damit plattformunabhängig. Solche Schnittstellen scheinen erheblich robuster als ihre streng typisierten Vettern zu sein, weil die Adressierung der Prozedur über einen lesbaren Namen erfolgt – und davon auszugehen ist, dass sich Parameternamen sehr viel seltener ändern als Typ oder Position eines Parameters. Darüber hinaus besteht die Möglichkeit, eine plattformübergreifende Lösung für verteilte Rechner zu nutzen.[37]

Das ‚Besondere’ an XML liegt insgesamt im hohem Abstraktionsvermögen als Beschreibungssprache. Dadurch kann sie an verschiedenste Randbedingungen angepasst werden und ist nicht an konkrete Technologien/Geschäftsprozesse gebunden. Aber auch die Anwendung von XML braucht Standards – zumindest wäre es von Vorteil, wenn man sich an vorhandenen Standards orientieren könnte.

2.2 Standardisierte Produktkataloge

Im Zentrum der anwendungsbezogenen Standardisierungsdiskussion stehen DTDs bzw. Schemata, die letztendlich auf Schnittstellen abgebildet werden. Als Belege für die bereits weit fortgeschrittenen Bemühungen um die Entwicklung von produktdefinierenden Standards seitens der Finanzindustrie sei auf http://www.xml.org/xmlorg_registry/index.shtml hingewiesen.

Bislang fehl(t)en Standards zur Beschreibung von Finanzprodukten, was zu erheblichen Transaktionskosten hinsichtlich der Integration von Anwendungen führt(e). Benötigt werden Produktbeschreibungen, die direkt als Input bspw. für Pricing-Modelle, VaR-Berechnungen und Back-Office-Bewertungen verwendet werden können. Solche „Produktkataloge“ sind Klassifikationssysteme, die hierarchisch aufgebaut sind. Sie ordnen Produkte unterschiedlich tief gegliedert entsprechend ihren Eigenschaften. Die Eigenschaften können über Entitäten und Attribute abgebildet werden, die für eine vollständige Klassifizierung unverzichtbar sind. Für jede Produktklasse wird ein typischer Set von Eigenschaften (und ggfs. die zulässigen Werte) definiert.

[34] vgl. auch C.F.Vasters: SOAP – die Lösung aller Interoperabilitätsprobleme?, in OBJEKTspektrum 6/2000, S. 26ff. Sofern der Datenaustausch zwischen Systemen im Mittelpunkt einer Architektur steht, empfiehlt VASTERS (statt aufwändige Middlewaresysteme) den Einsatz von SOAP (a.a.O., S. 28)

[35] zum Einsatz von XML in der komponentenbasierten Softwareentwicklung vgl. A.Jung: XML – Schlüsseltechnologie für Softwarearchitekturen, OBJEKTspektrum 1/2001, S. 71ff. Der Autor empfiehlt, XML als Datenbus einzusetzen, in dem von den Quell- bis hin zu den Zielsystemen durchgängig mit XML-Dokumenten gearbeitet wird. Die Parameterlisten der funktionalen Schnittstellen reduzieren sich im Wesentlichen auf die Übergabe von XML-Dokumenten.

[36] vgl. R.Westphal: Strong Tagging als Ausweg aus der Interface-Versionshölle, in OBJECTspektrum 4/2000, S. 60ff. Auch SOAP hat vordringlich eine enge Kopplung von Komponenten im Sinn, wenn auch plattformübergreifend und unabhängig von konkreten Technologien wie bspw. CORBA

[37] so werden in XML-RPC (Remote-Procedure-Call) die Parameter zum Aufruf von Prozeduren, Rückgaben und Fehlermeldungen in XML ausgedrückt

Eine solche Ausdifferenzierung und Definition von Produktmerkmalen ist eine aufwändige Aufgabe, die an „Fach-Experten“ zu delegieren ist, die die (Bank-) Geschäfte kennen, verstehen und abstrakt sowie formal (z.B. mittels eines „entity-relationship-modells“) abbilden können. Ziel muss es sein, die Geschäftssemantiken (zentral) zu dokumentieren – da Daten und ihre Verwendung die Kern-Kompetenz eines Kreditinstituts widerspiegeln (gleichzeitig bilden sie die Basis für die Informationsverarbeitung). „Surely maintainable business semantics – and the enterprise integrity that results from using a proven technique – is worth a little formal design effort involving application vendors and developers“.[38]

Es ist zu vermuten, dass sich viele XML-basierte Produktkataloge auf dem Weg zur „Standardisierung“ befinden. Im Finanzsektor sind bislang keine entscheidenen Fortschritte zu sehen. Bei Einigung auf einen Produktkatalog ist darauf zu achten, dass ein solcher Standard nur dann breite Akzeptanz finden wird, wenn

- die Bedürfnisse der potenziellen Nutzer in der Entwicklung berücksichtigt werden,
- der Standard offen ist und
- ohne Lizenzkosten

integriert werden kann.

Diese Problematik zeigt sich derzeit bspw. bei FpML, als „Standard Protokoll“ für die Datenverarbeitung von Swaps, FRAs, Cap/Floors und Swaptions: ‚proprietäre‘ Weiterentwicklungen dieses Produktkatalogs könnten dann in eine Sackgasse münden, wenn nicht geklärt wird, wie die Harmonisierung zukünftig aussieht. So wäre es beispielsweise denkbar, über den Weg von ‚Namespaces’ Core- und Extension-Modell zu integrieren. Darüber hinaus ist aber auch die Frage der Lizensierung noch nicht geklärt.[39]

Ein wesentlicher Charakter von XML-Dokumenten, selbstbeschreibende Informationen via Norm-Dokumente beizulegen, erlaubt es aber auch, bei der Entwicklung neuer Anwendungen unabhängig voneinander zu arbeiten, um nicht auf die Entwicklung von ‚großen‘ Firmen oder ‚abstimmungspflichtigen‘ Kommittees abhängig zu sein.

Praktisch alle Anbieter von „Marktplatzsystemen“ im Nicht-Finanzbereich setzen beim Austausch von digitalen Produktkatalogen auf XML-basierte Formate, allerdings ist ein Standardaustauschformat für den digitalen Handel (auch) dort noch nicht in Sicht.[40] Die gewünschte Unabhängigkeit von proprietären Katalogformaten motiviert (auch) dort zur Schaffung eines ‚eigenen’ Produktdatenstandards.

Die Entwicklung eines solchen ‚proprietären’ Standards sollte sich auf jeden Fall an die W3C-Standards anlehnen, d.h. für jedermann einsehbare Drafts[41] publizieren und halbwegs präzise definieren, wie die Weiterentwicklung aussieht. Anregungen zur Vorgehensweise, Dokumentation und Vermarktung bietet www.xbrl.org: XBRL ist eine XML-Spezifikation, die Software-Entwicklern und Anwendern zur Abbildung/zum Austausch von Informationen aus Finanz-Berichten (Bilanzen, GuV, etc.)

[38] D.McGoveran: Business Semantics, in EAI Journal 10/2000, S. 10
[39] vgl. o.Verf.: FpML set to become industry standard, in Risk 12/2000, S. 22
[40] vgl. R.Ade: Mangelnder Austausch, in iX 1/2001, S. 122ff
[41] konsensfähiger Entwurf

dient. Derzeit liegt eine entsprechende XBRL Taxonomie für US GAAP[42] vor (IAS und ‚lokale' Berichtstaxonomien (z.B. HGB) sind geplant).[43]

Die (den ausgetauschten XML-Dokumenten via Schemata/DTDs beigelegten) Standards werden zur automatischen Validierung eingesetzt und bieten damit die Möglichkeit für eine deklarative Kontrolle der Verarbeitung. Hierbei ist zu überlegen, ob immer oder nur in besonderen Fällen das empfangene XML zu validieren ist, bevor es interpretiert wird. Ohne eigenen Prüfaufwand kann man dann aber sicher sein, mit formal korrekten und vollständigen Daten zu arbeiten.[44]

Die Schaffung alltagstauglicher Schnittstellen von oder zu vorhandenen Datenbeständen ist alles andere als trivial: die Existenz eines digitalen Produktkatalogs, der die Prüfung gegen die DTD durch einen validierenden Parser besteht, bedeutet noch nicht, dass er nicht zahlreiche fehlerhafte Informationen enthält. Eventuelle Längen- oder Einheitvorgaben des Katalogformats lassen sich nur mit Hilfe eines XML-Schemavalidierers testen. Zuverlässige Überprüfungen auf innere Konsistenz (z.B. ob die Daten auch tatsächlich vorhanden sind) sind oft nur durch Eigenentwicklungen - allerdings auf Basis von XML-Technologien wie XSLT - möglich.

Darüber hinaus ist zu beachten, dass auch ein XML-basierter Produktkatalog keineswegs ohne weiteres in einen anderen XML-basierten Katalog konvertiert werden kann. Der reinen Lehre des W3C folgend, müsste ein XSLT-Skript zur Transformierung ausreichen. Doch dieser Weg könnte bei kompletten Produktkatalogen schlicht an den Datenmengen scheitern. Es ist daher im Vorfeld zu prüfen, ob (bzw. ab welchem Datenvolumen) die derzeit existierenden XSLT-Prozessoren überfordert sind.[45]

2.3 XML im Risikomanagement

Strukturierte Finanzprodukte, das Management von Kredit-, Markt- und operationalen Risiken, und hier insbesondere Transaktionsrisiken, sowie (demnächst) bankweites Risikomanagement sind Aspekte, die immer stärker in den Fokus der Finanzinstitute rücken – damit aber noch stärker die Problematik, große Datenmengen in meist unterschiedlichen DV-Systemen zu verwalten. Aus der konsequenten Verwendung von XML als technisches Vehikel zur Abbildung des Datenflusses könnten hier Effizienzeffekte resultieren:[46]

1. Geschäftsabschluss: Bei OTC-Geschäften ist mit dem Kontrahenten ein entsprechender Vertrag auszuarbeiten, der zwischen den Vertragspartnern abgestimmt werden muss. Da bspw. für strukturierte Produkte das Vertragswerk sehr umfangreich und komplex ist, stellt deren Abstimmung einen aufwändigen, sich wiederholenden Prozess dar.

[42] vgl. S. de la Fe/C.Hoffmann/E. Huh: XBRL Taxonomy Financial Reporting for Commercial and Industrial Companies, US GAAP, http://www.xbrl.org/us/gaap/ci/2000-07-31/us-gaap-ci-2000-07-31.pdf
[43] vgl. hinsichtlich des Marktpotentials PwC (Hrsg.): Value Reporting Forecast: 2001 Trends in Corporate Reporting, o.O., S. 12ff
[44] mit einem geeignetem Repository ließe sich so in beliebig feiner Granularität eine deklarative Kontrolle für Zugriffe, Transaktionen und Protokollierung auf der Ebene von XML-Elementen und –Attributen realisieren. MlFi, eine XML-basierende Sprache, spezifiziert bspw. Finanzkontrakte, die anschließend einer automatischen Bewertung und Verwaltung im gesamten Produkt-Lebenszyklus zugänglich sind. Vgl. o.Verf.: MlFi, in Risk 1/2001, S. 50
[45] am 12.12.2000 hat das W3C einen ersten Entwurf für die nächste Generation der XSL-Transformationen veröffentlicht. XSLT 1.1 soll vor allem (sobald entsprechende Werkzeuge vorhanden sind) ermöglichen, dass mittels eines einzigen Stylesheet mehrere Ausgabedateien erzeugt werden können (dies leisteten bislang nur proprietäre Erweiterungen einzelner Entwickler)
[46] vgl. M.Jost/A.Kalhoff: Internet-Technologie im Risikomanagement, in geldinstitute 9/2000, S. 42ff

Durch die Aufbereitung der Vertragsdaten als XML-Dokumente lässt sich die Überprüfung neuer Vertragsversionen automatisieren, indem vereinbarte Änderungen jeweils von den Partnern in ihren XML-Dokumenten eingearbeitet werden. Der Austausch der Dokumente findet elektronisch statt und wird auch elektronisch mit Hilfe der DTDs bzw. XML-Schemata auf Übereinstimmung validiert. Hierdurch sinkt das Dokumentationsrisiko (als Teil des Transaktionsrisikos) und damit das zu allokierende Kapital für das operationale Risiko.

Bei Geschäften mit standardisierten Finanzinstrumenten, für die bereits Rahmenverträge existieren, brauchen ‚nur' die das Produkt betreffenden Daten abgestimmt werden. Auch dies geschieht über einen XML-basierten Datenabgleich.

2. Datentransfer: Ist das Geschäft in den juristischen Bestand des Kreditinstitutes übergegangen, so ist eine regelmäßige Messung der Markt- und Kreditrisiken des Geschäftes durchzuführen. I.d.R. existieren unterschiedliche Datenbanken in der Handelsabteilung (Front Office) und im Risikocontrolling (Middle Office), so dass die Geschäftsdaten von der Handelsabteilung an das Risikocontrolling zu übermitteln sind. Aufgrund der großen Zahl von unterschiedlichen Finanzprodukten ist eine flexible, leicht zu erweiternde Datenübertragung angebracht.

   Im Rahmen einer XML-basierten Lösung extrahiert eine XSLT-Anwendung im Handelssystem die für die Risikomessung benötigten Daten und sendet diese an das DV-System des Risikocontrollings. Mit einer auf die Datenformate des Risikocontrolling-Systems abgestimmten XSLT-Anwendung lassen sich dann die Daten in die Middle-Office-Datenbank importieren.

   Für die Abwicklung sowie für das Rechnungs- und Meldewesen der gehandelten Finanzprodukte im Back Office werden ebenfalls Daten aus der Handelsabteilung (und ggfs. auch aus dem Risikocontrolling) benötigt. Der Datentransfer erfolgt analog via XML.

3. Analyse: Die Bereitstellung von Marktdaten, die u.a. von der Handelsabteilung zur Opportunitätsberechnung möglicher Geschäfte, vom Risikocontrolling zur Risikomessung und vom Back Office zur Erstellung der Grundsatz-I-Meldungen benötigt werden, erfolgt von einer zentralen Stelle aus (um einen ansonsten nötigen Datenabgleich zwischen den beteiligten Bereichen zu vermeiden). Sofern die DV-Systeme in den einzelnen Bereichen mit unterschiedlichen Datenformaten arbeiten, werden die Daten beim Import transformatiert. Zur Minimierung von Fehlerquellen und zur Effizienzsteigerung ist eine einheitliche Lösung anzustreben – d.h. Daten sollten im XML-Format geliefert und mit den für die einzelnen Systeme entwickelten XSLT-Anwendungen für den Import formatiert werden.

Das hier knapp dargestellte „straight-through-process"-Szenario sollte lediglich aufzeigen, dass

- sich bei konsequenter Anwendung der XML-Technologien der Aufwand für Definitionen, Implementierung und Pflege von Schnittstellen zwischen unterschiedlichen Systemen erheblich reduzieren lässt

- das Datenmanagement (aufgrund von immer neuen Finanzmarktprodukten sind die Schnittstellen ständig zu erweitern und Teilbereiche sind neu zu definieren) durch den Einsatz von Metadaten-Formalismen wesentlich effizienter gestaltet werden kann, weil lediglich die DTDs und XSL-Anwendungen ggfs. anzupassen bzw. zu erweitern sind
- durch die (einheitliche) Modellierung[47] von Finanzprodukten auf viele bilaterale Schnittstellenformate verzichten werden kann.

Ein weiterer wichtiger Aspekt ist die Sicherstellung der *Datenqualität*: Die Akzeptanz und die Aussagekraft aktueller Steuerungssysteme leidet, wenn die Datenqualität mangelhaft ist. XML-Technologien bieten hier die Möglichkeit einer quasi ‚automatischen' Prüfung (sowohl gegen die vereinbarte Syntax als auch gegen die Semantik).

Aufgrund der Dynamik des Bankgeschäfts muss auch die „Risk-IT" ständig den Veränderung angepasst und erweitert werden. So hat RiskMetrics das Potenzial von XML auch in dieser Hinsicht erkannt: RiskMetrics kündigte eine XML-basierte ‚risk-engine' an („RiskServer"), der in bestehende Anwendungen für Zwecke des Risikomanagement integriert werden kann.[48] ‚RiskServer' hat einen festen Verweis zu 'DataMetrics' (Sammlung historischer Zeitreihen) und umfasst diverse Algorithmen (Berechnungsmethoden, Stress-Testing, Risiko-Analysen) basierend auf RiskMetrics Methodology.

Im Risikomanagement stellen auch die Anforderungen des Gesetzgebers eine äußerst komplexe Herausforderung dar. Zwar verfügt jedes einzelne Handelssystem über eigene Tools zur Risikobegrenzung (risk engines), diese sind aber nicht in der Lage, die vielfältigen Wechselbeziehungen zwischen den verschiedenen Handelsaktivitäten zu berücksichtigen (Gesamtbankrisiko).

Das Problem liegt insgesamt darin, dass es sehr unterschiedliche Datenquellen/Speicherformate gibt. Ein möglicher (Aus-)Weg liegt darin, eine zentrale Datenbank (Data Warehouse) anzulegen – mit dem Nachteil, dass die Daten immer wieder konvertiert und zwischengespeichert werden müssen und die Rekonsolidierung und Aktualisierung der Daten extra Zeit und Rechnerkapazitäten benötigt. Wird ein neues Produkt im Front Office eingeführt, was sehr häufig vorkommt, muss im Data Warehouse der Code geändert werden (sofern adäquate Schnittstellen-Technologien fehlen – siehe Kapitel 3).

Ein alternatives ‚Virtuelles Data Warehouse' nutzt hingegen Connectivity-Technologie, um Daten aus allen Systemen aktuell und in Echtzeit zu konsolidieren. Es sind Regeln zu hinterlegen, die definieren, welche produktiven Systeme für welche Abfrage angespielt werden müssen. Diese Zielsysteme reichen ihre Daten über den Server direkt an die Benutzerworkstation durch. Die Auswertungen werden also dort gefahren, wo die Daten tatsächlich gespeichert sind, und die Ergebnisse werden zentral zusammengeführt. Auch daraus ergibt sich eine einheitliche Sicht auf die

[47] die Einheitlichkeit wird dadurch gewährleistet, dass die Modellbildung deduktiv, d.h. durch ‚eigenes Nachdenken' erfolgt. Das induktive Modellierungsprinzip (d.h. die schlichte Adoption der spezifischen Modellwelten entsprechender Liefersysteme) wäre hier fehl am Platze, weil nicht zielführend im Sinne des o.g. Schnittstellenmanagements

[48] vgl. P.D.Tailor: RiskMetrics to Introduce RiskServer – the First Internet-Based Risk Engine, in RiskMetrics Journal 11/2000, S. 5

Handelsdaten für Auswertungen, Risikobewertung, Marktanalysen und Berichtswesen. Eine solche Lösung soll flexibler und schneller sein.[49]

[49] vgl. R.Bastian: Virtuelles Data Warehouse, in geldinstitute 3/1997, S. 56f

3 Schnittstellenmanagement

Bei der Zusammenarbeit von zwei oder mehreren Geschäftsbereichen entstehen organisatorische und/oder systembedingte Übergänge. Solche Übergänge werden im folgenden als „Schnittstelle“ bezeichnet. Systemschnittstellen sind mithin durch den (innerbetrieblichen) Datenaustausch zwischen fremden Systemen und/oder Komponenten bedingt.

