Donald Kossmann, Gustavo Alonso

Software Mass Customization

Dieser Artikel führt ein neues Ziel bei der Entwicklung von Informationssystemen ein: Software Mass Customization. Der Artikel soll motivieren zukünftige Informationssysteme mit diesem Ziel zu entwickeln, und zeigt auf, wieso die heutige Technologie zur Entwicklung von Informationssystemen nicht ausreichend zur Erreichung dieses Ziels ist. Es wird ein kurzer Exkurs gegeben, wieso XML-Technologien zielführend sein könnten. Der Artikel soll zur Diskussion anregen.

1 Prognose

»The SQL party is over. Could somebody, please, tell the guests to go home.« Dieses Zitat kommt von Roger Bamford, dem Principal Architect von Oracle, auf der SIGMOD-Konferenz 2006. Wenn man einen »Vision 2016«-Artikel schreibt, dann muss man drastisch sein. Deshalb gehen wir noch einen Schritt weiter: 2016 ist auch die Java-Party vorbei! Java und SQL werden hier als Platzhalter für die heutige Generation von Programmiersprachen verwendet, z.B. PHP, die Programmiersprachen von .NET, ABAP oder eben Java/J2EE. Noch mehr: Java und SQL werden als Platzhalter für die heute gängigen Softwarearchitekturen verwendet, in denen in einer Mehrschichtenarchitektur Anwendungslogik in einer Programmiersprache wie Java geschrieben wird und die Daten persistent in einer Datenbank gehalten werden, auf die mittels einer Anfragesprache wie SQL zugegriffen wird.

Natürlich wird es auch im Jahr 2016 noch SQL- und Java-Anwendungen geben – genauso wie es heute noch zig Millionen Zeilen von aktivem und nicht wegzudenkendem COBOL-Code gibt. Doch SQL und Java werden stagnieren oder gar zurückgehen. Das Problem ist, dass die heutigen Programmiersprachen und Softwarearchitekturen nicht gut die Anforderungen der jüngsten gesellschaftlichen, wirtschaftlichen und technischen Trends der Softwareindustrie erfüllen. Nach Ansicht der Autoren wird sich die Softwareindustrie analog zu anderen Industrien entwickeln. Konkret wird sich das Konzept der »Mass Customization« [Wikipedia] durchsetzen. Generell ist das Ziel von Mass Customization, Produkte nach individuellen Wünschen der Kunden zu fertigen, doch dabei genauso effizient zu sein wie bei einer traditionellen Massenfertigung. Die Magie liegt darin, dass in diesem Modell Benutzer von Software zu Entwicklern von Software werden. Hierdurch wird die unvermeidbare Komplexität der Software, die in der Modellierung und Automatisierung einer komplexen Welt liegt, zumindest teilweise von den Benutzern, also von Domänenexperten, bewältigt. Die Aufgabe der Informatik liegt darin, von der technischen Komplexität (unterschiedliche Hardware- und Softwareplattformen, Optimierung, Sicherheit etc.) zu abstrahieren und die notwendige Flexibilität für individuelle Erweiterungen zu geringen Kosten zu schaffen.

Dieser Artikel versucht, die gesellschaftlichen und wirtschaftlichen Hintergründe für Mass Customization von Software zu motivieren und zu erklären, wieso »Java«, »SQL« und die klassischen Softwarearchitekturen nicht hinreichend sind. Des Weiteren versucht der Artikel, ein wenig in die Kristallkugel zu blicken, wie das nächste »Java« und »SQL« aussehen könnte. Insbesondere wird versucht, Prognosen für die Rolle von Datenbanken und Datenbanktechnologie zu geben. Dieser Artikel wird keine Lösungen präsentieren und auch kein vollständiges Bild geben. Er dient dazu, Beobachtungen wiederzugeben und Diskussionen anzuregen.

2 Hintergrund: Gesellschaftliche und wirtschaftliche Trends

Nach unserer Meinung wird die nächste Technologiewelle durch zwei engverknüpfte gesellschaftliche Trends angetrieben: (a) Globalisierung und (b) Individualisierung. Zur Zeit des kalten Krieges hatte die Auseinandersetzung um die Eroberung des Weltraums zwischen der USA und der Sowjetunion einen starken Einfluss auf den technologischen Fortschritt. Nach Ende des kalten Krieges hatten uns die Schlachten der wirtschaftlichen Imperien wie derzeit Microsoft gegen Google in Atem. Laut Vorhersage von Tom Friedman in seinem Buch »The World is Flat« [Friedman 2006] wird es in Zukunft Frau Krishnamurty gegen Herrn Müller gegen Frau Jones heißen. Oder noch besser: Frau Krishnamurty mit Herrn Müller und mit Frau Jones. Das Individuum wird zum Global Player, indem es mit anderen Individuen kooperiert und auf einem globalen Markt agiert.

Friedman nennt in seinem Buch viele Beispiele von Industrien, die bereits diesem Trend gefolgt sind. Am deutlichsten sichtbar wird dieser Wandel durch eine immer tiefere und spezialisiertere Lieferkette in der Produktion. Ein bekanntes Beispiel in Deutschland ist die Fertigung des Porsche Cayenne, in dem die Lieferkette bis zum kleinen Ein-Mann-Unternehmen zum Einfärben von Leder geht. Porsche selber kümmert sich nur noch um Design, Marketing, Endmontage und Kundenservice. In der Konsumgüterindustrie haben die Marktführer wie Procter&Gamble, Nestle und Unilever inzwischen einige ihrer Werke verkauft und konzentrierten sich auf Marketing und Einkauf. Die Werke werden häufig durch ein Management-Buy-out übernommen und spezialisieren sich weiter. Dies führt manchmal zu bizarren Prozessen, in denen ein Unternehmen Rohstoffe unter Ausnutzung seiner Marktmacht einkauft, sie an frühere Werke weiterverkauft und die fertigen Produkte von diesen Werken wieder einkauft.