Technische wie organisatorische Schnittstellen sind nicht statischer Natur. Mit der Änderung von (IT-)Strategien und Strukturen verändern sich auch die bestehenden Schnittstellen. Da die Geschäftsbereiche im ständigen Informationsaustausch stehen und Bankgeschäfte bspw. zwecks interner Ergebnisrechung und externem Rechnungswesen laufend darzustellen sind, muss es Ziel sein, einen möglichst reibungslosen Strom an Informationen über ein geeignetes Schnittstellenmanagement sicherzustellen.

Ein solches Schnittstellenmanagement wird hier als vertragliche Übereinkunft gesehen, Konsistenz in dem Sinne sicherzustellen, dass ein bestimmtes Finanzinstrument trotz unterschiedlicher Zuliefersysteme bankweit identisch abgebildet, bewertet und dargestellt wird. Ändern sich die Schnittstellen, sind die Verträge zu ändern bzw. anzupassen.

3.1 Vertragsgestaltung

Aus betriebswirtschaftlicher Sicht birgt die Forderung nach vertraglich gestalteten Schnittstellen eine Fülle von Problemen. Die nachfolgenden Ausführungen können die betriebswirtschaftliche Schnittstellenproblematik als solche nicht lösen, können aber dazu beitragen, die Determinanten der Vertragsgestaltung klarer herauszuarbeiten und so eine Basis für konkrete, technologische Lösungsansätze vorzubereiten.

Wesentliches Gestaltungsprinzip bei der Schnittstellenbildung ist die Orientierung am Datenfluss.[50] Hierbei vermitteln Schnittstellen zwischen Quell- und Zielsystemen. Der Informationsfluss, der durch diese Schnittstellen übermittelt wird, besteht aus Ausdrücken einer Sprache, deren Vereinbarung zwischen Quell- und Zielsystemen vorausgesetzt wird. Der Vertragsgestaltung fällt demnach die Aufgabe der Ausformung einer gemeinsamen sprachlichen Syntax und Semantik zu:

- die syntaktische Vertragsgestaltung ist ein informationstechnisches Problem ohne unmittelbare betriebswirtschaftliche Relevanz. Sie erfordert, dass die Signale, die durch die Schnittstelle physikalisch übermittelt werden, von Sender und Empfänger in gleicher Weise als Zeichen der vereinbarten Sprache interpretiert werden. Darüber hinaus muss gewährleistet werden, dass die Kombinationen von Zeichen durch Sender und Empfänger nach den gleichen syntaktischen Regeln als übereinstimmende Ausdrücke der zugrundeliegenden Sprache gebildet bzw. verstanden werden
- in semantischer Hinsicht muss dafür Sorge getragen werden, dass die übermittelten Ausdrücke jeweils von Sender und Empfänger - bei übereinstimmender Syntax - mit gleichen Bedeutungen belegt werden, um Interpretationsspielräume

[50] das Prinzip der Funktionsorientierung richtet die Gestaltung von Schnittstellen hingegen an den Aufgaben (Funktionen) aus, die von den verknüpften Teilen des betrieblichen Informationssystems erfüllt werden sollen. Über solche Schnittstellen werden zwar real Informationsströme (ebenso wie bei datenorientiert gestalteten Schnittstellen) ausgetauscht, doch die beteiligten Systeme sprechen sich gegenseitig nicht im Hinblick auf die auszutauschenden Informationen, sondern hinsichtlich der Aufgaben ab, die vom jeweils gerufenen Teilsystem für das jeweils anfordernde Teilsystem erfüllt werden sollen. Vgl. S.Zelewski: Schnittstellen bei betrieblichen Informationssystemen, Arbeitsbericht 6, Universität Köln 1986, S.6ff

zu vermeiden. Umgekehrt ist auch sicherzustellen, dass gleiche Sachverhalte von Sender und Empfänger mit den gleichen Ausdrücken bezeichnet werden, da sonst inhaltlich relevante Informationen infolge unterschiedlicher Bezeichnungen übersehen werden könnten. Beide Bedingungen definieren zusammen die Konsistenz einer Schnittstelle – wodurch erst Finanzprodukte aus unterschiedlichen Anwendungen identisch interpretiert und dadurch auch gleich behandelt werden können. Nur wer über den Umfang der relevanten Daten und ihrer semantischen Bedeutung informiert ist, kann in einem nächsten Schritt eine Integration „sinn"-voll (mittels der so gewonnenen Meta-Daten) durchführen.

Hinsichtlich der Qualität von Schnittstellen können folgende betriebswirtschaftliche Beurteilungskriterien zugrundegelegt werden:[51]

- Korrektheit ist dann gegeben, wenn eine syntaktisch unverfälschte Informationsübertragung durch die Schnittstellen vorliegt
- Konsistenz erstreckt sich auf die semantische Schnittstellendimension durch die Forderung nach gleichen Interpretationen für gleiche Ausdrücke und gleichen Ausdrücken für gleiche Sachverhalte
- Vollständigkeit ist dann gegeben, wenn eine Schnittstelle alle Leistungsmerkmale ihrer Spezifizierung erfüllt.
- Reagibilität liegt dann vor, wenn die spezifizierten Schnittstellenleistungen in einem solchen zeitlichen Intervall erfolgen, dass die zeitlichen Reaktionsbedingungen der miteinander verknüpften Systeme nicht verletzt werden. Die vorgehaltenen Informationen sollen die betroffenen Sachverhalte folglich mit dem Aktualitätsgrad abbilden, der von den informationsabrufenden Systemen zur Erfüllung ihrer Verarbeitungsziele benötigt wird.
- Güte misst die Qualität der Leistungserfüllung. Die Güte einer Schnittstelle mit informationsvermittelnder Funktion lässt sich etwa durch das Ausmaß konkretisieren, in dem es ihr gelingt, den Informationsstrom, der sie durchsetzt, auf die jeweils relevanten Informationen zu reduzieren.[52]
- Kontrollierte (In-)Transparenz hinsichtlich der Arbeitsweise der unmittelbar angesprochenen Schnittstelle, weil es für die Bereitstellung der angeforderten Ergebnisse unerheblich ist, wie diese zustandegekommen sind. Sie räumt damit der Schnittstelle den Freiheitsgrad ein, selbst über die Art zu entscheiden, in der die angeforderten Ergebnisse von anderen Teilsystemen herbeigeschafft werden. So bietet sich Freiraum für den Entwurf von „intelligenten" Schnittstellen, die den Prozess der Ergebnisbereitstellung, insbesondere die Lokalisierung der Ergebnisquellen, eigenständig planen, ausführen und überwachen.
- Flexibilität ermöglicht, die Schnittstellen an veränderte Gestaltungsanforderungen und ebenso an veränderte (erweiterte) Gestaltungsmöglichkeiten anzupassen (dient der Adaption an informationstechnische Fortschritte).

Ein weiteres wichtiges Beurteilungskriterium ist der Verbreitungsgrad einer Schnittstellenspezifikation: Schnittstellen die einem Standard folgen, haben eine größere Akzeptanz, weil allgemeingültige Spezifikationen

- den Einarbeitungsaufwand begrenzen,
- die Kommunikation zwischen den Entwicklern sowie
- den Datenaustausch erleichtern und
- die Integration durch Werkzeuge vereinfachen.

[51] vgl. S.Zelewski: a.a.O., S. 22f

[52] während bspw. das Reporting auf das Wesentliche zu reduzieren ist, ist die Nachvollziehbarkeit bis in Detail sicherzustellen

Während einerseits die oft langwierige und aufwändige Standardisierung durch offizielle Gremien an Bedeutung gewinnt (W3C), trifft man andererseits häufig aber auch auf den Begriff Standard im Sinne von „Industriestandard“ (oder besser: „Pseudo-Standard“), was lediglich einen hinreichenden Konsens und eine große Verbreitung meint. Der Vorteil dieser „proprietären Standards“ liegt darin, dass man frühzeitig mit einer gewissen Infrastruktur und den nötigen Werkzeugen aus der Hand einer Organisation unterstützt wird.

Die historisch/kulturell bedingte Vielfalt unterschiedlicher „Bezeichner“ für (syntaktisch) gleiche bankbetriebswirtschaftliche Begriffe sowie verschiedener „Bezeichner“ für inhaltsgleiche Sachverhalte führt dazu, dass die Schaffung einer semantischen Vereinheitlichung keine triviale Angelegenheit ist. Sie verlangt daher nach einer betriebswirtschaftlich geprägten Modellierung, in dem die relevanten Nutz-Daten analysiert, strukturiert und (über das Mittel der Abstraktion) generalisiert werden. Ziel dieser Datenmodellierung muss es sein, soviel wie möglich an Syntax und Semantik dokumentiert (in Form eines ‚Beipackzettels’ für die Nutz-Daten) dem Datentransfer zur Verfügung zu stellen.

Um die erforderlichen Meta-Daten zu bestimmen und zu strukturieren, spielt die Architektur der vorhandenen Umgebung eine zentrale Rolle. Diese wird bestimmt durch die operativen Quellsysteme, externen Systemdienste (Gattungsdaten, Partnerdaten, Marktdaten, usw.) sowie die abzubildenden Geschäftsprozesse.

Vorangehend wurden die Dimensionen der Schnittstellengestaltung allgemeingültig aus betriebswirtschaftlicher Sicht skizziert. Nachfolgend werden Ansätze zur konkreten Ausfüllung dieses Gestaltungsrahmens vorgestellt.[53]

3.2 Schnittstellenvielfalt

In den letzten Jahren kam es zu einem sich ständig ausweitenden Angebot teils neuartiger Handelsprodukte, erhöhter Risikosensibilität und in Folge verschärften Meldepflichten durch die Aufsichtsbehörden. Die internen (z.B. MaH) und externen (z.B. CAD) Anforderungen an die Verfügbarkeit von Informationen über Geschäftsaktivitäten haben hinsichtlich Vergleichbarkeit, Detaillierungsgrad, Aktualität, Qualität und Menge der Informationen eine neue Dimension erreicht.

Aufgrund fachlicher Notwendigkeiten existieren in den operativen Systemen verschiedene Sichtweisen und Ansprüche an die lokalen Daten, die für globale Aus- und Bewertungszwecke mittels der o.g. „Verträge“ zu integrieren sind. Informationstechnisch liegt eine heterogene Applikationslandschaft bzw. eine Vielzahl von Schnittstellen vor und die Notwendigkeit, die anfallenden Daten homogenisiert zusammenzuführen. Darüber hinaus ist die notwendige Schnittstellenqualität hinsichtlich der o.g. Kriterien zu gewährleisten. Die Herausforderung besteht insgesamt darin, mit den verschiedenen Zielsystemen, Dateiformaten und unterschiedlichen Kommunikationswegen komplexitätsreduzierend umzugehen.

3.2.1 Schnittstellenarithmetik

Die numerischen Dimensionen eines sog „Point-To-Point“-Datenaustauschprozesses zeigt folgende Grafik:

[53] zur Kapselung von Schnittstellen für die Host-Integration mittels XML und Cobol vgl. H. Sneed: Host-Integration mit Bordmitteln meistern, in Computerwoche 15/2001, S. 24

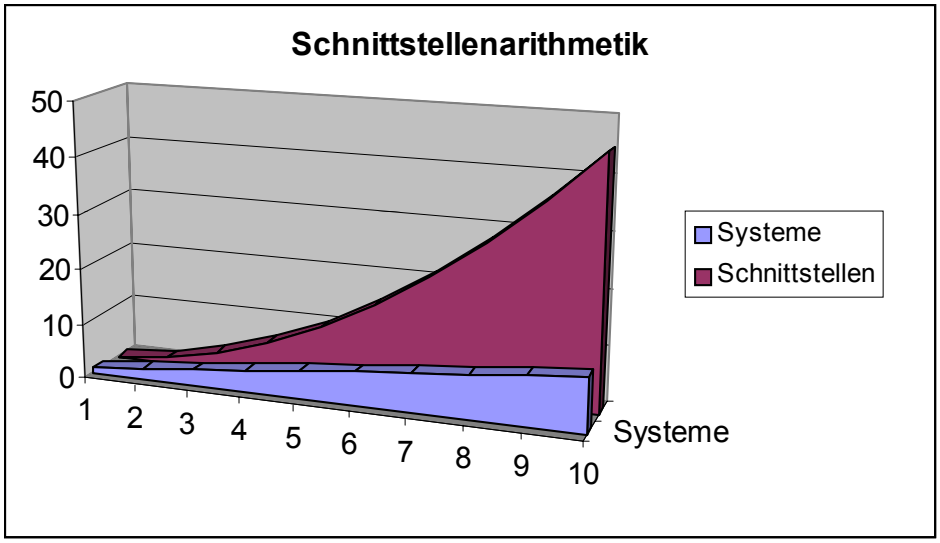


ABBILDUNG 8: SCHNITTSTELLENARITHMETIK

Bei nur 10 unterschiedlichen Systemen und Formaten mit bidirektionalem Datenfluss, ergibt sich eine notwendige Schnittstellenanzahl von 45 {[n*(n-1)]/2}: während die Anzahl der Systeme linear wächst, wachsen die Schnittstellen exponentiell. Damit stellt sich die Frage nach Lösungen, die dieses Phänomen effizient kontrollieren: die nummerische Schnittstellenexplosion erfordert auf jeden Fall eine prinzipielle technologische Neuorientierung, um die Dynamik des Bankgeschäfts problemadäquat abzubilden. Es gilt darüber hinaus konfigurierbare (anstatt ‚hard-coded') Lösungen einzusetzen, um Wartungsintensität und Fehleranfälligkeit zu minimieren.

Zunächst ist zu fragen, was die Entwicklung einer Schnittstelle grundsätzlich beinhaltet. Das sind[54]

- die Schaffung einer technologischen Transportmöglichkeit zwischen den beteiligten Systemen (Logging, Message Transfer, Routing, ...)
- die Konvertierung (mapping) der Daten des Quellsystems in das Format und die Struktur des Zielsystems
- die Entwicklung von Programmen zur
  1. Pflege statischer Referenzdaten für alle Systeme (Gattungsdaten, Kontrahenten, ...)
  2. Validierung von Dateninhalten
  3. Bearbeitung von Daten- und Mappingfehlern und
  4. Bestätigung von Datentransfers
  5. Abstimmung von Datenbeständen zwischen Quell- und Zielsystem.

Aufgrund dieser Regelmässigkeit bei der Schnittstellenentwicklung liegt es nahe, auf Basis einer Schnittstellen-Architektur sowohl (i) den Programmcode durch Wiederverwendung (d.h. Standardisierung der Funktionen Mapping, Datentransfer, Datenvalidierung, Fehlerbehandlung und Bestandsabgleich) als auch (ii) die Anzahl der Schnittstellen durch semantische Abstraktion zu reduzieren:

[54] vgl. J. Büchler: Pack den Tigra in den Tank, in IT FOKUS 4/2001, S. 22

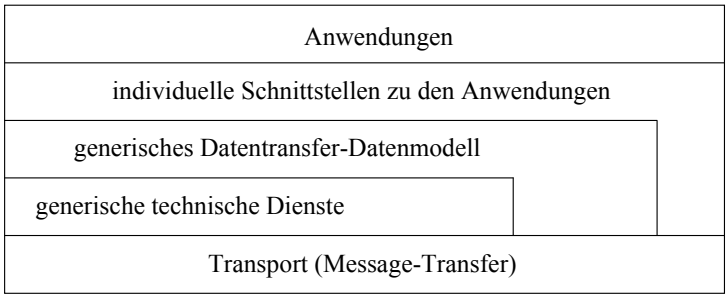


ABBILDUNG 9: SCHNITTSTELLEN-ARCHITEKTUR

Quelle: in Anlehnung an J. Büchler: Pack den Tigra in den Tank, in IT FOKUS 4/2001, S. 22

Für die technischen Integrationsdienste kommt der Einsatz von Middleware in Betracht (verstanden als ein ‚Technologie-Bündel' für Transfer, Routing und Transformation von Messages sowie Object brokering). Eine andere wichtige Integrationskomponente ist ein Datentransfer-Datenmodell und seine entsprechende Implementierung mittels XML.

Das Management von Schnittstellen ist (auch) ein Kernproblem der sog. ‚Enterprise Application Integration' (EAI)-Lösungen.[55] Eine EAI-Lösung kombiniert Technologien und Prozesse zwecks Point-to-Point-Verknüpfungen von Anwendungen und kann als umfassende Ausprägung von Middleware-Services begriffen werden. Sie umfassen[56]

- traditionell definierte Dienste, wie Message-Oriented Middleware (MOM), Object Request Broker (ORB), Remote Procedure Calls (RPC) und Transaction Processing Monitoring (TPM), aber auch neuere Dienste, wie
- Application Adapters, die den Zugriff auf die Applikationsdaten und –funktionalität, die Datentransformation, das Message Routing wie auch das Prozessmanagement ermöglichen.

EAI sind softwaretechnisch Komponenten, die den Datenaustausch auf Applikationsebene ermöglichen. Sie gehören zur Familie der Middleware, die das Rückgrat für die technische Verbindung zwischen Anwendungen bildet. Die meisten Hersteller bieten eine modular aufgebaute Softwarearchitektur, die sich um Speicher- und Übertragungskapazität, Netzwerkverbindungen, Transaktionsschutz, Sicherheitsprüfung und Systemmanagement-Funktionen kümmert.[57]

Hinsichtlich ihrer Architektur sind grundsätzlich zu unterscheiden:[58]

- „Hub and Spoke": alle Applikationen und Systeme sind mit einem zentralen Server („hub") verbunden. Die Kommunikationskanäle zu diesen Anwendungen sind die „spokes". Der „hub" stellt Dienste (Messaging, Routing, Transformation) zentral zur Verfügung; Konnektoren/Adapter[59] eröffnen den Zugang zu jedem „spoke". Durch eine solche Architektur gelingt es, das Wachstum der

[55] vgl. zur Software-Architektur als Rahmen für Software-Anwendungen R. Nußdorfer: Evolution durch neue Technologien, in IT FOKUS 1-2/2001, S. 39ff
[56] vgl. R.Lory: Middleware, in Schweizer Bank 12/2000, S. 92
[57] einen guten Überblick über die EAI-Technologien gibt R. Oberdorfer: Allround-Adapter, in iX 5/2001, S. 136ff
[58] zu weiteren vgl. J.C.Lutz: EAI Architecture Patterns, in EAI Journal 3/2000, S. 64ff
[59] für die Kommunikation zwischen dem Mysap.com Workplace und den Mysap-Komponenten kommen bspw. die SAP eigenen Schnittstellen Business Application Programming Interfaces (BAPIs) zum Einsatz. Vgl. o.Verf.: SAP setzt auf XML für übergreifende Integration, in Computerwoche 18/2000, S. 26. Bei diesem „Datenaustausch über Schnittstellen" werden nur Nutzdaten ausgetauscht (eine Nachricht gibt es nicht, da die Daten als Parameter an die Schnittstelle übergeben werden – was zu einer engen Kopplung führt)

Schnittstellen zu linearisieren, d.h. die Schnittstellenanzahl bei einer Point-to-Point-Verknüpfung von [n*(n-1)/2] auf n zu drücken:

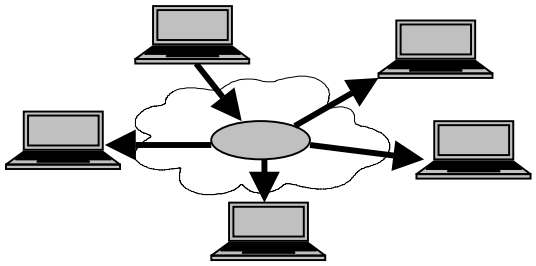


ABBILDUNG 10: HUB AND SPOKE

- „Bus/Service-orientiert“: die Dienste sind über das gesamte System verteilt. Relevante Informationen werden repliziert. Eine Bus-Architektur impliziert einen gemeinsamen Transport-Dienst für alle angeschlossenen Systeme[60]; eine solche Architektur erfordert eine komplexerer Konfiguration wegen der Replizierung und Koordination:

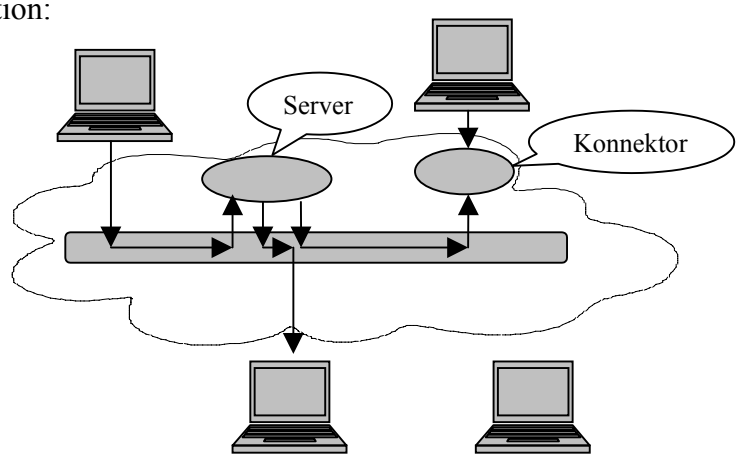


ABBILDUNG 11: MESSAGEBUS (PIPE)

EAI-typisch ist es, nicht Point-to-Point-Verbindungen zu definieren, sondern eine unternehmensweite Infrastruktur zum generellen Datenaustausch zu installieren. Wer wem welche Daten zu welchem Zeitpunkt unter welchen Bedingungen schickt, ist eine Frage der Workflow-Definition: Es bieten sich hier strategische und operative Vorteile, ein solches Schnittstellenmanagement zentral über eine Instanz abzuwikkeln.

EAI sorgt mithin für die transparente Zusammenführung von unterschiedlichen Systemen. Der Nutzen dieser Technologie ergibt sich aus dem betriebswirtschaftlichen Blickwinkel, lassen sich doch mittels EAI komplette Geschäftsprozesse abbilden und integrieren. EAI-Anbieter bieten eine komplette Integrationsarchitektur mit sämtlichen Schlüsselfunktionen zum Austausch von Informationen in Transfer-, Transformations- und Erweiterungsprozessen - somit die Möglichkeit, die Schnittstellenproblematik erfolgreich zu managen.

[60] beim Messaging wird eine Nachricht in einem festgelegten Format übermittelt. Neben den Nutzdaten enthält die Nachricht Kontrollinformationen, die dem Empfänger mitteilen, wie er die Nachricht verarbeiten soll (was zu lose gekoppelten Systemen führt)

3.2.2 Enterprise Application Integration (EAI)

Im Folgenden wird ein EAI-Modell skizziert, das die notwendigen Funktionalitäten eines Schnittstellenmanagments in Gänze darstellt. Anschließend wird XML entsprechend diesem Modell positioniert.

Zur Erinnerung: Die abzubildende Problemstellung liegt darin, Daten (Geschäfts-, Produkt-, Markt- und sonstige Daten) einheitlich bankweit zur Verfügung zu stellen, um auf der jeweils gewünschten Aggregationsebene entsprechende Informationen zu gewinnen. Diese Anforderungen können nur dann erfüllt werden, wenn eine integrierte, anwendungsübergreifende Sicht auf die relevanten Daten besteht, welche die - aufgrund der unterschiedlichen Geschäftsarten und Handelserfordernisse notwendige - Applikationsvielfalt einerseits beibehält, andererseits aber auch eine vereinheitlichte Darstellung der Geschäftsdaten auf einer übergeordneten Ebene stabil und sicher ermöglicht.

Die Grundidee ist die Schaffung eines „Single Point of Controls“ für alle Schnittstellen: zu jeder Applikation wird nur eine technische Schnittstelle aufgebaut, die Integrationslogik wird zentral vorgehalten: [61]

1. „System-Management“ überwacht und steuert sämtliche physischen und logischen Ressourcen (Leitungsverbindungen, Hardware, Software, etc.). Sie wacht über die Verfügbarkeit und Performance aller Ressourcen, Applikationen und Systeme
2. „Message-Handling“ sorgt für den physischen Transport der Daten zwischen Applikationen und Systemen. Je inhomogener das vorhandene Umfeld, desto mehr Adapter werden benötigt.
3. „Broking“ und „Conversion“ nimmt vom Inhalt abhängige Steuerungs- und Umsetzungsvorgänge wahr (arbeitet damit auf betriebswirtschaftlicher Ebene syntaktisch und semantisch). Da der betriebswirtschaftliche Prozess über Applikationen hinweg verstanden sein muss, steckt hier der größte Aufwand. Sie koppelt Anwendungen auf betriebswirtschaftlicher Ebene.

Aufgrund einer OVUM-Studie, die EAIs umfassend analysierte, wird folgendem Schnittstellen-Modell gefolgt:[62]

<table>
<tr><td colspan="2">Process-Management
Transformation, Koordination, Management</td><td rowspan="3">Development:
Process-Modellierung, Spezifikation der Transformation, Schnittstellen-entwicklung,</td><td rowspan="3">Management:
Zuverlässigkeit, Verfügbarkeit, Skalierbarkeit, Überwachung</td></tr>
<tr><td colspan="2">Transformation:
Identifikation, Validierung, Synchronisation, Transaktionsunterstützung, Transformation, Routing</td></tr>
<tr><td>Connectivity:
Kommunikation, Adressierung, Zustellung, Sicherheit</td><td>Resource interfacing:
Übersetzung der Schnittstellen, Metadata-repository</td></tr>
</table>

ABBILDUNG 12: EAI-MODELL (OVUM, EAI-REPORT JUNI 1999)

[61] vgl. D.Blum: EAI-Beispiele im R/3-Umfeld, in Computerwoche 47/2000, S. 28f

[62] vgl. K.Ring/N.Ward-Dutton: Enterprise Application Integration: Making the Right Connections, Ovum Ltd., London 1999, S. 41ff. Vgl. auch R.Nußdorfer: Daten- und Middleware-Integration, in it FOKUS 9/2000, S. 55

Die Funktionalität eines Schnittstellenmanagements kann demnach in 4 technisch/betriebswirtschaftliche Dienste aufgetrennt werden, die um Entwicklungs- und Management-Funktionalität ergänzt werden:

1. **Connectivity**: entsprechende Dienste ermöglichen unter Verwendung von Middleware die Integration auf Datenebene; Kommunikations-[63], Adressierungs-[64], Sicherheits-Dienste extrahieren Daten aus einer Applikation und transportieren sie zu einer anderen, wobei i.d.R. technische Systemgrenzen zu überwinden sind (Hardware, Software, Protokolle).
2. **Transformation**: solche Dienste unterstützen die Integration auf Objektebene. Hier werden die bereitgestellten Daten in Formate umgewandelt, welche das Zielsystem interpretieren kann[65]: zur Identifikation der eintreffenden Daten stehen Metadaten-Repositories und Datenspeicher zur Verfügung; Synchronisierungsdienste steuern den zeitlichen Ablauf/das Zusammenspiel bei der Abarbeitung von Transformationen. Der Output der Transformationsdienste muss ggfs. an unterschiedliche Zielsysteme versandt weren. Routing Services unterstützen dies. Sog. ‚Transaction Processing Services' überwachen und unterstützen die Durchführung, Vollständigkeit und Konsistenz bei komplexen, mehrere Systeme betreffenden Operationen. Insgesamt lässt sich der Transformationsprozess wie folgt darstellen:[66]

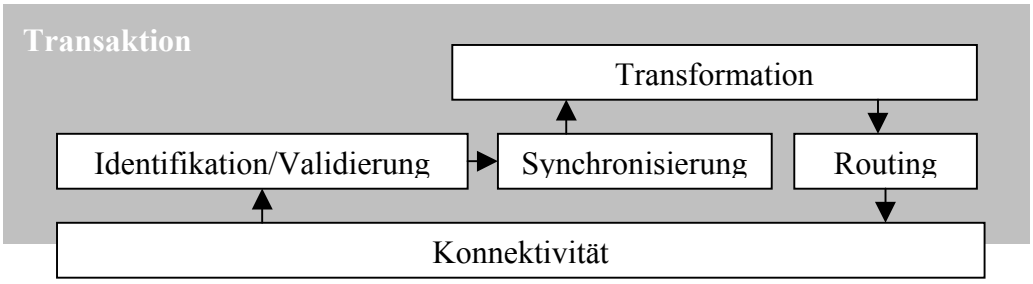


ABBILDUNG 13: TRANSFORMATIONPROZESSE

3. **Process-Management**: legt die Basis für die Umsetzung durch gezielte Steuerung geplanter Transaktionen. Dabei gelten alle Applikationen eines Unternehmens als potenzielle Integrationsobjekte. Unterstützt wird insbesondere die Integration auf Prozessebene, d.h. Prozesse (workflows) und Abläufe werden organisiert und administriert (EAI-Lösungen müssen sich an den Geschäftsprozessen orientieren).
4. **Recource Interface**: ‚normalisiert' die Art, wie Zugang und Interaktion mit den Integrationsobjekten (Datenbanken, Dateien, Individualsoftware, Middleware, etc.) erfolgt. [67] Wesentliche Dienste sind: ‚Interface Translation Services', die für die (technische) Integration auf Datenebene sorgen. ‚Metadata

[63] synchrone Kommunikation für ‚request/response'-Interaktionen bei enger Systemkopplung (ein client macht eine Anfrage bei einem Server) oder asynchrone Kommunikation für ‚event-notification'-Interaktionen (ein client informiert eine Anzahl anderer, dass etwas Interessantes vorliegt) bei loser Systemkopplung

[64] die Adressierung kann (i) point-to-point/brokered (bei request/response-Kommunikation), (ii) broadcast/multicast und (iii) publish-and-subcribe (beide bei event-notification-Kommunikation) erfolgen. Die Adressierung ist unabhängig von synchroner oder asynchroner Kommunikation. Publish-and-subscribe bietet die Möglichkeit, Informationen automatisch zu verteilen

[65] Transformations-Funktionen: 1:1-mapping, 1:n-mapping, n:1-mapping, n:n-mapping, via Algorithmen oder zusätzliche request/response-Interaktionen an andere Systeme

[66] vgl. K.Ring/N.Ward-Dutton: a.a.O., S. 44

[67] da es Standardkonnektoren zwischen 2 Applikationen nicht gibt, entsteht hier hoher Arbeitsaufwand. Allein durch das Customizing eines Systems ergeben sich unterschiedliche semantische Abbildungen. Ein Konnektor erleichtert zwar die technische Abbildung, berücksichtigt jedoch keine inhaltlichen Vereinbarungen. Hier muss auf jeden Fall eine individuelle Anpassung erfolgen

Repositoriy Services' unterstützen die (semantische) Integration auf Objekt- und Prozessebene durch Bereitstellung/Verarbeitung von Informationen zur Struktur der Nachrichten; darüber hinaus werden Validierungsprüfungen unterstützt. Die folgende Grafik veranschaulicht diese Dienste im Kontext:[68]

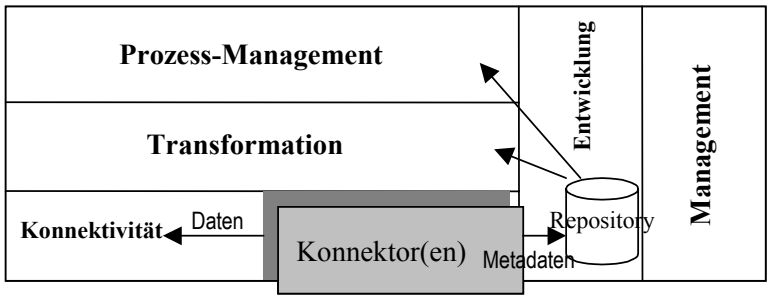


ABBILDUNG 14: KONNEKTOREN

5. **Development**: Entwicklungstools unterstützen die Erstellung und den Aufbau von Integrationslösungen dort, wo die standardisierte Integrationsunterstützung der EAI-Lösungen nicht ausreichend ist. Für die Entwicklung von Transformationen existieren sowohl Drag-and-Drop-Schnittstellen als auch solche, die ausschließlich Script-basiert sind; für die Entwicklung von Prozessmodellen stehen i.d.R. graphische Modellumgebungen zur Verfügung. Weitere Dienste: Testunterstützung und Support für Entwicklerteams sowie EAI-Repository
6. **Management**: enthält Elemente zur Verbesserung der Zuverlässigkeit, Verfügbarkeit und Skalierbarkeit. Für die Runtime-Distribution gibt es folgende Optionen: (i) Hub-and-Spoke: die Elemente ‚Ressource Interface' und ‚Connectivity' werden anwendungsseitig lokal an der zu integrierenden Applikation installiert. Die Elemente ‚Connectivity' (EAI-seitig), ‚Transformation' und ‚Process-Management' werden zentral gehalten, (ii) federated architecture: die Dienste ‚Connectitity', ‚Transformation' und ‚Process-Management' können beliebig plaziert werden. Darüber hinaus gibt es i.d.R. Konsolen zum Monitoring der realisierten Lösung hinsichtlich Performance, Verfügbarkeit und Zustand.

Ein solche Lösung spielt ihre Komponenten dort aus, wo es gilt, (Geschäfts-)Risiken zeitnah zu beurteilen. Batch-orientierte Informationswege sind hierzu nicht geeignet. Die LB Kiel hat deshalb für die Kommunikation eine Infrastruktur aus Broking- und Conversion- sowie Messaging-Ebene eingeführt. Daran angeschlossen sind u.a. auf der einen Seite die Software Kondor+ zur Erfassung der Geschäfte durch die Handelsbereiche sowie auf der anderen Seite die Module SEM von SAP zur Risikoüberwachung.[69] Die Broking-Ebene erhält aus Kondor+ via Messaging nicht mehr nur Files, sondern einen kontinuierlichen Strom abgeschlossener Handelsgeschäfte, den sie für das SAP-System nach unterschiedlichen betriebswirtschaftlichen Zuordnungen aufteilt. Nach den Regeln, die im Mapping hinterlegt werden können, selektiert der Broker die Daten und wandelt sie in die gewünschte Datenstruktur um. Der Broker entnimmt Stammdaten direkt aus einer Datenbank. Durch die Verknüpfung über die Broking und Conversionsebene ist sichergestellt, dass die Datenströme synchron laufen.

[68] vgl. K.Ring/N.Ward-Dutton: a.a.O., S. 48. Die Bandbreite der Konnektorentechnologie reicht vom reinen Abhören der TCP/IP-Datenströme, zum Anwendungsadapter für SAP bis zum XML-Adapter
[69] vgl. D.Blum: a.a.O., S. 29

Die Idee des „single point of control“ wird durch Repositories technisch sichergestellt.[70] Dies ist eine Stelle, an der zentrale Änderungen durchgeführt und an alle Systeme weitergeleitet werden. Ändern sich die Daten in den operativen Systemen, kann dennoch nachvollzogen werden, ob die entsprechenden Reports noch funktionieren bzw. es ist ersichtlich, welche betroffen sind und welche nicht. Grundlage hierfür ist die Semantik der Daten. „Real electronic communications requires a semantic understanding of the data being exchanged.“[71]

Die Verwaltung und Abbildung der Metadaten ist der zentrale Punkt für die Weiterentwicklung und den Betrieb von „Datensammelstellen“. Metadaten sind zwar in jedem EAI-Werkzeug vorhanden, wichtig ist aber, diese an einem Punkt zusammenzuführen und so die Basis für eine zentrale Dokumentation zu schaffen. Management, Metadatenhandling und Semantik sind die Kernkriterien, welche über die Güte und Leistungsfähigkeit einer Integrationslösung entscheiden.[72]

3.3 XML-based Application Integration (XAI)

In diesem Abschnitt soll die Frage geklärt werden, welche Rolle XML beim Datenaustausch i.w.S. einnehmen könnte. Entsprechend dem sog. ‚Plattformkonzept’ ist die Heterogenität der Anwendungen durch eine einheitliche Applikationsschnittstelle zu verbergen.[73] Einheitlich bedeutet hier, dass jeder Applikation die gleichen Mechanismen mit einheitlicher Aufrufsemantik zur Verfügung zu stellen sind. Die heterogenen Daten-Quellen werden über eine homogene Semantik eingebunden, was impliziert, dass von den Datenmodellen der Quellsystemen zu abstrahiert ist. Diese Entkopplung hat den Vorteil, dass Änderungen seitens der Quell-Systeme nicht notwendigerweise zu entsprechenden Anpassungen an den Schnittstellen der Empfangssystemen führen. Robustheit, Tragfähigkeit und Ausbaufähigkeit sind Anforderungen an solche Integrationsvorhaben.