Ein weiterer Indikator, der Tom Friedmans Thesen belegt, ist die Gliederung von Unternehmen in immer kleinere Profitcenter. Jedes dieser Profitcenter agiert autonom innerhalb und auch außerhalb eines Unternehmens: Das Profitcenter ist nicht gezwungen, mit Profitcentern des eigenen Unternehmens zu kooperieren, sondern kann frei entscheiden. Ein Profitcenter ist häufig eine Abteilung mit vielleicht nur fünf oder weniger Mitarbeitern. Ein Profitcenter muss ständig seine Existenzberechtigung gegen andere Profitcenter innerhalb und außerhalb des Unternehmens beweisen. Diesen Beweis kann das Profitcenter nur antreten, indem es extrem flexibel und anpassungsfähig ist. Es muss täglich in der Lage sein, sich komplett neu zu definieren, ansonsten wird es aufgelöst und die Mitarbeiter for-

mieren sich zu neuen Profitcentern – innerhalb oder außerhalb des Unternehmens. »Reorganisation« ist der Albtraum vieler Mitarbeiter und häufig schimpfen Mitarbeiter, dass die nächste Reorganisation vom Management befohlen wird, bevor die vorherige Reorganisation beendet und evaluiert wurde. Es mag sein, dass manches Management unüberlegt reorganisiert (und fusioniert), um sich zu profilieren, doch es steht außer Frage, dass der Erfolg eines Unternehmens von seiner Anpassungsfähigkeit auf dem Weltmarkt abhängt: Die Welt dreht sich immer schneller und nur die Unternehmen (oder Staaten/Volkswirtschaften), deren Profitcenter sich am schnellsten anpassen und in vorderster Front die Rotation der Welt beschleunigen, werden überleben.

Was ist der Engpass bei der Anpassung eines Unternehmens, eines kleinen Profitcenters oder eines einzelnen Individuums? In der Tat gibt es kulturelle Probleme, und es liegt in der Natur des Menschen, den Wandel zu fürchten, selbst wenn es dem Menschen schlecht geht. Trotzdem ist der Mensch allein nicht der Engpass. In den Mitarbeitern von jedem funktionierenden Unternehmen stecken viele Ideen, wie man die Welt verbessern könnte. Das Problem ist, dass es immer noch sehr schwierig ist, alle diese Ideen umzusetzen. Neben verkrusteter organisatorischer Strukturen und Hierarchien in einer Organisation und Gesellschaft gehören die Zwänge der heutigen Softwarearchitekturen zu den wesentlichen Gründen. Softwaresysteme sind heute so komplex, dass jede Änderung ein großes Risiko darstellt und mit hohen Kosten verbunden ist. »Never change a running system« ist zu einem geflügelten Spruch der Informatik geworden.

Die Gralshüter dieses geflügelten Spruches sind die IT-Abteilungen von Unternehmen. Die meisten Unternehmen waren bereits sehr erfolgreich, alle Fachabteilungen in Profitcenter umzuwandeln. Somit sind die Fachabteilungen ehrgeizig und drängen nach Innovationen, weil sie den globalen Wettbewerb bereits spüren. Die IT-Abteilungen sind in fast allen Unternehmen Stabstellen, deren einziges Ziel es ist, keine Fehler zu machen. Innovation ist gegen das Interesse einer IT-Abteilung. Die Zukunft der Informatik liegt darin, die IT-Abteilung abzuschaffen und der Fachabteilung die Mittel zu geben, ihre Innovation (Prozesse, neue Produkte, neuer Service) zu implementieren. Outsourcing der IT-Abteilung wird nicht reichen! Obwohl viele Informatiker in IT-Abteilungen beschäftigt sind, sollten wir Informatiker uns vor diesem Schritt nicht fürchten. Stattdessen sollten Informatiker in die Fachabteilungen drängen und dort helfen, neue Ideen zu entwickeln und umzusetzen.

Bevor sich dieser Artikel den technischen Anforderungen widmet, noch ein kurzer Blick auf die IT-Industrie. Die IT-Industrie hat wesentliche Beiträge zur Globalisierung und Individualisierung der Gesellschaft geleistet [Friedman 2006]. Das »Web« (oder Internet) ist fast zum Synonym für die technische Grundlage der modernen Informationsgesellschaft geworden. Paradoxerweise hat sich die IT-Industrie selbst kaum gewandelt: Sie wird immer noch von den Big Playern wie SAP, Oracle oder Microsoft dominiert. Man kann sogar fast behaupten, dass das »Web«, das fast sämtliche Industrien verändert und modernisiert hat, die (veralteten) Strukturen der IT-Industrie gestärkt hat. Die Supply Chain der Softwareindustrie ist extrem flach: Die Softwareriesen haben es sich zum Hobby gemacht, Start-ups eher zu schlucken, als sich in eine externe Abhängigkeit zu einem Start-up zu begeben. Für ein IT-Start-up ist es fast unmöglich, sich am Markt zu etablieren. Selbst die Open Source Community ist streng hierarchisch und praktisch wie ein großes IT-Unternehmen organisiert und erlaubt keine tiefen Lieferketten. Die sehr straff geführte Apache-Organisation ist ein gutes Beispiel hierfür. »Off-Shoring« nach Asien, Osteuropa oder Südamerika ist der heute sichtbare Einfluss der Globalisierung auf die IT-Industrie. Doch Off-Shoring bedeutet keinen wesentlichen Strukturwandel, weil Off-Shoring das fundamentale Problem nicht löst und weil das heutige Off-Shoring von den Softwaregiganten kontrolliert wird. Das fundamentale Problem der Softwareindustrie und der Grund für diese starren Strukturen der Softwareindustrie sind dieselben wie für die Daseinsberechtigung von IT-Abteilungen in Unternehmen anderer Industrien: Heutige Softwaresysteme sind einfach zu komplex, als dass ein IT-Start-up seine Innovationen integrieren kann. Die Softwaregiganten haben häufig kein Interesse, die technische Komplexität zu reduzieren, weil diese Komplexität Abhängigkeiten schafft und neue Dienstleistungen im Bereich Beratung ermöglicht. Das »Web« hat die Softwaresysteme paradoxerweise noch komplexer gemacht (eine zusätzliche Schicht) und somit gleichzeitig die Giganten der Softwareindustrie wirtschaftlich noch gestärkt, indem es ihnen einen Grund gegeben hat, neue Versionen ihrer Software zu verkaufen. Doch die Zeit ist reif, um IT-Start-ups im großen Stil einen Marktzugang durch das Web zu verschaffen, weil die Frustration mit den Big Playern der IT-Industrie bei Entwicklern (uns Informatikern) und Anwendern (unseren Kunden) einfach zu groß ist. (Hier ist die Hoffnung Vater des Gedankens!)

3 Beispiel: Konferenzmanagement

Die Komplexität heutiger Softwaresysteme und die damit verbundene Tendenz zur Monopolisierung soll an einem kleinen Beispiel illustriert werden: einem webbasierten Konferenzmanagementtool. Praktisch alle größeren Konferenzen werden heute mit einem solchen Tool abgewickelt. So ein Tool unterstützt die Einreichungen von Beiträgen, die Zuweisung von Beiträgen zu Gutachtern, die Verwaltung der Gutachten, die Diskussionen im Programmkomitee, die Benachrichtigung von Autoren sowie die Einreichung der Camera-Ready-Versionen von angenommenen Beiträgen. In der Datenbank- und vielen anderen Communities hat sich CMT von Microsoft als Konferenzmanagementtool durchgesetzt. Zur Unterstützung des Begutachtungsprozesses bei Zeitschriften hat sich sowohl bei IEEE als auch bei ACM das Manuscript-Central-System von ScholarOne durchgesetzt.

Wenn man als Organisator einer Konferenz (oder Herausgeber einer Zeitschrift) ein wenig über das gewünschte Tool nachdenkt, würde man abschätzen, dass ein Praktikant ein solches Tool in ca. zwei bis drei Wochen erstellen kann. Die Technologie ist da (HTML/XML + Java/C# + SQL) und wohl verstanden, und der Begutachtungsprozess selber ist einfach und ebenfalls gut verstanden. Tatsächlich sind aber CMT, ManuscriptCentral und verwandte Systeme extrem komplex und geben Anlass zu permanenter Frustration. Wo liegt das Problem?

Das Problem ist, dass Systeme wie CMT oder ManuscriptCentral gewachsen

sind und »Generizität durch Systemparameter« erreicht haben. Wenn eine Konferenz eine neue Idee eingeführt hat wie z.B. eine Auktion von Gutachtern um Beiträge, dann wurde diese Idee in diese Systeme eingebaut und zusätzlich wurde ein Parameter eingeführt, mit dem Konferenzen entscheiden können, ob sie dieses neue Feature verwenden möchten oder nicht. Eine erfolgreiche Software (wie CMT) hat Organisatoren von Konferenzen zu vielen Ideen inspiriert: unterschiedliche Auktionsmodelle, unterschiedliche Conflict-of-Interest-Modelle, Double-blind Reviewing, virtuelle Programmkomiteesitzungen, Organisation von Programmkomitees in Tracks und Gruppen usw. Diese Komplexität und der Dschungel an Parametern hat zu folgenden Frustrationserlebnissen geführt:

- *Organisatoren machen Fehler bei der Konfiguration der Software:* Ein Fehler, der bei der Konfiguration gemacht wurde, kann eventuell erst Monate später sichtbar werden und ist dann nicht mehr reparierbar. Die Konfiguration und Anpassung der Software findet am Anfang einmal für alle Zeiten statt und nicht kontinuierlich.
- *Das System verkapselt alle Daten:* Man kann einfachste Anfragen nicht stellen, wenn das System sie nicht a priori unterstützt. Im Fachjargon sagt man, dass ein System nur »pre-canned« Anfragen unterstützt. Zum Beispiel kann man in CMT oder ManuscriptCenter praktisch nicht ermitteln, wie streng ein Gutachter war (d.h. die vergebene Durchschnittsnote des Gutachters).
- *Ein Organisator kann die Software nicht ändern und die Spezifika seiner Konferenz einbringen:* Zum Beispiel könnte die Konferenz eine besondere Policy für den Umgang mit Beiträgen mit patentfähigen Ideen haben. Selbst wenn der Organisator den Sourcecode hat und sich perfekt in der Programmierung solcher Systeme auskennt, ist dies nicht möglich, weil sich die Investition nicht lohnt. Eine Änderung kann nur vom »Owner« der Software durchgeführt werden und wird entweder für alle oder für keine Konferenz aktiviert oder es wird ein zusätzlicher Parameter eingeführt.

Die Ursache für diese Probleme ist, dass erfolgreiche Softwaresysteme mit jeder Version in großen Schritten wachsen: In jeder Version werden die gesammelten Ideen der gesamten Benutzer-Community verarbeitet. Dadurch ist dieser Schritt für den einzelnen Benutzer der Software zu groß: Der Benutzer kann die neuen Features der Software nicht nutzen (oder braucht eine teure Schulung). Der Benutzer kann die Software nicht ändern, selbst wenn der Benutzer alle technischen Voraussetzungen hat. Ziel muss es letztendlich sein, dass eine Software kontinuierlich (nicht diskret in Releases) und individuell mit jedem einzelnen Benutzer wächst und nicht jeder Benutzer mit dem »durchschnittlichen« Wachstum der Software vorlieb nehmen muss, das für den individuellen Benutzer zu schnell oder zu langsam sein kann, aber praktisch nie perfekt passt.