XML kann zunächst überall dort eingesetzt werden, wo Daten ausgetauscht, interpretiert und ggfs. transformiert werden müssen. Die korrespondierende Positionierung von XML im OVUM-Modell zeigt die folgende Grafik:

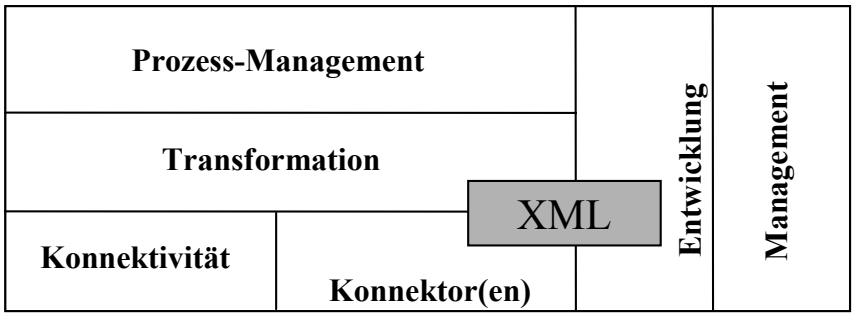


ABBILDUNG 15: POSITIONIERUNG VON XML

Entsprechend seiner ‚natürlichen’ Funktionalität als Beschreibungssprache ist XML als Austauschformat prädestiniert. Zum eigentlichen Datentransport kommt die Vali-

[70] vgl. D.S.Linthicum: Creating Enterprise Metadata Repositories for EAI and e-Business, in EAI Journal 1/2000, S. 13ff
[71] J.P.Morgenthal: XML and the New Integration Frontier, in EAI Journal 3/2000, S. 27
[72] vgl. G.Kolb: Jeder kann mit jedem, in IT Management 1/2001, S. 77
[73] vgl. M.Kolland/T.Mehner/K.-J.Kuhn: Software-Plattformen – Mehr als ein Schlagwort, in Wirtschaftsinformatik 1/1993, S. 23ff

dierung und die Transformation hinzu – und zwar bereits an der Schnittstelle. Mittels XSLT können auch komplexere Transformationen vorgenommen werden.

Neben Nutzdaten (hier: Geschäftsdaten) beinhaltet ein XML-Dokument immer auch Strukturinformationen in Form von DTDs oder Schemata, die durch Ablage in einem zentralen Repository die Grundlage für das Schnittstellenmanagement sind. Repositories werden damit zu einem wichtigen Werkzeug, um ein gemeinsames Verständnis der abzubildenden Geschäftsprozesse zu bekommen – was ganz wesentlich ist für die zukünftige Pflege/Erweiterung der Schnittstellen.

Mittels XML erfolgt ein semantisches (Tag-)Mapping, wodurch Entwickler sich auf die Modellierung und Implementierung von Geschäftsobjekten konzentrieren können. Die Wartbarkeit von Software wird zudem erleichtert, weil Schnittstellen ohne Beeinträchtigung von Feldern und Strukturen erweiterbar sind. Mit der Entwicklung eines eigenen API (SAX oder DOM) gesellt sich zum standardisierten Datenaustauschformat ein Verfahren, das den Zugriff auf Applikationsebene erlaubt. Die folgende Abbildung skizziert den workflow eines „dokumentenbasierten“ Schnittstellenmanagements, wie er innerhalb des Projektes „Generische Schnittstelle“ der WestLB umgesetzt wurde:[74]

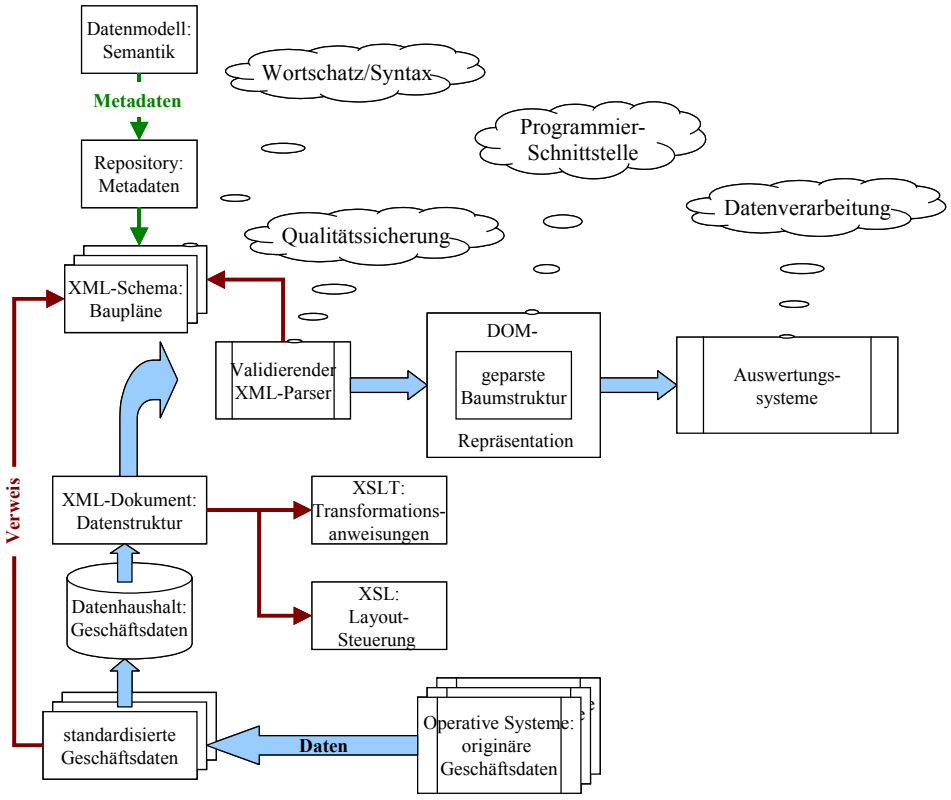


ABBILDUNG 16: XML-VERARBEITUNG

[74] vgl. H.Siegert/H.Louis/M.Drabben: Konzernweites Informationsmanagement via XML, in Betriebswirtschaftliche Blätter, 11/2000, S. 11/516ff

Ersichtlich ist eine Schnittstellenarchitektur, die unter Nutzung von XML und verwandter Standards sowie des Internet Browers 5.0[75] verschiedene operative Systeme einbindet. Der integrative Charakter dieser Architektur wird spätestens durch den Einsatz verschiedener Datenquellen und/oder Backend-Anwendungen deutlich. Die grundsätzliche Voraussetzung für eine solche (intelligente) Infrastruktur ist ein Datenmodell, das die Gesamtheit aller operativen Daten vollständig und konsistent abbildet – zudem gegenüber Veränderungen operativer Daten (durch das Mittel der Abstraktion) weitestgehend stabil ist.

Die mit XML definierten Norm-Dokumente als Träger des Datenmodells sind deskriptiver Natur – sie spezifizieren einerseits die gewünschten Daten, andererseits können die Daten auch entsprechend dieser „Baupläne" validiert werden. Dadurch lassen sich Geschäftsdaten von jeder Anwendung weiterverarbeiten, sofern diese nur den definierten Wortschatz kennen. Dies ermöglicht einen Datenaustausch über Systemgrenzen hinweg.

Müssen aber Applikationen ‚intensiv' miteinander kommunieren (bspw. im Realtime-Umfeld), ist es erforderlich, ein zentrales Kommunikations- und Verteilsystem i.S. einer „Informationsdrehscheibe" einzurichten, an das die einzelnen Schnittstellenmodule der operativen Systeme angeschlossen werden. Jedes operative System hat dann nur noch eine bilaterale Kommunikationsschnittstelle zum zentralen Verteilsystem. Eine solche zentrale „Informationsdrehscheibe" löst alle Geschäfts- und Produktdaten (nebst Marktdaten), entsprechend dem beigelegten Bauplan, in einzelne Datenelemente auf und

- beschreibt,
- konvertiert,
- transportiert,
- prüft,
- speichert,
- verwaltet und
- überwacht

sie.

Um die Integrität und Konsistenz der Daten zu sichern, kann in einem weiteren Schritt ein zentraler Datenpool an dieses Verteilsystem angeschlossen werden, welcher die Daten vollständig, redundanzfrei und korrekt den jeweiligen Auswertungssystemen zur Verfügung stellt.

Das Zusammenwirken der drei Komponenten Schnittstellen-Modul, Verteilsystem, Datenpool führt zu folgenden Leistungsmerkmalen:

- Über das Datenmodell werden die relevanten Daten vollständig und konsistent abgebildet; es dient sowohl zum Datentransport als auch zur Datenspeicherung.
- Hinsichtich der anzubindenden operativen Systeme wird lediglich eine Schnittstelle zum Datenimport und/oder -export benötigt. Die Definition der Schnittstelle erfolgt über Meta-Daten, deren Wurzeln im Datenmodell liegen. Abgelegt sind diese Informationen in XML-Schemata oder DTDs.
- Für jeden übertragenen Datensatz werden Integritätsprüfungen durchgeführt; dies geschieht über validierende Parser.

[75] wird als Front-End für die Anzeige/Pflege der gelieferten Datensätze eingesetzt, weil diese Anwendung auf jedem Rechner der WestLB konzernweit zur Verfügung steht. Sog. „Internet-Technologien" (auch HTTP) stellen mithin eine kostengünstige Infrastruktur für die Kommunikation bereit

- Datenmodell und Schnittstellenmodul gewährleisten, dass die Daten der operativen Systeme ggfs. vollständig und konsistent abgelegt werden. Ein solcher Datenpool bildet die stabile und aktuelle Grundlage für jedwede Art der Datenbereitstellung dispositiver Systeme. Hier wäre zu klären, ob die Ablage entsprechend der Struktur des XML-Dokuments erfolgt oder – mittels von vorgeschalteten Werkzeugen – in einer konventionellen relationalen Datenbank.
  Die einfachste (und restriktivste) Möglichkeit XML-Dokumente persistent zu halten ist, sie als flache Dateien im Dateisystem zu speichern. Was fehlt sind Mehrbenutzerzugriff, Transaktionssicherheit oder Abfragen (Queries). Zur Speicherung von XML-Dokumenten in Datenbanken können relationale, objektorientierte oder hierarchische Systeme verwendet werden. Dabei ist die komplexe, vernetzte, fein-granulare Struktur von XML zu berücksichtigen. Während hierbei sog. Content-Management-Systeme und klassische XML-Server den Weg über ein abstraktes Modell von XML gehen, d.h. eine Art persistentes und meist erweitertes DOM (was sie für die Verwaltung von XML-Dokumenten prädestiniert) abbilden, empfiehlt sich für XML-Daten die direkte Abbildung und Speicherung auf Java-Objekte.[76]
- Alle Komponenten sind über ein Transfersystem verbunden, das die notwendigen Funktionen zum Datenaustausch zur Verfügung stellt. Mithin besteht die Aufgabe darin zu untersuchen, ob bspw. SOAP in der Lage ist, diese Verteilung vorzunehmen.

Letztlich bietet das Datenmodell durch seine standardisierten Geschäftselemente die Möglichkeit zu einer zentralen Aufbereitung von cash flows bzw. zur Bewertung von Finanzinstrumenten. An dieser Stelle hört allerdings der Anwendungsbereich von XML auf: XML sollte nicht für Berechnungszwecke (via XSLT) missbraucht werden, dafür gibt es geeignetere Sprachen.

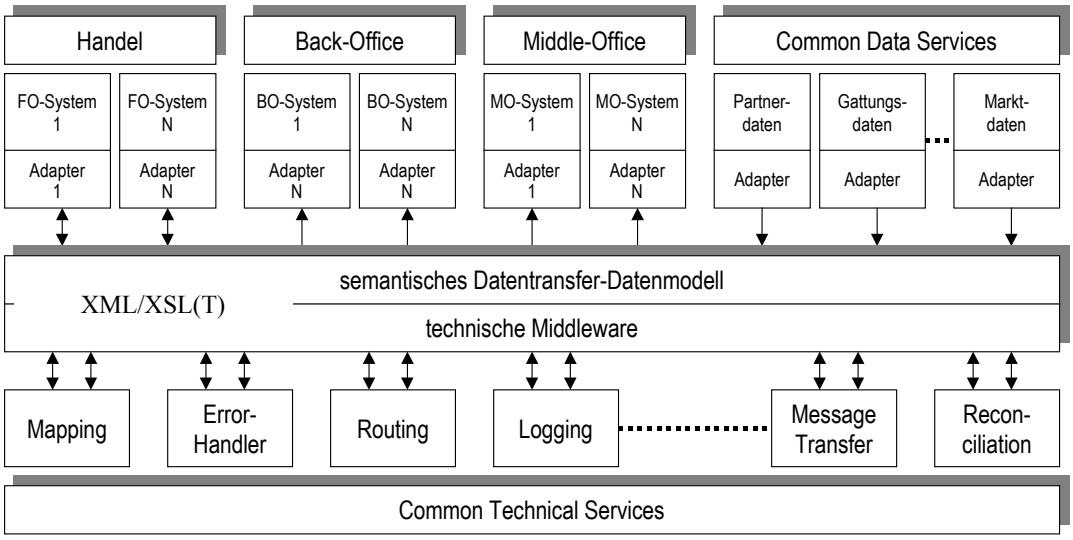
3.4 Busarchitektur für Applikationsschnittstellen im Real Time-Umfeld

Implementierungsaufwand und –zeit fällt insbesonders immer dann an, wenn neue Systeme (bzw. neue Produkte/Geschäfte) einzubinden sind, weil vom Front- bis Backoffice eine Kette von Schnittstellen geschaffen werden muss, damit die neue Funktionalitäten mit den bereits bestehenden Anwendungen kooperieren können.

Die DG Bank hat mittels ‚TIGRA' (Trading Room Integration Architecture) eine standardisierte Busarchitektur geschaffen mit der Zielsetzung, den oben geschilderten Integrationsaufwand sowie die Integrationszeit erheblich zu reduzieren.[77] Diese Schnittstellen-Architektur lässt sich wie folgt beschreiben:

- die bisherigen Punkt-zu-Punkt-Verbindungen wurden durch eine Busarchitektur ersetzt, die Daten für mehrere Empfänger aufbereitet, so dass nicht für jeden Empfänger eine eigene Schnittstelle zu schreiben ist,
- proprietäre Programme wurden durch ein CORBA-Framework abgelöst,
- das manuell programmierte Datenmapping wurde durch eine automatische Konfiguration via XML/XSL ausgetauscht: In der ersten Phase wurden zwar diverse Message Broker evaluiert, da es hier jedoch Probleme bei der Konvertierung der Datenmodelle gab, wurde XSL für die Transformation der Daten eingesetzt. So wurden CORBA und XML zur Basistechnologie:

[76] vgl. F. Thelen: XML, Java und Persistenz, a.a.O., S. 65. Thelen nennt als Beispiel die POET Object Server Suite (OSS)
[77] TIGRA wurde speziell für den Wertpapierprozeß (Frontoffice-, Middleoffice- und Backoffice-Anwendungen) entwickelt; vgl. hierzu J. Büchler: a.a.O., in IT-FOKUS 4/2001, S. 20ff



Erläuterung

- Common Data Services verwalten Referenzdaten zentral für alle Handelssysteme
- Handelssysteme (i) empfangen diese Referenzdaten und die Handelsdaten anderer Systeme
- Handelssysteme (ii) verteilen lokale Handelsgeschäfte über ihren Adapter
- Back- und Middleoffice-Systeme empfangen alle Handelsdaten über eine standardisierte Schnittstelle
- generische Schnittstelle entkoppelt die beteiligten Systeme mittels einheitlichem Datentransfer-Datenmodell auf Basis XML

Abbildung 17: Trading Room Integration Architecture (DG Bank)

Quelle: in Anlehnung an J. Büchler: Pack den Tigra in den Tank, in IT-FOKUS 4/2001, S. 20

Der Nutzen dieser Architektur liegt in der

- Reduktion von Integrationskosten. Die DG Bank geht davon aus, dass
  - 40 % der dv-Entwicklungkosten eingespart werden, weil u.a. 90 % der Software der ersten Schnittstelle wiederverwenden lässt
  - die Entwicklungszeit reduziert sich um 30 %, was insbesondere durch den Einsatz von XML/XSL zurückzuführen ist (mit XML/XSL wird das gesamte Datenmapping und der Transfer zwischen den Systemen konfiguriert anstatt programmiert)
- Bankproduke, die bisher wegen der zu erwartenden Realisierungskosten manuell bearbeitet wurden, können jetzt dv-technisch transferiert werden – was zur notwendigen Flexibilität der Technik gegenüber neuer Produkte/Geschäftsanforderungen führt
- indem alle Daten auf ein Standardformat konvertiert werden, läuft der Datentransfer wesentlich transparenter ab: Transferprobleme oder falsche Dateneingaben werden direkt an der Schnittstelle erkannt
- durch die Verwendung einer einheitlichen Technologie bei neuen Schnittstellen wird die IT-Landschaft homogener, was wiederum eine größere Stabilität im RZ-Betrieb bedeutet.

Jede Handelsapplikation wird über Adapter an die Schnittstelle angeschlossen. Ein Framework gibt für jeden Adapter den Integrationskern vor, wodurch sichergestellt ist, dass die Lösung dem Standard entsprechend programmiert wird.

Die Architektur basiert auf die Technologiekombination CORBA (die allerdings noch nicht optimal im Application-Server-Umfeld einsetzbar ist), XML und XSL. Die kommende CORBA –3– Spezifikation wird Enterprise Java Beans und Applica-

tion-Server-Technologie integrieren, wodurch der Weg in Richtung Application-Server-Technologie vorgezeichnet ist.

4 XML-Techniken und Werkzeugunterstützung

Nachdem im letzten Kapitel eine logische Sicht gewählt wurde, um die Problematik der Integration von Anwendungen zu strukturieren, folgt nun die Sicht auf die ‚physischen' Zusammenhänge, wobei ein architekturbasierter Werkzeugansatz gewählt wurde – haben sich doch in den letzten Jahren viele Einzeltechnologien und Standardisierungsansätze zu erfolgversprechenden Lösungen zusammengefunden („Applikations-Server"). Die Rolle von XML wird in diesem Kontext nachfolgend dargestellt.[78]

4.1 Applikations-Server

Applikations-Server kombinieren in einer 3-Schichten-Anordnung „neue" Standards wie HTTP, CORBA-IIOP oder XML (SOAP) in einer aufeinander abgestimmten, integrierten Umgebung, um heterogene Clients/Backends untereinander zu verbinden.[79] Im Bankenumfeld geht es dabei um

- die Interpretation der Geschäftsprozesse für das Backoffice, und
- die Synchronisierung transaktionsorientierter Front-Office-Systeme mit der Geschwindigkeit und Verarbeitungslogik der Legacy-Systeme (oft batchorientiert)[80].