4 Anforderungen

Wie kann die kleine Fünf-Personen-Fachabteilung ihre neuen Ideen implementieren? Wie kann ein Unternehmen wie Siemens von heute auf morgen ohne organisatorische Schmerzen eine Kooperation mit Nokia schließen, um sein Telekommunikationsbusiness outzusourcen? Wie kann Frau Jones ihr kleines Software-Gimmick entwickeln und auf dem Internet vermarkten? Die Antwort lautet: Anstatt sechs Millionen organisierte und gemanagte Programmierer wird die Welt in Zukunft sechs Milliarden unorganisierte Programmierer haben. Das heißt, jeder Mensch wird zum potenziellen Programmierer, so wie heute bereits jeder Mensch ein potenzieller Schriftsteller, Spediteur oder Eisverkäufer ist. Zusätzlich zu einem Memo werden Sachbearbeiter in Zukunft »Skripten« schreiben, die kleine Prozesse implementieren. Als Metapher soll der Begriff der »Mass Customization« [Wikipedia] verwendet werden. In der industriellen Fertigung bezeichnet dieser Begriff die Fertigung von Produkten nach individuellen Kundenwünschen mit derselben Effizienz wie eine Massenfertigung – ein wichtiger Aspekt hierbei ist die Einbindung des Kunden in den Entwurfs- und Produktionsprozess. Ähnlich wie Client/Server-Computing Skalierbarkeit dadurch erreicht hat, dass die Hardwareressourcen des Clients ausgenutzt werden, erreicht Mass Customization Skalierbarkeit, indem die intellektuellen Ressourcen des Clients ausgenutzt werden.

Uns ist bewusst, dass mit dieser Forderung nach Mass Customization in der Softwareindustrie nichts anderes als die Suche nach dem heiligen Gral des Software Engineering propagiert wird. Seit 40 Jahren wird zum Beispiel an einer kollaborativen und komponentenorientierten Softwareentwicklung gearbeitet, und der kritische Leser wird zu Recht fragen, wieso gerade in den kommenden zehn Jahren ein Durchbruch erzielt werden sollte. Dieser Artikel wird keine Lösungen bieten, sondern lediglich versuchen, die Suche etwas zu fokussieren. Als Informatik-Community sollten wir allerdings nicht zu pessimistisch sein. Es geht nicht darum, mit künstlicher Intelligenz automatisch Software zu generieren oder automatisch zu integrieren. Es geht darum, die richtige, nächsthöhere Abstraktionsstufe für die Softwareentwicklung zu finden, um den Programmierer möglichst wenig mit technischen Details wie Optimierung, Sicherheit und der Implementierung anderer Anforderungen zu belasten. Ein Großteil der Programmierung findet ohnehin durch eine *Benutzung* der Software statt, also unbewusst und ohne spezielle Programmierkenntnisse. Außerdem haben nicht wir, die Informatiker, sondern die Gesellschaft das Problem definiert (Abschnitt 2). Wir können davon ausgehen, dass die Gesellschaft einiges beitragen wird. Computer sind zu einem Massenprodukt geworden; wieso sollte Programmierung nicht in zehn Jahren zur Allgemeinbildung gehören? Wer die Welt verändern möchte, egal wie, muss Software produzieren. Wir können diesen Leuten helfen oder es sein lassen, doch aufhalten können wir diese Veränderungen nicht.

Auf jeden Fall wird Mass Customization eine viel tiefere und durchgehendere Einbeziehung des Benutzers in den kompletten Softwareentwicklungsprozess erfordern und somit an einigen Grundpfeilern der heutigen Praxis der Softwareentwicklung rütteln. Als Beispiel soll das Wasserfallmodell dienen, das in Varianten die aktuelle Softwareentwicklung in der Praxis immer noch bestimmt. Das Wasserfallmodell sieht eine Interaktion mit dem Benutzer nur in den frühen Phasen (Anforderungsanalyse und teilweise Entwurf) sowie in späteren Phasen (beim

Testen und der Abnahme) vor – dieses Modell wird in Zukunft nicht tragbar sein, weil der Benutzer den kompletten Prozess und alle Entwicklungszyklen tragen wird. Entsprechend werden »Java« und »SQL« und andere Sprachen, Technologien und Softwarearchitekturen, die dediziertes Expertenwissen für die mittleren Phasen des Wasserfallmodells voraussetzen, nicht tragbar sein. Im Folgenden werden einige Anforderungen an modernes Software Engineering und moderne Informationssysteme gestellt, die nach unserer Auffassung für ein solches Mass Customization notwendig sind.

4.1 Anforderungen

Die folgende Liste enthält einige wichtige Anforderungen an eine IT-Infrastruktur, um das Ziel der Mass Customization von Software zu erreichen. Diese Liste soll zur Diskussion anregen. Sie kann nicht vollständig sein, da noch niemand alle Auswirkungen dieses Zieles überblicken kann.

- *Fine-grained extensibility:* Es muss möglich sein, in sehr feingranularer Weise Software zu ändern. Dies betrifft das Datenmodell einer Software, die Funktionalität einer Software sowie den Workflow (d.h. den Ablauf bestimmter Funktionen einer Software). Zum Beispiel muss es möglich sein, ein Kommentarfeld zu einem Business-Objekt hinzuzufügen. Weiterhin muss es möglich sein, die Autorisierung, eine bestimmte Funktion, die entscheidet, wer welche Daten sehen darf, zu ändern. Auch die Reihenfolge, in der ein Auftrag abgewickelt wird, muss zu ändern sein. Diese Anpassungen müssen individuell für einen Benutzer oder eine Gruppe von Benutzern erfolgen können. In der Regel werden diese Anpassungen durch einen Benutzer interaktiv mit Hilfe einer grafischen Benutzerschnittstelle gemacht (ähnlich wie ein Benutzer heute bereits das Hintergrundbild seines PC-Desktops einstellen kann).
- *Component extensibility/Mashup:* Neben der feingranularen Anpassung der Software muss es die Möglichkeit geben, ganze Softwareblöcke (Module oder Komponenten), bestehend aus Datenmodell, Funktionen und Workflows, in einer gröberen Granularität auszutauschen. Salesforce.com bietet zum Beispiel mit dem AppExchange Framework Benutzern die Möglichkeit, komplette Konfigurationen von anderen Benutzern zu übernehmen. Hierdurch entsteht ein weiterer Markt.
- *Hot Swap:* Die Anpassung der Software (fein- und grobgranular) muss stattfinden können, ohne den Betrieb der Software zu unterbrechen. Das heißt, Auszeiten für eine Durchführung einer Schemaevolution in der Datenbank oder für eine Neukompilation von Programmen ist nicht akzeptabel.
- *Integrated Tool Chain:* Neue Software und Upgrades müssen automatisch oder per Knopfdruck aktiviert werden. Aufwendige Installationen und Konfigurationen sind nicht akzeptabel. Insgesamt müssen alle Werkzeuge, die die Entwicklung und Anpassung der Software unterstützen, eng an den Betrieb gekoppelt sein. So muss es zum Beispiel möglich sein, direkt während des Betriebes der Software Texte (zum Beispiel Hilfsanweisungen) zu editieren. Die Grenzen zwischen Entwicklung, Anpassung und Benutzung der Software müssen verschwimmen. Analog verschwimmen die Grenzen zwischen den Rollen der beteiligten Akteure (z.B. Entwickler, Berater und Nutzer).
- *Code is data:* Programmcode muss genauso wie Daten gehalten werden. Das heißt, es muss möglich sein, Programmcode zu verschicken, so wie man Daten verschickt. Es muss möglich sein, Integritätsbedingungen auf Code zu definieren und zu überprüfen – möglichst in einer deklarativen Art. Auf diese Weise kann man zum Beispiel sicherheitskritische Aspekte des Programmcodes überprüfen.
- *Versioning, Lineage, Traceability:* Daten (und damit auch Programmcode) müssen versioniert sein, so dass es jeder Zeit möglich ist, auf alte Versionen zurückzugreifen. Es muss auch möglich sein, Versionen von Daten und Programmcode zu vergleichen. Es muss möglich sein, die Herkunft und alle Änderungen aller Daten und von Programmcode nachzuverfolgen.
- *Packaged Base Functionality:* Umfangreiche Standardfunktionalität muss automatisch für jede Art von Software bereitgestellt werden. Hierzu gehört eine moderne grafische Benutzeroberfläche sowie zum Beispiel eine Suchmaschine, die innerhalb von Anwendungen und anwendungsübergreifend Daten und Funktionen findet. Software muss plattformübergreifend z.B. auf Mobiltelefonen, Laptops und in eingebetteten Umgebungen ausführbar sein. Anwendungsprogrammierer sollten sich um die unterschiedlichen Plattformen und ihre Anforderungen (Optimierung, Größe des Displays) keine Gedanken machen müssen.
- *Standard, Hype:* Es gibt keinen Grund, das Rad neu zu erfinden. Standards für Datenaustauschprotokolle wie zum Beispiel HTTP, HTTPS, SMTP und SMS können weiterhin verwendet werden. Auf der Anwendungsebene wird es ebenfalls eine weitere Tendenz zur Standardisierung geben. Die Erwartung ist allerdings, dass hier nicht ein komplettes ERP-System (wie z.B. SAP R/3) als Standard aufgefasst wird, sondern dass wesentlich kleinere Funktionsblöcke standardisiert werden. Besonders wichtig werden Standards zum Austausch von Daten und Business-Objekten wie zum Beispiel XBRL (eXtensible Business Reporting Language) für den Finanzbereich [XBRL] oder Health Level 7 im Gesundheitswesen [HL7]. Diese Standards müssen flexibel und offen genug sein, um Innovationen bei Prozessen auf diesen Business-Objekten zu erlauben.
- *Turing Complete:* Obwohl die meisten Benutzer nur kleinere Anpassungen an Software durchführen, muss das gesamte Modell letztendlich beliebige Arten von Anpassungen erlauben.
- *Cheap:* Alle diese Anforderungen dürften letztendlich dazu führen, dass Software preiswerter wird, weil Teile der Programmierung und damit auch Wartung und Qualitätskontrolle auf den Benutzer oder Spezialisten übertragen werden. Die Kosten zur Bereitstellung einer solch flexiblen Softwareinfrastruktur sollten pro Benutzer im Bereich der Kosten für ein Betriebssystem wie Windows liegen. Aktuelle Trends (z.B. hochskalierbares server-side Computing [Hamilton

2007]) müssen ausgenutzt werden und die Kostenvorteile müssen weitergereicht werden.