Grundlage vieler Applikationsserver ist die „Java 2 Platform, Enterprise Edition" (J2EE). Java-Applikations-Server entwickeln sich zu einer der wichtigsten Technologie- und (komponentenbasierten) Integrationsarchitektur, indem sie sich an den Spezifikationen der Java 2 Enterprise Edition (Version 1.3 steht kur vor der Freigabe) orientieren. Geboten werden (i) Werkzeuge, (ii) skalierbare Ablaufumgbung für Java-Komponenten und (iii) Schnittstellen für diverse Integrationsaufgaben; ermöglicht werden damit (i) die einheitliche Verwendung einer Programmiersprache, (ii) eine Web-Unterstützung, (iii) umfangreiche Bibliotheken und (iv) Java-Frameworks (z.B. JWAM).

Anbieter wie IBM, Bea Systems, Sun/Iplanet, Iona, Borland, Oracle und Silverstream nutzen das Komponentenmodell Enterprise Javabeans (EJB) als Kern von J2EE. Wichtige J2EE-Technologien sind u.a.

- Java Naming und Directory Interface (JNDI)
- Java Database Connection (JDBC)
- Java Servlets
- Javaserver Pages (JSP)
- Java Transaction API (JTA)
- Java Transaction Service (JTS)
- Java Messaging Service (JMS)
- Remote Method Invocation (RMI).

Derzeit laufen Arbeiten für

- die Steuerung von Geschäftsprozessen über ein Control-Flow-Framework und der (bisher rudimentären) XML-Unterstützung

[78] eine gute Einführung in die verschiedenen Aspekte von XML bieten H. Behme/S. Mintert: XML in der Praxis, München 2000

[79] vgl. zum Stand der (durchaus auch kritischen) Diskussion S. Alexander: Praxis und Trends bei EJB-Servern, in Computerwoche 16/2001, S. 20-21; einen guten Einstieg in dasThema J2EE-Applikationsserver bieten auch J. Frey/A. Lobeck: Weg von der Insel, in IT-FOKUS 4/2001, S. 32-38

[80] die Firma FERNBACH, die Enterprise JavaBeans Technologie einsetzt, synchronisiert mittels eines ‚Data Integration Layer'; vgl. FERNBACH (Hrsg.): Orientierung im Datenmeer, o.O. 2001, S. 14f

- Messaging mittels der neuen „Message Driven Beans“ (MDS), um asynchrone Methodenaufrufe mit Hilfe von JMS-APIs zu ermöglichen. Hiernach stellen Clients ihre Nachricht in eine Warteschlange, für die der EJB-Server als Subscriber registriert ist; die MDS arbeiten die Requests ab und rufen dabei die On-Message-Methode auf. Der Vorteil liegt darin, dass die Clients dabei nicht wie bisher bei synchronen Verbindungen auf Antwort warten müssen. Der EJB-Server könnte damit künftig als Basisplattform vergleichbarer Messaging-Produkte für EAI fungieren
- die Entwicklung einer J2EE-Spezifikation „Java Connectors Architecture“ (JCA) für einen Standardadapter. Sie verspricht konzeptionell eine JDBC vergleichbare standardisierte Integration bspw. von SAP R/3, Legacy-Anwendungen oder CRM-Produkten (deren Erfolg allerdings von der Unterstützung durch die Standardsoftwarehersteller abhängt, da sie eine JCA-konforme API publizieren müssen)[81]
- Web-Services, die eine nachrichtenbasierte, echtzeitnahe Kommunikation darstellen, über den Programme/Komponenten mit Hilfe von Standardprotokollen und festgelegten Nachrichtenformaten untereinander bestimmte Dienste (z.B. über das Internet) veröffentlichen und nutzen können. Zwar folgt J2EE zunächst einem ähnlichen Konzept, indem JNDI ein Directory mit Business Services bietet, während RMI/EJB und JMS für den Aufruf solcher Dienste genutzt werden könnten. Der Unterschied ist aber der, dass dieser Weg nur zwischen Java-Programmen funktioniert, während Web-Services von einem beliebigen Client über SOAP erreichbar sein werden.

Derzeit liegt ein Vorschlag für ein „Java API for XML based Remote Procedure Call“ (JAX RPC) vor. Kombiniert mit dem Vorschlag eines „Java API for XML Registries“ (JAXR) und dem „Java API for XML Messaging“ (JAXM) sind es die ersten Schritte, um J2EE Web-Services zu spezifizieren. JAXR soll ein Standard-API liefern, über das Java-Anwendungen mit neuen Initiativen für den Aufbau einer Web-Services-Infrastruktur (wie bspw. UDDI und ebXML) kommunizieren können. JAXM definiert eine Java-API, die speziell auf ebXML abgestimmt ist (ggfs. als Nachfolger für EDI-Messaging). JAXM soll wie JMS genutzt werden und den automatisierten Empfang und das Versenden von XML-Nachrichten (ebXML-Dokumenten) über HTTP erlauben. Die Kombination aus JAXR für die Service-Suche und JAXM für asynchrone Aufrufe könnten J2EE-Entwickler in die Lage versetzen, Anwendungen für Web-Services zu erstellen, die wie traditionelle Message-oriented Middleware (MOM) arbeiten. JAX RPC wird hierbei für die standardisierte Interoperabilität zwischen SOAP und den neuen W3C XP-Protokoll sorgen. Vergleichbar mit EJB, wird es einem Rechner ermöglichen, Methoden und Prozeduren synchron über das Internet aufzurufen und XML-Dokumente über HTTP zu verschicken, steht doch mit JAX-RPC eine Schnittstelle zur Verfügung, mit der sich die Methoden einer sog. „Stateless Session Bean“ als synchrone Prozedur von einem beliebigen Client (VB, Perl, C#, etc.) aufrufen lassen. Ebenso wird jede beliebige verteilte Anwendungsplattform, die einen Web-Service anbietet, über JAXR zu finden und über JAX-RPC aufrufbar sein.[82]

Die Java 2 Enterprise Edition bietet zusammenfassend eine Plattform für die Entwicklung mehrschichtiger Anwendungen:

[81] mit „Comporsys Connector“ hat die Firma ‚Comporsys Hansa’ ein Produkt für J2EE-Application-Server vorgestellt, das die Integration von CICS-Anwendungen in EJBs, JSPs und Java-Servlets verspricht

[82] Sun bietet „Open Net Environment“ (ONE) als Bauplan für Web-Dienste auf Basis von Java und XML an; vgl. o.Verf.: Suns ONE setzt ganz auf Java und XML, in Computerwoche 7/2001, S. 32

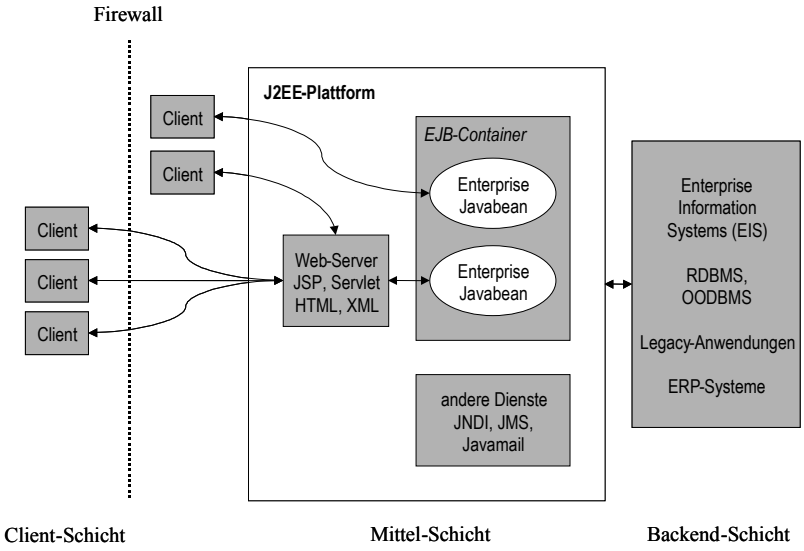


ABBILDUNG 18: INTEGRATIONSPLATTFORM J2EE

Quelle: S. Alexander: Praxis und Trends bei EJB-Servern, in Computerwoche 16/2001, S. 21

4.2 XML – Rückgrat einer Software-Architektur

XML ist hinsichtlich der Frage des Einsatzes von Werkzeugen zunächst eine plattform-neutrale Technologie, d.h. eine breite Palette von Sprachen, Parsern und Servern kommen grundsätzlich in Betracht:

- Tools für die Arbeit mit XML gibt es für Perl, C++, Java, JavaScript und jede COM-fähige Sprache,
- der Internet Explorer (ab Version 5.0) hat bereits einige Möglichkeiten zur Verarbeitung mit XML-Dokumenten eingebaut,
- XML-Tools tauchen mittlerweile auch zunehmend in großen relationalen Datenbanken auf, genauso wie auf Web- und Applikationsservern.

Von besonderem Interesse sind in den Finanzinstituten Werkzeuge, die unter einem Mainframe wie OS/390 direkt einsetzbar sind und/oder Architekturen, die die Nutzung gängiger Werkzeuge in Kooperation mit der OS/390-Welt erlauben.

Im Dezember l.J. hat IBM das „eServer Modell z900“, zusammen mit einer neuen Version seines Speichersystems „Shark“ auf den Markt gebracht. Die z-Modelle der eServer-Reihe sind die Nachfolger der bisherigen OS/390-Rechner. Als wichtigste technische Verbesserungen werden 3 Systemelemente hervorgehoben, die alle auf eine erhebliche Geschwindigkeitssteigerung (was insbesonders hinsichtlich einer verbesserten ‚XML-Tauglichkeit’ erwähnenswert ist) ausgelegt sind:[83]

1. Mit einer 64-Bit Datenverarbeitungsbreite können 4 Milliarden mal so viel Daten im Arbeitsspeicher direkt, also ohne teilweise Auslagerung auf Festplatten, adressiert werden als es bisher mit 4 Gigabyte (4 Mrd. Byte) möglich war. Da Daten wesentlich schneller in Arbeitsspeicher als auf Festplatten geschrieben werden können, bedeutet dies eine enorme Geschwindigkeitssteigerung bei großen Datenmengen
2. Die virtuelle Festplatteneinteilung (partioning) erlaubt die Unterbringung einer Reihe von Anwendungen auf einem Server

[83] vgl. o.Verf.: IBM liefert ihr neues Flaggschiff aus, in Handelsblatt v. 18.12.2000 sowie o.Verf.: OS/390 ist tot – lang lebe zOS, in Computerwoche 16/2001, S. 32

3. IBM arbeitet hier mit Linux-Servern unter dem Systemdach des neuen Mainframe[84], dessen traditionelle Vorteile bei der gesicherten Datenübertragung nach außen liegen.

Applikations-Server bilden die mittlere Schicht einer 3-Schichten-Architektur (siehe obige Abbildung); sie erfüllen eine ganze Reihe von technischen Basisdiensten wie Kommunikation zum Client, Transaktionsmanagement, Security und optimierter Datenbankzugriff. Darüber hinaus unterstützen sie Stabilität, Performanz und Skalierbarkeit von Anwendungen durch weitere Dienste wie Fail Over, Load Balancing und Connection Pooling. Zusätzlich können professionelle Werkzeuge eingebunden werden, die für das System Management im produktiven Betrieb erforderlich sind – und sie bieten verschiedene Möglichkeiten für die Kopplung einerseits mit dem Internet, andererseits mit Legacy-Anwendungen an.[85]

Der Vorteil liegt dabei insbesonders darin, dass in einer solchen Architektur weder die Programme noch die Daten zum Client zu transportieren sind. Hier kommen die Vorteile der Objekttechnologie voll zum Tragen, da Geschäftsobjekte nicht in den Clients „gefesselt" sind, sondern als wiederverwendbare Komponenten zentral zur Verfügung stehen:

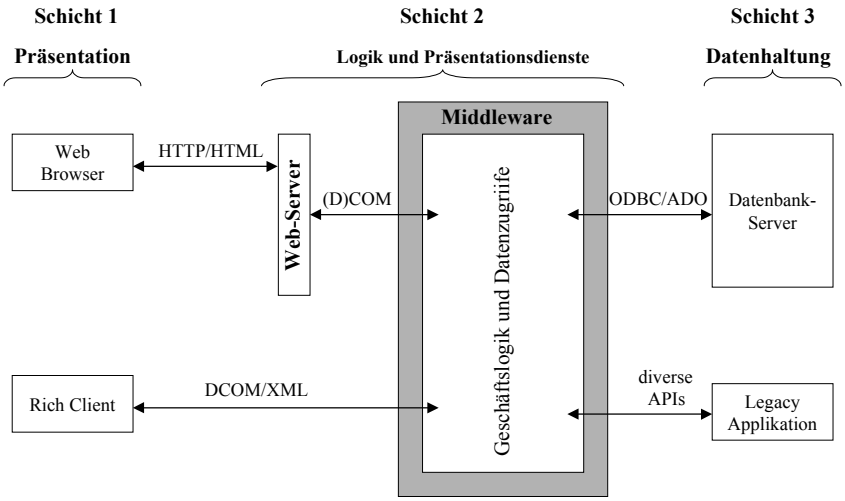


ABBILDUNG 19: 3-SCHICHTEN-ARCHITEKTUR

Quelle: in Anlehnung an R. Hubert/B. Hullunder/M. Buchheit: Applikations-Server öffnen die Unternehmens-IT für das Internet, in JavaSPEKTRUM 1/2001, S. 56 und A. Jung: XML – Schlüsseltechnologie für Softwarearchitekturen, in OBJECTspektrum 1/2001, S. 71

Welche Rolle spielt nun XML in einer klassischen 3-Schichten-Architektur? Da wäre zunächst der Client-Bereich - dort werden i.d.R. XML-Daten (u.a.) nach HTML aufbereitet. Daneben gibt es den Datenbankbereich; hier bieten zunehmend die Hersteller XML-basierte Schnittstellen an. Als drittes gibt es den Bereich Datenaustausch mit fremden Systemen; hier spielen Ansätze wie „BizTalk", „WebMethods" oder der „SAP Business Connector" eine Rolle.

In der klassischen Drei-Schichten-Architektur kommt XML bisher folglich nur punktuell (d.h. an den Schnittstellen) zum Einsatz – der Bereich Middleware (das

[84] der Begriff „Mainframe" bezeichnet im hier verstandenen Sinne sowohl einen Rechnertyp als auch die damit verbundene Softwaretechnik (u.a. Cobol, PL/1, CICS, VMS, VSAM). Das ist allerdings nicht so zu verstehen, dass moderne Technologien (Java, Application Server, Web-Technik) nicht genauso genutzt werden könnten wie mit Unix- oder NT-Rechnern!

[85] vgl. R.Hubert/B.Hullunder/M.Buchheit: Applikations-Server öffnen die Unternehmens-IT für das Internet, in JavaSPETRUM 1/2001, S. 55ff

Innere der zweiten Schicht) fehlt vollständig. Das führt notwendigerweise zu einem Technologiebruch, weil XML-Daten ausgepackt und auf die Funktionen des jeweiligen Systems umgesetzt werden müssen, während die Ergebnisse wieder in XML verpackt werden. Die Verarbeitung selbst erfolgt nach wie vor mit den herkömmlichen Mechanismen der Middleware.[86]

JUNG hält es vielmehr für konsequent, XML auch innerhalb der Middleware durchgängig einzusetzen: anstatt die XML-Daten an den jeweiligen Grenzen zu transformieren, werden sie ‚einfach' an die betroffenen Komponenten weitergereicht.[87] Die Zustandsverwaltung (einschliesslich der Manipulation) erfolgt nach diesem Modell im XML-Dokument und mit XML-Technologien, wodurch es u.a. gelingt, die Schichtenbildung innerhalb der zweiten Schicht deutlicher herauszubilden und klar nach Zuständigkeiten und Verantwortlichkeiten zu gliedern. Diese Vorgehensweise ist grundsätzlich auf alle komponentenorientierten Middleware-Technologien anwendbar.[88]

Die praktische Auswirkung eines solchen ‚Datenstrom-Modells'[89] liegt darin, dass nun die XML-Daten vom Frontend durch die Middleware bis hin zum Zugriff auf die Datenhaltung – ohne Unterbrechung – fließen: XML übernimmt dabei die Funktion eines Datenbusses. Die Parameterlisten der funktionalen Schnittstellen reduzieren sich durch XML-basierte Schnittstellen im Wesentlichen auf die Übergabe eines XML-Dokumentes bzw. auf die Spezifikation der Struktur (z.B. als XML-Schema):[90]

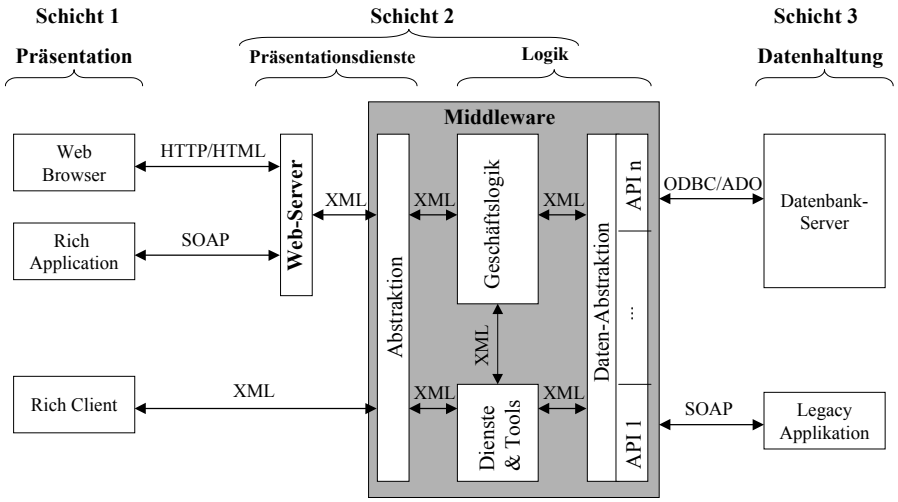


ABBILDUNG 20: XML ALS DATENBUS

Quelle: A. Jung: XML – Schlüsseltechnologie für Softwarearchitekturen, in OBJEKTspektrum 1/2001, S. 72

4.3 Architekturbasierter Werkzeugeinsatz

Die technische Grundlage für Applikations-Server sind verteilte Objekttechnologien und die entsprechenden Standards. Das Enterprise JavaBeans (EJB-) Komponenten-

[86] vgl. A. Jung: XML – Schlüsseltechnologie für Softwarearchitekturen, in OBJEKTspektrum 1/2001, S. 71. Der Autor stellt sich zu Recht die Frage, „ ...ist es wirklich notwendig, XML auf Objekte und Komponenten abzubilden, indem dort der Zustand (synonym mit Status – Anm. d.Verf.) gesetzt, Funktion aufgerufen und Parameter übergeben werden? Nur, um hinterher das Ergebnis wieder aus dem Zustand und den Funktionswerten herauszulesen und zurück nach XML zu transformieren?" (A. Jung: a.a.O., S. 71)

[87] ders.: a.a.O., S. 72

[88] mitunter wird auch der Begriff „COMWare" in diesem Zusammenhang genannt; vgl. http://www.objectwatch.com

[89] zu den Auswirkungen auf den Softwareentwicklungsprozess vgl. A. Jung: a.a.O., S. 72f

[90] vgl. hierzu ausführlich R. Westphal: Strong Tagging als Ausweg aus der Interface-Versionshölle, in OBJECTspektrum 4/2000, S. 60ff

modell von Sun Microsystems kann als Teil einer umfangreichen Plattform Java2EnterpriseEdition (J2EE) gesehen werden. J2EE integriert neben EJB diverse Internet-zentrische Dienste wie Servlets, Java, ServerPages, Java Messaging sowie XML-Funktionalitäten – insgesamt ist sie verantwortlich für das homogene Zusammenspiel der verschiedenen technischen Basisdienste und gibt den architektonischen Rahmen für ihre Benutzung vor.