- *Platform-Independent:* Es darf keine Monopolisierung geben. Es muss mehrere Plattformanbieter geben, die gegenseitig in Konkurrenz stehen und somit die Plattformen und die Möglichkeiten zur Mass Customization von Software kontinuierlich verbessern.
- *Data Spaces:* Trotz feingranularer Erweiterungen von beliebigen Anwendungen muss es möglich sein, anwendungs- und geräteübergreifend auf Daten zuzugreifen. Schlüsselwortsuche (z.B. Desktop-Suche mit Apple Spotlight oder Google Desktop) ist ein Beispiel für eine solch übergreifende Suche. Mächtigere Anfragemöglichkeiten – inklusive strukturierter Suche mit Anfragesprachen wie SQL oder XQuery – werden notwendig sein, um konsolidierte Sichten auf den gesamten Datenraum eines Benutzers oder einer Organisation zu ermöglichen. Diese Forderung wurde bereits in einigen Forschungsprojekten konkretisiert [Dittrich et al. 2005, Franklin et al. 2005].

4.2 Nicht-Anforderungen

Die meisten Leser werden nichts gegen die im letzten Abschnitt aufgeführten Anforderungen einzuwenden haben. Manche Punkte mögen wichtiger erscheinen als andere, aber keine dieser Forderungen tut weh, sofern sie denn erfüllt ist. Doch natürlich haben alle diese Anforderungen ihren Preis und über den kann man derzeit nur spekulieren. Um eine etwas kontroverse Perspektive auf diese Anforderungen zu geben, enthält die folgende Liste einige »Non-Requirements«, also die Punkte, die in der Aufzählung des vorherigen Abschnittes bewusst weggelassen wurden. Hier stehen einige Punkte drauf, die gerade der Datenbank-Community heilig sind:

- *Optimal Performance:* Es wird nicht wichtig sein, die beste Performanz zu erzielen. In Bezug auf Performanz müssen nur zwei Kriterien erfüllt sein. Erstens müssen Antwortzeiten und Durchsätze für den Benutzer akzeptabel sein. Zum Beispiel ist eine Antwortzeit von einer Sekunde pro Mausklick akzeptabel für eine interaktive Anwendung – alles was schneller als eine Sekunde ist, ist unnötiger Luxus. Zweitens müssen alle Algorithmen so formuliert werden, dass sie linear mit der Anzahl der verfügbaren Maschinen (oder Prozessorkernen) skalieren.
- *Security, Availability, Consistency etc.*: Was für die Performanz gilt, gilt auch für andere nicht funktionale Eigenschaften. Natürlich wird es viele Anwendungen geben, für die Sicherheit kritisch ist. Für solche Anwendungen muss es auch möglich sein, ein solches Maß an Sicherheit zu gewährleisten. Doch das heißt nicht, dass alle Anwendungen mit einem solch hohen Maß an Sicherheit laufen müssen. Die so genannten »ilities« müssen für den Benutzer transparent und individuell anpassbar werden – insbesondere müssen die Kosten der »ilities« transparent werden.

  Konsistenz ist eine Eigenschaft, die der Datenbank-Community besonders ans Herz gewachsen ist. Doch auch Konsistenz ist ein Luxus, den sich nicht jede Anwendung leisten kann. Deswegen sind ACID-Transaktionen kein heiliges Dogma mehr, auch wenn sie schon wegen ihrer Einfachheit halber für viele Anwendungen relevant bleiben. Es muss allerdings möglich sein, auch andere alternative Konsistenzmodelle wie ein Plug-in in einer Anwendung zu verwenden.
- *Shared Database Integration:* Der große Erfolg von Datenbanksystemen und Anwendungssystemen wie SAP R/3 basiert darauf, dass die Datenbank als Integrationsplattform dient. Neue Funktionalität wird integriert, indem das Datenbankschema erweitert wird. Diese Vorgehensweise vereinfacht die Implementierung von »Data Spaces« und konsolidierter Sichten über die gesamte Datenbank hinweg. Doch diese Vorgehensweise skaliert nicht, weil mit zunehmender Komplexität die Erweiterung des Datenbankschemas immer schwieriger wird. Bei einer skalierbaren Integration von unterschiedlichen Komponenten (oder Funktionsbausteinen) nach dem Prinzip des Information Hiding findet das Data Sharing über Kommunikation und nicht über einen gemeinsamen Zugriff auf dieselbe Datenbank statt. Genau aus diesem Grund wird die Rolle der klassischen (SQL-)Datenbanktechnologie mit fixen Schemata in Zukunft abnehmen.

5 Web 2.0

5.1 Architektur

Nicht überraschend wird das Web eine große Rolle bei der Mass Customization und der Einbeziehung des Benutzers in die Softwareentwicklung spielen. Die Autoren verwenden hier den Begriff des Web 2.0 [O'Reilly 2005], weil dieser Begriff im Jahr 2006 genügend Aufmerksamkeit erregt (675 Millionen Treffer bei Google zum Zeitpunkt des Verfassens dieses Artikels im August 2006) und unspezifisch genug ist, um hinreichend Interpretationsspielraum zu bieten. Derzeit umfasst der Begriff alles von Ajax bis Mashups im wilden Durcheinander. Hier also die Geschichte des Web, wobei Phase 4 Web 2.0 entspricht:

- *Phase 1 – Publishing Content:* Das Web ist groß geworden, weil es niedrige Einstiegshürden hatte und jedem ermöglicht hat, Inhalte zu publizieren. Im Wesentlichen kann jedes Individuum seine eigene Webseite pflegen. MySpace.com, YouTube.com und Flickr.com sind die aktuellen Plattformen, um Inhalte auf dem Web zu veröffentlichen. Das Ziel, sechs Milliarden potenzielle Verleger zu haben, ist also bereits erreicht.
- *Phase 2 – Collaborative Publishing of Content:* Nach dem Erzeugen der eigenen Webseite versucht ein Web-Junkie seine Webseite populär zu machen. Dazu beteiligt er sich an Communities und arbeitet mit anderen an größeren und mächtigeren Inhalten. Wikipedia ist der Inbegriff des kollaborativen Publizierens von Inhalten auf dem Web und funktioniert erstaunlich gut.
- *Phase 3 – Publishing Services:* Der jüngste Hype ist das Publizieren von Softwarediensten auf dem Web. Dieser Hype wird aktuell mit den Schlagworten »Software as a Service«, »Software on demand« oder »Hosted Services« umschrieben. Die bekannteste kommerzielle Erfolgsgeschichte ist Salesforce.com.
- *Phase 4 – Collaborate Publishing of Services (Web 2.0):* Die logische