Mit zwei neuen Programmierschnittstellen verknüpft Sun Microsystems Java enger mit XML. Konkret handelt es sich um die APIs „Java API for XML Messaging“ (JAXM) und „Java API for XML Parsing“ (JAXP)[91]. Zusammen mit dem noch in der Pipeline befindlichen „Java API for XML Data Binding“ (JAXB) stellen sie laut Sun den Kern der XML-Unterstützung der Java2-Plattform dar. JAXB stehe kurz vor der Vollendung durch das Java-Weiterentwicklungsverfahren „Java Community Process“. Alle drei APIs sollen standardmäßig in die nächsten Java-Versionen (J2EE, J2SE) implementiert werden. Die beiden veröffentlichten APIs sind über die „Java Developer Connection“ (http://java.sun.com/jdc) zu beziehen.[92]

Für die offene Welt (also jenseits von Microsoft) scheint das EJB-Komponentenmodell derzeit ein favorisierter Ansatz zu sein. So gibt es mittlerweile mehr als 40 Implementierungen, die den EJB-Standard umsetzen – allerdings sind diese Implementierungen hochgradig unterschiedlich.[93]

Unbestritten ist, dass XML ein anerkannter Standard für den Datenaustausch ist. Es bahnt sich aber Streit an, welche Infrastruktur sich bzgl. XML durchsetzen wird. Sun und deren Partner (u.a. OASIS und die United Nations) arbeiten an „ebXML“. Microsoft konkurriert dazu mit dem „BizTalk Server 2000“. Dieser erlaubt u.a. die Verbindung von XML-basierten E-Marktplätzen und die Integration von „Backend“-Systemen. Welche dieser beiden Spezifikationen sich am Markt durchsetzen wird, ist noch nicht klar.

Damit zeigt sich, dass sich die Auswahl eines geeigneten Servers für ein Projektvorhaben unbedingt an den Rahmenbedingungen für das Projekt orientieren muss; zu empfehlen ist folgende allgemeine Vorgehensweise:

- Je nach Schwerpunkt der geplanten Anwendung bietet sich eine erste Grobauswahl der Produkthersteller an
- Eine Priorisierung der dann in Frage kommenden Produkte lässt sich am besten durch einen repräsentativen Prototypen ermitteln. Angesichts der hohen Dynamik in diesem Markt gilt es dabei unbedingt zu beachten, dass sich die Produkte hinsichtlich Qualität und Funktionsumfang schnell ändern können.

Der Vorteil von architekturbasierten Werkzeugansätzen ist, dass mit ihnen die Risiken bei der Auswahl eines Applikationsservers minimiert werden, denn sie ermöglichen bereits im Systemdesign eine möglichst hohe Produktunabhängigkeit. Dies wiederum erlaubt den Austausch von Komponenten zu einem späteren Zeitpunkt bei relativ geringem Aufwand. Die Lösung besteht darin, Komponenten so zu bauen,

[91] implementiert ist DOM Level 1 als „Optional Package“
[92] unter http://www.java.sun.com/xml/download.html ist die Version 1.1 Early Access 2 des Java API für XML Processing verfügbar
[93] vgl. R.Hubert/B.Hullunder/M.Buchheit: a.a.O., S. 56. Sie untersuchten folgende Anbieter: Hersteller von CORBA Implementierungen, Hersteller relationaler Datenbanken, Hersteller objektorientierter Datenbanken und Hersteller objektrelationaler Tools sowie Neuentwicklungen von Java-Applikations-Servern (u.a. IBM). In WebSphere von IBM kommen eine Reihe von Basistechnologien zum Einsatz, die allerdings unabhängig voneinander entwickelt wurden. Ein interessanter Aspekt von WebSphere bildet die Mainframe-Connectivity

dass die Fachlogik und die Infrastruktur klar getrennt bleiben. Ein Architekturstil definiert, wie dies geschehen soll, und die darauf basierenden Werkzeuge können es weitgehend mit Hilfe von Code-Generierung und Komponentengenerierungs-Tools automatisch umsetzen.[94]

4.4 XML-Techniken im Datenstrom

Die folgende Abbildung zeigt die bedeutendsten XML-Spezifikationen und ihren Anwendungsbereich. Für die Zwecke des Datenaustausches kommen vor allen Dingen XML-Schemata in Betracht und die ParserAPI zwecks Validierung gegen das XML-Schema; XML-Transformationen sind nur dann von Interesse, falls Manipulationen im XML-Dokument vorzunehmen sind.

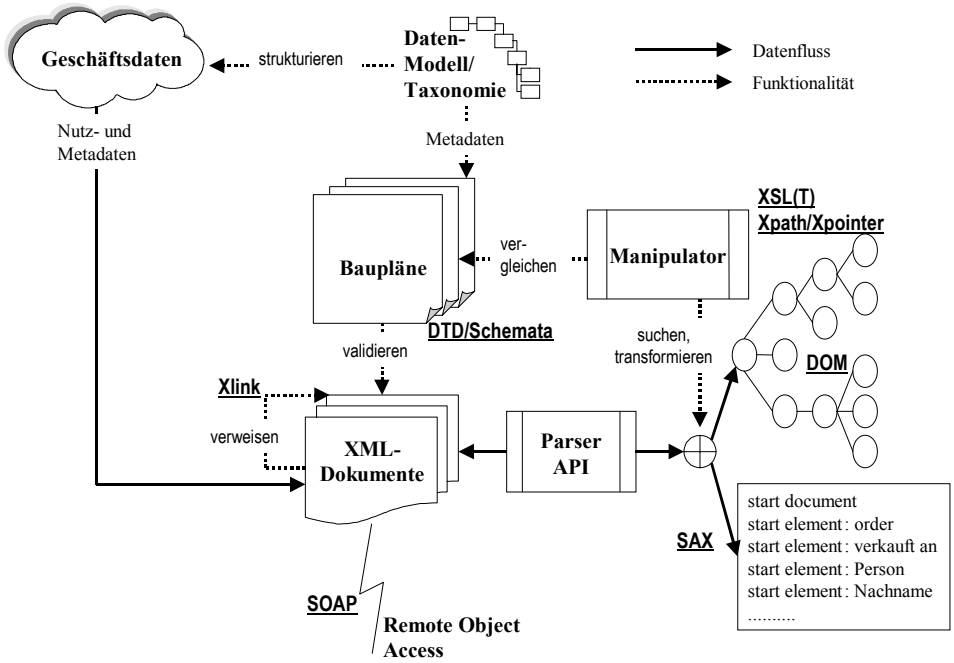


ABBILDUNG 21: XML-TECHNOLOGIEN

Dem Datenstrom folgend sind folgende XML-Technologien von Bedeutung:

1. Parser APIs kommen als Document Object Model (DOM) und SAX (Simple API for XML) in Betracht. Das DOM ist eine Spezifikation des W3C. Die erste Version dieses Standards (DOM Level 1) wurde im Oktober 1998 verabschiedet. Die zweite Version (DOM Level 2, die insbesondere die Unterstützung von XML-Namespaces bringt) liegt zur Zeit als „Candidate Recommendation“ vor – d.h. dass das W3C noch auf Implementierungs-Feedback wartet, bevor die Spezifikation in die nächste Phase „Proposed Recommendation“ kommt.

   DOM ist eine plattform- und sprachunabhängige Spezifikation, folglich keine Implementierung. Daneben gibt es Sprachanbindungen (language bindings), die das API in der jeweiligen Sprache definieren. So besteht die Sprachanbindung für Java aus einer Sammlung von Java-Interfaces, die von konkreten DOM-Implementierungen umzusetzen sind. Mittlerweile gibt es zahlreiche DOM-Implementierungen, z.B. Xerces von Apache.

[94] vgl. R.Hubert/B.Hullunder/M.Buchheit: a.a.O., S. 57

Ein DOM-Parser (genauer: ein XML-Parser, der DOM unterstützt) parst ein XML-Dokument vollständig, bevor er im Hauptspeicher ein komplettes Abbild davon in Form eines Baums präsentiert. Dieser Baum besteht aus Objekten vom Typ ‚Node' bzw. dessen Sub-Typen. Über die DOM-Interfaces kann eine Applikation über den Baum navigieren und einzelne Knoten verändern.[95]

Das Simple API for XML (SAX) 1.0 ist im Mai 1998 erschienen; SAX 2.0 gibt es seit Mai 2000 (es bringt insbesonders die Unterstützung von XML-Namespaces). Verfügbar ist es als „Optional Package" JAXP für die Java-Plattform J2SE und J2EE.

Wie DOM ist auch SAX eine plattform- und sprachunabhängige Spezifikation mit Anbindungen an verschiedene Programmiersprachen. Die meisten XML-Parser unterstützen sowohl DOM als auch SAX (so auch Xerces von Apache).

SAX ist rein ereignisbasiert – was SAX das Attribut ‚einfach' verdankt. Ein SAX-Parser (genauer: ein XML-Parser der SAX unterstützt) erzeugt während des Parsens eine Sequenz von Ereignissen, die der hierarchischen Struktur der gelesenen XML-Instanz entspricht. Nach Beendigung des Parsens steht ausschließlich das zur Verfügung, was bei dem Bearbeiten der Ereignisse selbst aufgebaut wurde. Die Vorteile von SAX liegen in seiner Effizienz und seinem geringen Speicherverbrauch, sodass auch sehr große XML-Dokumente verarbeitet werden können. Im Gegensatz zu DOM erlaubt SAX nur das Lesen über die Ereignis-Callbacks des Parsers. Nachteilig ist auch der fehlende Kontext in einem Callback, da tiefere Teile noch nicht geparst, höhere Teile vielleicht schon wieder vergessen sind. SAX wird zum Aufbau spezieller, eigener Datenstrukturen empfohlen, die nach Beendigung des Parsens für die weitere Bearbeitung zur Verfügung stehen.[96]

Während DOM und SAX mit abstrakten Begriffen wie ‚Element' oder ‚Node' arbeiten, ist es das Ziel des „XML Data Binding", welches als JSR-31 im Rahmen des „Java Community Process" erarbeitet wird, dass eine Applikation mit den konkreten Begriffen ihrer Problemdomäne arbeitet.[97] Der Ansatz basiert darauf, dass ein konkretes XML-Schema (oder eine DTD) auf eine Menge von Java-Klassen abgebildet wird. Diese Java-Klassen stellen die Schnittstelle bereit, über die die XML-Objekte (im Speicher repräsentiert durch echte Java-Objekte) navigiert und manipuliert werden können:

[95] vgl. F. Thelen: XML, Java und Persistenz, in JavaSPEKTRUM 1/2001, S. 61ff. Der Autor zeigt bspw. auch das Vorgehen im Code, bevor über einen DOM-Baum navigiert werden kann
[96] vgl. F.Thelen: XML, Java und Persitenz, a.a.O., S. 62
[97] vgl. F.Thelen: XML, Java und Persistenz, a.a.O., S. 64

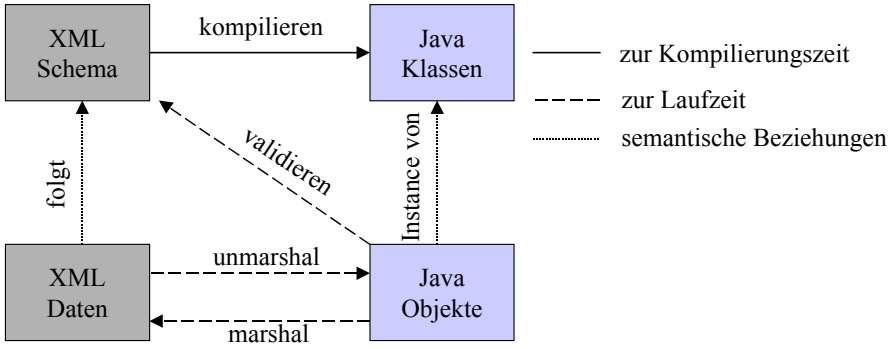


ABBILDUNG 22: XML DATA BINDING FÜR JAVA

Quelle: F. Thelen: XML, Java und Persistenz, in JavaSPEKTRUM 1/2001, S. 64

Die Implementierung besteht aus einem Compiler und einem Runtime-Framework. Der Compiler erzeugt aus einem XML-Schema (oder einer DTD) Java-Klassen; bei einem XML-Schema kann das vom Schema definierte Typsystem direkt auf Java abgebildet werden. Das Runtime-Framework erledigt das ‚Marshalling' und ‚Unmarshalling', also das Transformieren eines XML-Dokuments in die entsprechende Laufzeitrepräsentation (d.h. in die Java-Objekte) und umgekehrt. Manuelles Parsen, Kopieren, Formatieren und Validieren entfällt – das erledigt das Framework: Eine Applikation, die auf diesem Mechanismus aufsetzt, verwendet ausschließlich die erzeugten Java-Klassen und die Klassen des Frameworks.

Interessant ist das „XML Data Binding" für Java immer dann, wenn es gilt, XML als Format (Daten-, Transport-, Nachrichtenformat) oder Protokoll zu nutzen. Dies ist regelmässig dann der Fall, wenn sich z.B. 2 Applikationen auf ein bestimmtes XML-Format einigen müssen, um sich zu verständigen. Es ist daher zu empfehlen, den Austausch von Geschäftsdaten basierend auf XML-Schemata direkt mit den Begriffen der Problemdomäne (statt indirekt über DOM oder SAX) zu verarbeiten, genauer zu analysieren.

2. DTDs/Schemata: DTDs haben die Aufgabe, konkrete XML-Formate strukturell zu beschreiben. Der XML-Standard definiert, wie DTDs für XML auszusehen haben. Da dieses Konzept (wie XML insgesamt) an SGML angelehnt ist, eignen sich DTDs primär für die Verwaltung von Dokumenten i.e.S., nicht aber unbedingt auch für Daten. Da es jedoch für Daten extrem wichtig ist, Integritätsbedingungen und Typinformationen definieren zu können, muss nach alternativen Konzepten gesucht werden.

   Der aussichtsreichste Kandidat für einen solchen Standard ist derzeit XML Schema, das auch vom W3C erarbeitet wird. Er bietet alles, was man sich für ein effizientes Datenmanagement auf der Basis von XML wünscht:[98]
   - Trennung von Elementen und Typsystem
   - Simple Types und Complex Types
   - abgeleitete Typen (Vererbung, Polymorphie)
   - anonyme und abstrakte Typen/Elemente
   - Keys, Unique Keys, Referenzen.

[98] vgl. F. Thelen: XML, Java und Persistenz, a.a.O., S. 64

3. Xlink: Xlink spezifiziert die Möglichkeit von XML, Verweise (Links) auf externe Ressourcen zu ermöglichen. Insbesonders können Links u.a. dazu dienen, Beziehungen zwischen verwandten Dokumenten auszudrücken, die Navigation durch XML-Dokumente zu regeln oder dazu verwendet werden, um Nicht-XML-Daten (Graphiken, binäre Dateien) in ein Dokument zu integrieren.

4. Xpath/Xpointer: Die XML Pointer Language wird ein Adressierungsschema definieren, mit dessen Hilfe innerhalb eines Uniform Resource Identifier (URI) auf interne Strukturen eines Dokumentes zugegriffen werden kann (dabei setzt es auf Xpath auf). Xpointer könnte zukünftig dazu dienen, mittels Browser auf einen bestimmten Dokumentabschnitt zuzugreifen, ohne dass an dieser Stelle ein ‚Anker' eingefügt werden muss. Darüber hinaus ist das Verfahren flexibel und erlaubt die Formulierung von Bedingungen.

5. XSL/XSLT: Bei der Formatierung von Dokumenten geht es nicht nur darum, ein solches für die visuelle Darstellung im Web aufzubereiten. Wenn ein XML-Dokument für verschiedene Anwendungen genutzt werden soll, dann müssen aus denselben Ursprungsdateien unterschiedliche Ausgabeformate erzeugt werden. Die XSLT-Spezifikation konzentriert sich auf XML-Dokumenten-Transformation, die XSLF-Spezifikation befasst sich mit Formatierungsobjekten und XPath beschreibt, wie man bestimmte Knoten aus der hierarchischen XML-Struktur erreichen kann.

6. SOAP: Aus Entwicklersicht wäre es wünschenswert, einen durchgehenden objektorientierten Ansatz zu verfolgen, bei dem sich alle verteilten Dienste über Methodenaufrufe erreichen lassen:[99] Man spezifiziere eine XML-basierte Skript-Sprache für die Serialisierung von entfernten Methodenaufrufen und füge HTTP als Kommunikationsprotokoll hinzu. Dadurch lassen sich Webserver zu Object Request Broker[100] ausbauen. Infrastrukturen für verteilte Objekte/Komponenten folgen dieser Broker-Architektur: der Broker verbindet Client mit Server, über HTTP erfolgt die Kommunikation, über XML das Marshaling[101].