Konsequenz aus den ersten drei Phasen und aus den im Abschnitt 2 aufgeführten gesellschaftlichen Trends und Anforderungen ist die kollaborative Entwicklung von Software auf dem Web und die individuelle Anpassung von Software jedes Einzelnen für die eigenen Bedürfnisse und Ideen. Der derzeitige Hype im Bereich Mashups [Programmableweb] lässt erahnen, welche Anfänge diese vierte Phase bringen kann.

Die Technologie für Phasen 1 bis 3 ist bekannt und ist im Wesentlichen wie in Abbildung 1 (Web 1.0) gezeigt HTML und HTTP. Für Hosted Services braucht man noch Web-Services und auf der Serverseite Programmiersprachen wie PHP oder Java und SQL. Technologisch sind also die Phasen 1 bis 3 der »State of the Art«.

Die Technologie für Phase 4 wird der Gegenstand der Forschung vieler Teilgebiete der Informatik der kommenden zehn Jahre sein. Abbildung 2 zeigt eine mögliche Architektur von Web 2.0. Im Vergleich zur Architektur von Web 1.0 gibt es zwei wesentliche Änderungen:

- Die Rolle des Menschen und des Browsers wird erweitert. Der Mensch liest und editiert nicht nur Content, sondern der Mensch liest und editiert Prozesse oder Dienste in Form von Skripten. Diese Skripten werden mit grafischen Werkzeugen erstellt. Als Beispiel soll hier XQueryP als mögliche Skriptsprache für solche Zwecke aufgeführt werden [Chamberlin et al. 2006] (viele andere Vorschläge für Protokolle und Sprachen werden folgen).
- Die Beziehung zwischen Mensch und Maschine (zwischen Client und Server) wird symmetrischer. Sowohl Client als auch Server können der Ursprung von Content und Services sein.[1]

Diese beiden Punkte werden in den folgenden Abschnitten noch detaillierter ausgeführt. Für den Moment soll der Augenmerk auf die Gemeinsamkeiten zwischen den Abbildungen 1 und 2 gelegt werden. Die grundlegenden Protokolle (HTTP, HTTPS, SMTP, SMS usw.) bleiben dieselben. In der Tat gibt es im Solr-Projekt [Delacretaz 2006] bereits Bemühungen, aktive Daten via HTTP zu transportieren. Der Internet-Backbone, der durch Caching und andere Mechanismen das Internet skalierbar macht, bleibt auch erhalten. Architekturprinzipien wie lose Kopplung (Service-Oriented Architectures – SOA [Wikipedia]), die heute schon populär sind, werden ebenfalls zur Anwendung kommen. Der Unterschied liegt in der Art, wie man auf diesen Technologien aufsetzt.

1. In der Literatur redet man hier auch häufig von Peer-to-Peer-Architekturen. Der Begriff wird wegen gewisser technischer Konnotationen, die er weckt, hier vermieden.

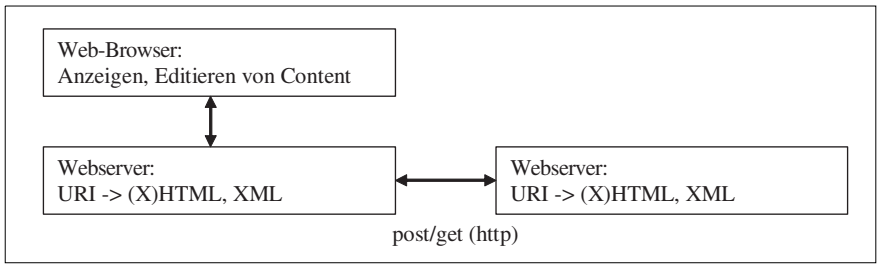

Abb. 1: Architektur von Web 1.0

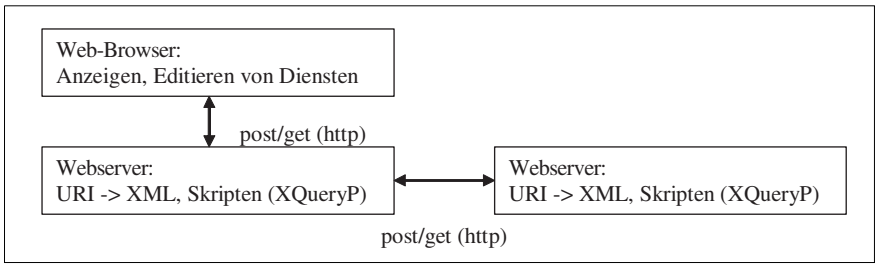

Abb. 2: Architektur von Web 2.0

5.2 Bausteine

Wie gesagt kann dieser Artikel keine Lösungen präsentieren. Dieser Artikel soll lediglich zur Diskussion anregen. Im Folgenden sollen einige aktuelle Technologien aufgezählt werden, die wesentlich zur Entwicklung von Web 2.0 beitragen könnten. Diese Technologien kommen aus verschiedenen Gebieten der Informatik. Auf eine besondere Technologie, XML, wird aufgrund des thematischen Bezugs zu Informationssystemen in Abschnitt 6 genauer eingegangen.