   SOAP ist eine solche plattformübergreifende Technologie, da es den einfachen Transport von Methodenaufrufen über HTTP-Verbindungen erlaubt. SOAP ist letztlich die Standardisierung einer bereits etablierten Vorgehensweise auf Basis von HTTP und XML: So spezifiziert SOAP ein XML-basiertes Kodierungsformat für Methodenaufrufe, Ergebnisse oder Fehlermeldungen. SOAP-Pakete repräsentieren demnach XML-Dokumente mit vorgegebenen XML-Elementen. Allerdings ist SOAP ein laufzeitintensiver Mechanismus, damit ist es für Echtzeitsysteme ungeeignet. Die Infrastruktur einer Realtime-Umgebung ist i.d.R. systemnah ausgelegt und entsprechend optimiert, während SOAP für die flexible Interoperabilität entworfen wurde.[102]

[99] vgl. A.Siffring: SOAP soll das Rückgrat von Web-Services bilden, in Computerwoche 50/2000, S. 73f

[100] objektorientierte Infrastruktur, die über eine zentrale Komponente (Broker) Kommunikationsverbindungen (wie HTTP) zwischen Clients und Server aufbaut. Hierunter erfolgt der Transport von Methodenaufrufen und ihrer Ergebnisse als ein Mechanismus, um die typisierten Nutzdaten in Form von Protokollpaketen mit XML zu übertragen

[101] Einpacken und Auspacken von Marshaling, also das Paketieren und Funktionsaufrufen sowie deren Argumenten für den Netzwerktransfer

[102] vgl. C.Vasters: a.a.O., S. 29. Einen interessanten Vergleich ‚SOAP vs. MIME' liefert Morgenthal; vgl. J.P. Morgenthal: Analyzing Approaches to XML Messaging, in EAI Journal 1/2001, S. 35ff

4.5 Werkzeugunterstützung

Zu den Firmen, die derzeit die XML-Liga anführen, gehört sicherlich IBM.[103] Auf der Technik-Web-Site (Alphaworks: http://www.alphaworks.ibm.com) finden sich viele Tools, angefangen von XML-Parser for Java bis hin zu Bean Maker, einem Programm, das aus einer DTD gleich die passenden XML-Beans erzeugt. Nachdem Microsoft XML als Kerntechnologie ausgerufen hat, will auch IBM dafür zahlreiche Tools anbieten. Diese reichen von Editoren und Schema-Übersetzern bis zum Toolkit für UDDI (Universal Description, Discovery and Integration) und einer Brücke zur Kommunikation mit SAP R/3.

Einige größere Projekte stehen im Zeichen von IBMs Bemühungen, eine Infrastruktur für Web-Services auf Basis des SOAP sowie der UDDI aufzubauen. Damit positioniert sich IBM explizit gegen Microsofts „Visual Studio .NET“, das d.J. auf den Markt kommen soll. Dies gilt vor allem für das Javabasierte „XML and Web Services Development Environment“.

IBM will Anforderungen, die darauf zielen die großen ERP-Pakete für die entstehenden Web-Dienste fit zu machen, mit der „XML Bridge for Java“ begegnen. Das Produkt ist in der Lage, im XML-Format eingehende Aufrufe von R/3-Funktionen in die nativen SAP-Formate zu übersetzen. Umgekehrt kann das R/3-System einen RFC oder BAPI-Aufruf an die XML-Brücke senden, die diesen dann als XML-Dokument darstellt und an das Zielsystem weiterleitet. Als Transportmechanismus für derartige XML-Daten sieht die Software SOAP über HTTP und „MQ Series“ vor. IBM positioniert die XML-Brücke sowohl für die Kopplung von R/3 mit Fremdsystemen als auch von R/3-Installationen untereinander. Aufgrund der Unterstützung von MQ Series sieht IBM auch die Möglichkeit, das ERP-System mit der hauseigenen Middleware zusammenzubringen.[104]

Der XML-Editor „Xeena“ prüft während der Eingabe von Daten ihre Konformität mit der DTD. Hierzu greift das Java-Programm auf den Parser „Xerces“ zurück. Xeena verfügt auch über eine Implementierung von XML Schema. Desweiteren gibt es von IBM ein Werkzeug namens „Xtransgen“ zur Umsetzung von Dokumenten von einer DTD auf eine andere.

Sun versucht mit einer Programmbibliothek Java ins Feld zu bringen. Der validierende XML-Parser ist DOM-konform und erlaubt das Einbinden von XML-Beans und unterstützt die XML-Namespaces[105].

[103] ein interessantes Szenario hinsichtlich des Einsatzes von IBM XML Tools, Visual Age for Java und WebSphere Application Server zeichnet IBM (Hrsg.): Integrated Scenario for Developing XML Applications Using the IBM XML Tools and Visual Age for Java, 1997/2000

[104] vgl. o.Verf.: IBM wetteifert mit Microsoft beim Angebot von XML-Tools, in Computerwoche 49/2000, S. 20

[105] weitere Hinweise zu XML-Werkzeugen unter www.wdvl.com

5 Case Study

Mittels einer Fall-Studie werden die bisherigen Gedanken nachfolgend in einem ganzheitlichen Rahmen dargestellt.

5.1 Vision: Lose gekoppelte Komponenten

In einer klassischen Client-Server-Architektur sind die beteiligten Komponenten durch zufällige und starre Strukturen verbunden. XML hingegen bietet einem Entwickler die Möglichkeiten, wirklich verteilte Systeme zu erstellen, die durch einen offenen Standard für selbstdokumentierende Daten zusammengehalten werden.

Clients, Browser oder andere Programme könnten ihre Anfragen zukünftig in Form von XML-Dokumenten an einen Server senden. Ein solches Dokument würde auch die Namen der Parameter und die entsprechenden Werte enthalten. Im Unterschied zum bisherigen Verfahren würde die Struktur der Anfrage durch ein genormtes Verfahren zur Laufzeit vom Server erfragt werden. Ein solches Verfahren dokumentiert nicht nur die Struktur der Anfragen, die der Server erlaubt, es würde dem Client auch erlauben, seine Anfrage vor der Versendung auf Korrektheit zu prüfen. Der Server könnte nach dem Erhalt der Anfrage auch eine Überprüfung der Anfrage durchführen.

Erhält der Server eine Anfrage, so kann er diese wie bisher abarbeiten oder er könnte sich Hilfe holen. So könnte XML dazu genutzt werden, die Kommunikationsart formal zu definieren. Da die von einem Server empfangenen Daten in einem solchen Fall auch XML-Dokumente wären, ist es für den ersten Server recht einfach, mehrere Dokumente miteinander zu verschmelzen oder ein Dokument in ein anderes Format zu transformieren, um so eine Anfrage zu befriedigen.

XML-Dokumente sind Daten, die relativ leicht manipuliert werden können. Das Übertragungsmedium für Informationen, in diesem Fall das XML-Dokument, macht keine Annahmen über die endgültige Verwendung der Daten. Sollte bekannt sein, dass ein Client HTML als Antwort erwartet, kann ein XML-Dokument entsprechend transformiert werden.

Da XML-Dokumente prinzipiell eine hierarchische Struktur haben, können auch nicht relationale Daten beschrieben werden; die meisten Datenbestände werden jedoch in relationalen Datenbanken vorliegen. Diese müssen dann in die Struktur eines XML-Dokumentes erst gegossen werden.[106]

Hat man sich für XML als Austauschformat entschieden, können Entwickler neue Komponenten erstellen oder bestehende Komponenten und Bibliotheken mit Routinen zur Manipulation von XML-Dokumenten nutzen.

Die Kopplung zwischen dem Server und einem Client ist ‚gelöst', da der Client die Struktur eines XML-Dokumentes selbst herausfinden kann. So können Applikationen erstellt werden, bei denen sich die Struktur der Dokumente im Laufe der Entwicklung ändern kann und deren Dokumente dennoch weiterhin verarbeitet werden können, ohne dass für jede Änderung der Struktur die Software angepasst werden muss. Es ist zu vermuten, dass eine Reihe von ‚Standard'-Tag-Mengen entwickelt

[106] eine Lösung für die Speicherung von XML-Dokumenten in DB2 beschreiben J. Cheng/J. Xu: IBM DB2 XML Extender, ICDE'00 Conference, San Diego, 2/2000

werden und Anwendungen die Struktur der Dokumente untersuchen werden, um sich so vor Fehlern durch Versionsänderungen zu schützen.

Im Ergebnis verwendet jedes Element der Verarbeitungskette in einem Netzwerk aus Servern, Clients und Anwendungen denselben Mechanismus für den Datenaustausch. Dieser Mechanismus ist beliebig erweiterbar und stützt sich auf die Erkennung von Dokumentstrukturen zur Laufzeit. Außerdem ist er auf jeder Plattform verfügbar, relativ leicht anzuwenden und in der Lage, Daten aus beliebigen Quellen oder von anderen Servern zu holen, um so Anfragen von Clients zu befriedigen. Das verändert das bisherige Client-Server-Modell zu einem echten verteilten System.

5.2 Aufgabenstellung

Aus den skizzierten Ausgangssituationen lassen sich folgende – von spezifischen Gegebenheiten abstrahierende – Aufgaben ableiten:

- bestehende Daten modellieren
- XML-Dokumente kontrolliert und dynamisch erzeugen
- XML-Dokumente ablegen
- XML-Dokumente reporten.

Im folgenden wird davon ausgegangen, dass ‚die' Daten bereits existieren. Es ist wichtig, diesen Punkt zu beachten, wenn ein Projekt mit Anwendung von XML geplant wird. Völlig unrealistisch wäre der Versuch, komplett auf XML umzusteigen und die bisherigen Methoden der Datenverwaltung zu vergessen.

5.2.1 Bestehende Daten modellieren

Die Erstellung eigener Baupläne ist für die Arbeit mit XML fundamental. Datenmodellierung wird benötigt, um via DTD bzw. Schemata XML Daten strukturiert abzubilden.

Ein Datenmodell bildet bekanntlich eine Beschreibung der Abläufe und Informationsflüsse in einer realen Organisation ab.[107] Diese Beschreibung ist unabhängig von konkreten informationstechnischen Systemen. Die Frage nach der Modellierung von Informationen hat also nicht ursächlich etwas mit XML zu tun, ist aber von größter Bedeutung: Ohne ein Modell hat man nur Daten, keine echten Informationen – es würde die Bedeutung der Daten (Semantik, Metadaten) fehlen.

Während statische Modelle sich auf die Beschreibung der zulässigen Zustände eines Systems beschränken (welche Typen von Objekten sind erlaubt, welche Eigenschaften haben diese Objekte und wie sehen die Beziehungen untereinander aus), dienen die dynamischen Modelle der Beschreibung des Datenflusses im System (Prozessmodelle, Workflow-Diagramme). Statische Modelle sind demnach von unmittelbarer Bedeutung für den Entwurf einer Datenhaltung, aus der heraus Informationen für die unterschiedlichsten Zwecke ausgewertet werden. Dynamische Modelle hingegen sind von direkter Bedeutung für den Entwurf von Nachrichten, die nur einem sehr speziellen Zweck dienen.

XML kann beide Arten von Daten beschreiben, wobei mit dem statischen Modell zu starten ist, weil hier die Grundlagen der Terminologie liegen. Darüber hinaus ist ein statisches Modell i.d.R. der stabilste und langlebigste Teil eines Systems.

[107] siehe im Einzelnen C. Finkelstein/P. Aiken: Building Corporate Portals with XML, New York u.a. 2000, S. 79ff

Die erste Aufgabe besteht darin, den Umfang und das Aufgabengebiet zu definieren; die *Strukturierung* ist durch Generalisierung und Abstraktion zu gewinnen. Im Einzelnen werden folgende Schritte empfohlen:[108]

- Identifizieren, benennen und definieren der Objekt(typen)
- Organisieren der Objekte in einer Klassenhierarchie (Taxonomie)
- Definieren der Relationen („Assoziationen" – nach UML-Jargon), Kardinalitäten und Einschränkungen
- Eigenschaften hinzufügen, um die Objekte mit erlaubten Werten auszudrücken.

Wesentlich ist, das Modell - als eine höhere Abstraktionsebene - frei von Einzelheiten zu halten, um es so verständlicher und allgemeingültiger zu machen.

Soll mittels XML der *Datenfluss* in einem System dargestellt werden, muß bekannt sein, wo Daten entstehen und wohin sie geleitet werden. Mögliche Techniken, die zu so einem dynamischen Modell führen sind:[109]

- Prozess- und Workflow-Modelle
- Datenfluss-Modelle (beschreiben Datenspeicher, Prozessoren und Datenflüsse)
- Object life histories (einzelne Objekte werden umfassend analysiert)
- Use cases (analysieren, welche Informationen zwischen Benutzer und System ausgetauscht werden)
- Objekt-Interaktions-Diagramme (analysieren den Nachrichtenaustausch zwischen den Objekten genauer als mit einem Datenfluss-Modell).

Durch Analyse der realen Welt und Repräsentation der so gefundenen Informationen in einem Modell sollte eine Basis für ein einheitliches Verständnis der Daten vorhanden sein. Der nächste Schritt besteht darin, XML-Dokumente so zu gestalten, dass die Modelle mit Leben gefüllt werden.

5.2.2 Bestehende Daten und Applikationen nutzen

Bestehende Daten als XML zu exportieren bedeutet zunächst, das Format genau festzulegen. Die Daten herkömmlicher Datenbanken exportieren Daten als relationale Datensätze. Die Extraktion von Daten aus bestehenden Systemen bringt demnach einige Probleme mit sich.[110] Die beste Lösung würde darin bestehen, die Daten in einem bekannten Format zu bekommen, d.h. einer XML-Darstellung der relationalen Struktur. Die müsste dann (noch) transformiert werden.

Der Vorteil in der Anwendung von XML zur Definition von XML-Dokumenten durch XML-Schemata liegt in der Automatisierung vieler Aufgaben. So kann ein Satz von Regeln erstellt werden, um aus einem Schema eine relationale Datenbank zu erstellen. Diese Regeln können verwendet werden, um einen Baum zu erzeugen, der eine Sammlung von Anweisungen zur Erstellung einer Datenbank enthält.[111]

XML und Datenbanken sind (noch) recht unterschiedliche Welten. Es gibt aber Methoden, diese Welten zu verknüpfen. Die Implementierungen von Datenbank-

[108] vgl. R. Anderson, u.a.: XML in der Praxis, Bonn 2000, S. 126ff
[109] vgl. R. Anderson, u.a.: a.a.O., S. 132ff
[110] vgl. hierzu im Einzelnen R. Anderson, u.a.: a.a.O., S. 454f
[111] die Abbildung bzw. Konvertierung von Daten, die einem Schema entsprechen, auf eine relationale Datenbank sowie die automatische Erstellung einer relationalen Datenbank aus einem Schema demonstrieren R. Anderson, u.a.: a.a.O., S. 458ff

Herstellern zeigen indes, dass diese bemüht sind, die Möglichkeiten von XML und Datenbanken zu vereinen.

5.2.3 XML-Dokumente kontrolliert und dynamisch erzeugen

Dynamisch generierte XML-Dokumente werden immer mehr zur Regel, wenn XML als Format für Dokumente benutzt wird. Angesichts der Datenmengen, die Unternehmen in relationalen Systemen liegen haben, werden Lösungen, die eine XML-Schicht über diesen Informationen platzieren, immer wichtiger und rentabler.[112]

In diesem Zusammenhang ist XML wie eine Datenstruktur zu betrachten (XML-Dokumente sind eben etwas anderes als Text-Dokumente): Ein XML-Dokument muss nicht in einer festen Form (Datei) existieren, vielmehr kann es im Bedarfsfall erzeugt werden – zum Beispiel von einem Webserver. Dann wäre es möglich, dass 2 Anfragen nach dem gleichen Dokument unterschiedliche Ergebnisse produzieren (bspw. weil neue Geschäftsdaten hinzugekommen sind).

Da relationale Daten (auch) als eine Reihe von Knoten dargestellt werden können, können die so erzeugten Daten an andere Systeme verteilt werden – vorausgesetzt, das System kann XML konvertieren. Hierzu muss nur sichergestellt sein, dass eine XML-Schnittstelle auf beiden Seiten besteht und Quell- und Ziel-Datenbank dieselbe XML-Grammatik verwenden.[113] Die XML-Daten werden gekapselt und über den HTTP-Port versandt (SOAP macht genau dies).

Daten können (relativ einfach) aus einer relationalen Datenbank ausgelesen und nach XML transformiert, d.h. auf eine Knoten-Struktur eines XML-Dokumentes abgebildet werden. Wichtig wäre es weiterhin, einzelne Knoten abzufragen. Eine solche Abfrage mittels XSL behandelt die gesamte Datenbank als ein großes XML-Dokument: Die Query-Syntax von XSL wurde als XPath spezifiziert und findet sich unter http://www.w3.org/TR/xpath.

Der effizienteste Weg XPath für Queries zu nutzen ist der, eine Anwendung zu erstellen, die diese XPath-Queries für den verwendeten Datenbanktyp übersetzt. Damit hängen die Einzelheiten von dem Datenbanksystem ab. Eine andere Möglichkeit besteht darin, XSL zur Selektion der Knoten im DOM zu nutzen. In diesem Fall könnte die verwendete Datenbank wechseln, während die Syntax der Anfragen gleich bliebe.[114] Der Nachteil hieran: es müssen zunächst alle Datensätze gelesen werden, bevor man (u.U. nur wenige) verarbeiten will. Zur Effizienzsteigerung ist es daher notwendig, die Querying-Fähigkeiten der verwendeten Datenbank zu nutzen und eine Kombination von XSLT und XPath zu verwenden: XPath erlaubt die Selektion von Knoten auf der Basis von Werten anderer Knoten im Baum. Die Modellierung der hierarchischen Strukturen ist in SQL recht einfach, so dass XPath-Anfragen auch durch SQL-Anfragen ausgedrückt werden können.

Insgesamt bietet XML eine mächtige Methode zum Austausch von Daten zwischen veschiedenen Systemen.[115]

---

[112] folgende Produkte (Auswahl) scheinen in diesem Umfeld interessant zu sein: LivePage, das zwischen XML und einer relationalen Datenbank konvertiert (http://www.livepage.com/); XMLDB nimmt ein XML-Dokument, generiert Anweisungen zur Erstellung der nötigen Tabellen und trägt die Daten ein (http://ceriumworks.com/tech.html);

[113] ODBC wurde speziell entwickelt, um die Lücke zwischen den Datenbank-Systemen zu überwinden - der abstrakte Mechanismus mit XML ist allerdings weit fortschrittlicher

[114] vgl. R. Anderson, u.a.: a.a.O., S. 446

[115] vgl. R. Anderson, u.a.: a.a.O., S. 450. Die Autoren zeigen an einem Beispiel, wie ein Produktkatalog in eine einfache Datenbank konvertiert werden kann

5.2.4 XML-Dokumente ablegen

Die Wiederverwendung von XML-Dokumenten zwingt zu Überlegungen, diese persitent zu speichern. Obwohl es für viele Anwendungsfälle ausreichend ist, Dokumente einfach in (Text-)Dateien abzulegen (um sie anschliessend mit einem der vielen XML-Tools zu bearbeiten), gilt dies nicht für wirklich ernsthafte Anwendungen. Damit ist aber auch eine andere Sicht auf ein XML-Dokument notwendig: zu bearbeiten ist dann nicht mehr eine elementare XML-Einheit, sondern zu bearbeiten sind dann einzelne Knoten.

Eine Anwendung liest eine XML-Datei nicht direkt ein, sondern sie kommuniziert über einen Parser oder XML-Prozessor. Dieser Prozessor, der XML-Daten einliest, um sie anschliessend an die Anwendung weiterzureichen, ist eine Komponente der Anwendung: Die Aufgabe von DOM ist es, die hierarchische Knotenstruktur zu verbergen und die Arbeit mit einem Baum von Knoten zu ermöglichen. XML in der Form von Text dient nur als bequemes Vehikel für Daten zwischen Systemen bzw. als Eingabeeinheit für einen Parser.

Die Abbildung durch Knoten ermöglicht es, auch parallel an einem XML-Dokument zu arbeiten. Voraussetzung hierfür ist ein System, das die Kontrolle über einzelne Knoten erlaubt bzw. über einzelne Dokumentteile ermöglicht. Ein Server, der die Knoten auf diese Weise zur Verfügung stellt oder ganze XML-Dokumente, die wiederum Teile größerer XML-Dokumente sein können, wird als XML-Server[116] bezeichnet.

Sowohl relationale als auch objektorientierte Datenbanken bieten hierbei die Kontrolle über kleine Bestandteile von Informationen. Es gibt zwar Tools, die eine Abbildung von Objekten auf relationale Datenbanken vornehmen, aber die Modellierung von hierarchischen Daten in einer relationalen Datenbank erfordert viele Joins auf Tabellen. Wenn der Objektbaum eine große Tiefe besitzt, dann müssen viele Tabellen benutzt werden, worunter die Performanz leidet.[117]

Objektorientierte Datenbanken[118] sind insbesonders dann von Vorteil, wenn ‚echte' Objekte vorliegen, die komplexe Beziehungen haben und tief verschachtelt sind. Ihre hiearchische Struktur entspricht der Hierarchie in XML-Dokumenten, darüber hinaus können individuelle Knoten eines Dokumentes als Objekte einer Datenbank repräsentiert werden.

Objektorientierte Datenbanken sind aber immer etwas langsamer als relationale Datenbanken und in der Praxis sind nur wenige OO-Datenbanken im produktiven Einsatz: Trotz der vielen Vorteile von OO-Datenbanken sind relationale Datenbanken weit verbreitet, weil sie viele reale Probleme modellieren und gute Antwortzeiten liefern.

Die grundlegenden Konzepte einer relationalen Datenbank sind bekanntlich (1) Tabellen, (2) Anfragen (Queries) und (3) Joins. Es stellt sich die Frage, wie diese Kon-

[116] als XML-Server wird bspw. Tamino vermarktet: Tamino speichert XML-Dokumente als XML. Ein wichtiges Feature ist die zusätzliche Abstraktionsschicht über bestehende Datenbanken, die eine Abbildung der Datenbank von und nach XML vornimmt. Das entsprechende Modul nennt sich X-Node (http://www.softwareag.com/)

[117] vgl. R. Anderson, u.a.: a.a.O., S. 429

[118] das Produkt eXcelon ist eine OO-Datenbank, die XML unterstützt, indem eXcelon die XML-Daten analysiert und neue Objekte erzeugt. D.h. statt Dokumente werden die entsprechenden Knoten (Elemente) in der Datenbank gehalten. Siehe dazu http://www.odi.com/excelon/. Ein weiteres Produkt stammt von POET: ähnlich wie eXcelon werden XML-Daten in Knoten transformiert (http://www.poet.com)

zepte im Modellierungsprozess der XML-Welt mit ihren hierarchischen Knoten verwendet und wie Konvertierungen automatisiert werden können.

Eine Zeile einer Tabelle kann leicht als ein Element repräsentiert werden, wobei die Spalten einer Tabelle die Attribute nachbilden. Problematischer ist es, die Beziehungen zwischen den Elementen (Knoten) auszudrücken. Hierzu müssen Joins verwendet werden: eine Eltern/Kind-Relation kann durch einen Join von einem Fremdschlüssel in der Tabelle des Kindes auf den Schlüssel der Eltern-Tabelle modelliert werden. Die Eltern/Kind-Hierarchie kann also durch Referenzen in der Kind-Tabelle auf die Eltern-Tabelle angezeigt werden, so dass relationale Datenbanken als Mechanismus zur Speicherung von XML-Knoten verwendet werden können.

5.3 Implementierungsrisiken

Es gilt zwar im allgemeinen die Regel „es gibt nichts Gutes, außer man tut es“ – aber allzu blauäugig sollte man sich nicht in die XML-Welt wagen. Problematisch ist vor allen Dingen die Schaffung eines gemeinsamen Datenmodells bzw. die Ableitung entsprechender Meta-Daten (was zunächst nichts mit XML zu tun hat), aber die Qualität der XML-Dokumente erheblich beeinflusst. Weitere Aspekte sind eher von technischer Natur: Sicherheit und Performanz .

5.3.1 Qualität der XML-Dokumente

Der Erfolg einer XML-Anwendung wird davon abhängen, wie gut die XML-Dokumente (bezogen auf den Inhalt, aber auch hinsichtlich ihrer Flexibilität gegenüber möglichen Anforderungen in der Zukunft) sind. Angesprochen ist damit der Entwurfsprozess, d.h. die Modellierung der Daten (damit das Verständnis für die Strukturen und die Bedeutung der Informationen).

5.3.2 Sicherheit und Performanz

Da XML ein textbasiertes Format ist, sind 2 Punkte besonders zu beachten: Sicherheit und Performanz.

- Sicherheit tritt als Problem primär an Systemgrenzen auf (was allerdings kein spezifisches XML-Problem ist). Besondere Aufmerksamkeit ist allerdings dann erforderlich, wenn die XML-Daten zusätzliche Aufgaben übernehmen sollen – bspw. Zugriffsberechtigungen zu verwalten haben. Verlagert man solche Arbeitsaufträge in die XML-Daten, so ist genau zu prüfen, ob bzw. in welcher Schicht die Sicherheitsmechanismen noch greifen können.

- Performanz könnte als Problem angesehen werden, wenn man den Speicherbedarf eines XML-Dokumentes und den darüber hinaus notwendigen Aufwand des XML-Parsers (Speicherung und Rechenzeit) in Rechnung stellt, und sie mit herkömmlichen (individuellen) Mechanismen vergleicht, die häufig für die jeweilige Gegebenheit optimiert sind. Performance-Untersuchungen sollten frühzeitig im Rahmen eines Anwendungsprojekts durchgeführt werden.[119]

[119] vgl. zur Vorgehensweise von Performance-Untersuchungen in Middleware-Lösungen: A.Nogli: Performance-Analyse von EJB-Applikationsservern, in JavaSPEKTRUM 1/2001, S. 39ff. Applikations-Server gehören zu den Middleware-Produkten, bilden sie doch das technische Binde- und Synchronisationsglied zwischen der bestehenden Geschäftswelt und der Dynamik der IT-(Internet-)Welt

6 Empfehlungen zur weiteren Vorgehensweise

Die Entwicklung von XML weg von den einfachen Wurzeln einer Auszeichnungssprache hin zu einem großen und breiter gefächerten Anwendungsfeld verläuft rasant. „XML“ impliziert eine mächtige und elegante Art und Weise, wie man mit Daten umgeht, diese mit anderen (auch über die Grenzen unterschiedlicher Betriebssysteme hinweg) austauscht.

Folgende Arbeitsschritte sind absehbar:

- die Daten inventarisieren, modellieren und die gewonnenen Meta-Daten in einem Repository ablegen (i.S. einer Strukturanalyse)
- den Datenfluss analysieren, erkennen, wo ggfs. Transformationen vorzunehmen sind (i.S. einer Schnittstellenanalyse)
- prüfen, wo und wie häufig die Nutz-Daten gegen die Meta-Daten zu validieren sind
- prüfen, in welcher Form die XML-Dokumente zu speichern sind
- prüfen, ob Zusatzdienste einzurichten sind (z.B. cash flows generieren, Bewertungen)
- prüfen, ob eine Historienführung/Archivierung notwendig ist bzw. per XML-Dokumente umgesetzt werden kann

Ein wichtiger Aspekt, der ausgiebig zu untersuchen wäre, ist die Suche bzw. Schaffung und Pflege eines standardisierten XML Vokabulars bzw. die Beantwortung der Frage, wie die Koexistenz mit proprietären XML-Tagfamilien aussehen könnte: Anwendungsintegration wird zwangsläufig auf eine Kombination von internen und externen Bauplänen hinauslaufen: das größte Problem bei der Integration von Anwendungen ist es, die Daten konsistent zu interpretieren. Hierzu ist es notwendig, ein Repository zu schaffen, in dem alle Meta-Daten strukturiert (entsprechend dem hinterlegten Datenmodell) abgelegt sind. Damit verbunden sind

- ein Verständnis über das ‚Tagging’ der Daten und die Schaffung einer lokalen Tag-Familie
- grundlegende Kenntnisse über die Konstruktion von XML-Dokumenten
- grundlegende Kenntnisse über die Konstruktion von XML-Schemata und der Validierung von XML-Dokumenten
- Strategien für die organisatorische und programmiertechnische Umsetzung, um die Datenintegrität auch über die Zeit sicherzustellen.

ESTES empfiehlt in diesem Zusammenhang ein ‚diszipliniertes’ XML; hierunter subsumiert er

- „Create a single point of maintenance
- Create a repository of extended information beyond that supported by the implementation language
- Provide operational control of both program and document versions“ [120].

[120] vgl. D. Estes: Disciplined XML, in EAI Journal 1/2001, S. 60

Stichwortverzeichnis

Literaturverzeichnis

Ade, R.: Mangelnder Austausch, in iX 1/2001, S. 122ff

Alexander, S.: Praxis und Trends bei EJB-Servern, in Computerwoche 16/2001, S. 20f

Anderson, R. u.a.: XML professionell, Bonn 2000

Bastian, R.: Virtuelles Data Warehouse, in geldinstitute 3/1997, S. 56f

Behme H./**Mintert**: XML in der Praxis, München 2000

Blum, D.: EAI-Beispiele im R/3-Umfeld, in Computerwoche 47/2000, S. 28f

Büchler, J.: Pack den TIGRA in den Tank, in IT-FOKUS 4/2001, S. 20ff

Cheng J./**Xu** J.: IBM DB2 XML Extender, ICDE'00 Conference, San Diego, 2/2000 (kann bei DR. SIEGERT & PARTNER angefordert werden)

Estes, D.: Disciplined XML, in EAI Journal 1/2001, S. 57ff

Farkus, M./**Milward**, R.: Working up a lather over SOAP, in RISK 10/2000, S. 42f

de la **Fe**, S./**Hoffmann** C./**Huh** E.: XBRL Taxonomy Financial Reporting for Commercial and Industrial Companies, US GAAP, http://www.xbrl.org/us/gaap/ci/2000-07-31/us-gaap-ci-2000-07-31.pdf

Fernbach (Hrsg.): Orientierung im Datenmeer, o.O., 2001

Finkelstein C./**Aiken** P.: Building Corporate Portals with XML, New York u.a. 2000

Folan, St.: Avoiding Babel, in Risk Professional 2/2001, S. 24ff

Frey, J/Lobeck, A.: Weg von der Insel, in IT-FOKUS 4/2001, S. 32ff

Hranitzky, N.: Editorial XMLspektrum, in JavaSPEKTRUM 1/2001, S. 49ff

Hubert, R./**Hollunder**, B./**Buchheit**, M.: Applikations-Server öffnen die Unternehmens-IT für das Internet, in JavaSPEKTRUM 1/2001, S. 55ff

Jaenicke, C.: The Information-Centric Enterprise – How XML makes all your data work for you, in EAI Journal 5/2000, S. 22ff

IBM (Hrsg.): Integrated Scenario for Developing XML Applications Using the IBM XML Tools and Visual Age for Java, 1997/2000, http://www7.software.ibm.com/vad.nsf/data/document4326/

Jost, M./**Kalhoff**, A.: Internet-Technologie im Risikomanagement, in geldinstitute 9/2000, S. 42ff

Jung, A.: XML – Schlüsseltechnologie für Softwarearchitekturen, OBJEKTspektrum 1/2001, S. 71ff

Kolb, G.: Jeder kann mit jedem, in IT Management 1/2001, S. 77ff

Kolland, M./**Mehner**, T./**Kuhn**, K.-J.: Software-Plattformen – Mehr als ein Schlagwort, in Wirtschaftsinformatik 1/1993, S.24ff

Linthicum, D.S.: Creating Enterprise Metadata Repositories for EAI and e-Business, in EAI Journal 1/2000, S. 13ff

Lory, R.: Middleware, in Schweizer Bank 12/2000, S. 92f

Lutz, J.C.: EAI Architecture Patterns, in EAI Journal 3/2000, S. 64ff

Marr, M.: Fleißige Ameise, in iX 2/2001, S. 56ff

McGoveran, D: Business Semantics, in EAI Journal 10/2000, S. 10

Morgenthal, J.P.: XML and the New Integration Frontier, in EAI Journal 3/2000, S. 27ff

Morgenthal, J.P.: Analyzing Approaches to XML Messaging, in EAI Journal 1/2001, S. 35ff

Nogli, A.: Performance-Analyse von EJB-Applikations-Servern, in JavaSPEKTRUM 1/2001, S. 39ff

Nußdorfer, R.: Daten- und Middleware-Integration, in IT FOKUS 9/2000, S. 55ff

Nußdorfer, R.: Evolution durch neue Technologien, in IT FOKUS 1-2/2001, S. 39ff

Oberdorfer, R.: Allround-Adapter, in iX 5/2001, S. 136ff

Oberdörster, A: XML mit Serviervorschlag, in c't 6/2000, S. 244ff

Oberdörster, A: XSL im Detail, in c't 6/2000, S. 250ff

o.Verf.: SAP setzt auf XML für übergreifende Integration, in Computerwoche 18/2000, S. 26

o.Verf.: Schemata machen XML für den Datenaustausch tauglich, in Computerwoche 47/2000, S. 17f

o.Verf.: IBM liefert ihr neues Flaggschiff aus, in Handelsblatt v. 18.12.2000

o.Verf.: SOAP soll das Rückgrat von Web-Services bilden, in Computerwoche 50/2000, S. 73f

o.Verf.: FpML set to become industry standard, in Risk 12/2000, S. 22

o.Verf.: MlFi, in Risk 1/2001, S. 50

o.Verf.: IBM wetteifert mit Microsoft beim Angebot von XML-Tools, in Computerwoche 49/2000, S. 20

o.Verf.: o.Verf.: The SunGard Network Trade Model, o.J.

o.Verf.: OS/390 ist tot – lang lebe zOS, in Computerwoche 16/2001

o.Verf.: Bei XML-Schemasprachen ist kein Standard in Sicht, in Computerwoche 13/2001, S. 24

o.Verf.: Suns ONE setzt ganz auf Java und XML, in Computerwoche 7/2001, S. 32

Peterson, M.: Data Chaos, in EAI Journal 11-12/2000, S. 47ff

PwC (Hrsg.): Value Reporting Forecast: 2001 Trends in Corporate Reporting, o.O.

Ring, K./**Ward-Dutton**, N.: Enterprise Application Integration: Making the Right Connections, Ovum Ltd., London 1999, S. 41ff.

Siegert, H./**Louis**, H./**Drabben**, M.: Konzernweites Informationsmanagement via XML, in Betriebswirtschaftliche Blätter, 11/2000, S. 11/516ff

Siffring, A.: SOAP soll das Rückgrat von Web-Services bilden, in Computerwoche 50/2000, S. 73f

Tailor, P.D.: RiskMetrics to Introduce RiskServer – the First Internet-Based Risk Engine, in RiskMetrics Journal 11/2000, S. 5ff

Thelen, F.: XML, Java und Persistenz, in JavaSPEKTRUM 1/2001, S. 59ff

Tresch, M.: Middleware – Schlüsseltechnologie zur Entwicklung verteilter Informationssysteme, in Informatik-Spektrum 19/1996, S. 249ff

Vasters, C.F.: SOAP – die Lösung aller Interoperabilitätsprobleme?, in OBJEKTspektrum 6/2000, S. 26ff

Westphal, R.: Strong Tagging als Ausweg aus der Interface-Versionshölle, in OBJECTspektrum 4/2000, S. 60ff.

Zelewski, S.: Schnittstellen bei betrieblichen Informationssystemen, Arbeitsbericht 6, Universität Köln 1986, S.6ff

Linkverzeichnis

Link	Beschreibung
www.xml.com	die XML-Zentrale; hier beginnt jede Reise in die Welt von XML
www.oasis-open.org/cover/sgmlnew.html	hier werden ständig neue Produkte und Projekte beschrieben
www.netheaven.com/~coopercc/xmlparser/intro.html	XMP-Parser-Modul für Perl
www.alphaworks.ibm.com	IBM ist die erste Anlaufstelle, wenn es um Tools im XML-Umfeld geht
www.riposte.com/xosl/	mit xosl will Microsoft eigene ‚Technologien' im ansonsten standardkonformen XML-Umfeld verankern
www.sun.com	Sun zählt zu den Vorreitern in Sachen Standardkonformität
www.megginson.com/SAX/	SAX-API
www.w3.org/DOM/	DOM-API
www.fpml.org	FpML – Produktbeschreibung via DTDs (Finanzprodukte)
www.xbrl.org.	XBRL – Reportbeschreibung via Schemata (Finanzreports)
www.finxml.org	FinXML – Produktbeschreibung via DTDs (Finanzprodukte)
www.weatherml.org	XML-basierte Produktbeschreibung für Wetterderivate
www.poet.com/ecs	POET eCatalog Suite
www.poet.com/oss/	POET Object Server Suite
Xml.apache.org/xerces-j/	Xerces von Apache (Parser)
Xml.apache.org/xalan/	Xanlan von Apache (XSLT Prozessor)
Java.sun.com/xml/docs/binding/DataBinding.html	XML Data Binding
www.w3.org/XML/Schema.html	XML Schema
www.xml.org/xmlorg_registry/index.shtml	Tag-Familien diverser Branchen
www.softwareag.com/tamino	Tamino: XML-Server für Electronic Business
www.wdvl.com	Web Developers Virtual Library – Fundgrube für XML-Tools
www.risk.sungard.com	SunGard Network Trade Model
www.w3.org/TR/REC-rdf-syntax	detaillierte Darstellung von RDF