- *Declarativity, Programming with Meta-data:* Funktionale Programmiersprachen, für die es grafische Editoren gibt, werden an Bedeutung gewinnen. Außerdem wird das Verhalten der Software viel stärker durch die Spezifikation von Metadaten bestimmt. Im Microsofts Avalon Framework kann man zum Beispiel schon mächtige Benutzerschnittstellen mit Interaktionen mit Web-Services in einer XML-Datei (XAML-Format) spezifizieren.
- *One data model:* Da die Grenzen zwischen den einzelnen Schichten einer Softwarearchitektur verwischen werden, wird es auch nur ein Datenmodell für die gesamte Software geben. Insbesondere wird die Aufteilung zwischen relationalem Datenmodell für die persistente Datenhaltung, objektorientierten Datenmodellen für die Programmierung von Anwendungslogik und semistrukturierten Datenmodellen für Kommunikation entfallen und ersetzt werden durch ein einziges Datenmodell, das mächtig genug ist, die Bedürfnisse aller Schichten zu befriedigen.
- *Information Hub:* Datenbanken werden in Zukunft zwei wesentliche Änderungen erfahren. Erstens werden sie sehr viel leichtgewichtiger in der Installation, Administration und im Betrieb werden. Sie werden nicht mehr die Softwarearchitektur bestimmen, sondern sich der Softwarearchitektur anpassen; das heißt, sie werden ein unsichtbarer und integraler Bestandteil der Infrastruktur werden. Zweitens werden Datenbanken sowohl Quellen als auch Senken von Datenströmen und Events sein: Datenbanken (genauer der komplette Information Hub) werden sowohl

ein »pull-orientiertes« als auch ein »push-orientiertes« Interface bereitstellen.

- *SOA, $O(n^2)$ Datenintegration:* SOA wird sich durchsetzen. Das heißt, man wird immer weniger versuchen, unterschiedliche Anwendungen in ein gemeinsames globales Schema zu integrieren. Stattdessen werden (relativ gesehen) immer kleinere Systeme mit immer mehr anderen kleinen Systemen kommunizieren, weil diese Architektur der in Abschnitt 2 beschriebenen Organisation der realen Welt entspricht. Diese Vorgehensweise führt zu einer Explosion der Anzahl der Schnittstellen: im schlimmsten Fall quadratisch in der Anzahl der Systeme anstelle des Ideals eines linearen Wachstums der Komplexität bei einer sternförmigen, kontrollierten Integration über ein Datenbanksystem. Man wird lernen, mit dieser Explosion an Schnittstellen umzugehen, indem man auf einen Information Hub zur Verwaltung dieser Schnittstellen vertraut. UDDI ist ein erster Schritt in diese Richtung [UDDI].
- *Commoditization:* Vermutlich der wichtigste Erfolgsfaktor wird sein, dass einfach versucht wird, viel mehr Software zu bauen. »Try and error« und »learning by doing« sind vermutlich immer noch die wichtigsten Erfolgsprinzipien der Informatik. Die Informatik steckt noch in den Kinderschuhen und ist noch weit davon entfernt, eine Wissenschaft zu sein, die Innovation durch rigorose Methoden erreichen kann. Während dieser Entwicklung müssen Fortschritte sehr schnell zur Commodity werden. Ohne kontinuierliche Innovation müssen die Gewinnmargen von Softwaregiganten wie Microsoft oder SAP auf das Maß einer Commodity-Industrie wie zum Beispiel dem Handel mit Gummibärchen oder der Produktion von Shampoo schrumpfen.

6 Wieso XML?

Bisher hatte dieser Artikel wenig mit »Datenbanken« zu tun. Die Datenbank-Community hält aber wohlmöglich einen entscheidenden Schlüssel in der Hand: XML. XML wurde nicht in der Datenbank-Community entwickelt, sondern ist in der heutigen Form aus Beiträgen von unterschiedlichen Communities (federführend war die Dokumentenmanagement-Community) entstanden. Der Beitrag der Datenbank-Community lag im Wesentlichen in der Entwicklung von Datenmodellen für XML (zum Beispiel semistrukturierte Datenmodelle oder das aktuelle XQuery-Datenmodell [Fernandez et al. 2006]) und der Entwicklung von deklarativen Sprachen zur Verarbeitung von XML-Daten (zum Beispiel der aktuelle XQuery-Standard [Boag et al. 2006]). Des Weiteren liefern die großen Hersteller derzeit skalierbare XML-Datenbanken, was dem Durchbruch von XML als Datenformat helfen wird.

XML könnte ein entscheidender Baustein für die Realisierung von Mass Customization in der Softwareindustrie werden, weil XML einige der Anforderungen aus Abschnitt 4.1 direkt berücksichtigt (z.B. Erweiterbarkeit und Plattformunabhängigkeit) und weil XML die Grundlage für einige Technologien aus Abschnitt 5.2 ist (z.B. deklarative Programmierung mit Metadaten und SOA). Im Folgenden soll zunächst das einleitende Zitat von Roger Bamford erläutert werden, wieso die heutige Generation von SQL-Datenbanksystemen ungeeignet ist. Danach soll etwas genauer erklärt werden, wieso gerade »XML« helfen kann. In Abschnitt 6.3 wird auf die Prognose aus Abschnitt 1 eingegangen und gefragt, was XML mit dieser Prognose zu tun hat.

6.1 Wieso nicht relationale Daten?

Die Antwort ist, dass relationale Datenbanken nicht flexibel genug sind, um die Anforderungen aus Abschnitt 4.1 zu erfüllen: Sie machen zu viele Vorgaben, die Benutzer und Entwickler zu sehr einschränken. Diese Einschränkungen liegen teilweise am relationalen Datenmodell und seiner Standardanfragesprache SQL. Teilweise sind diese Einschränkungen auch in der Architektur heutiger relationaler Datenbanksysteme begündet, die auf die Architektur der kompletten Softwareanwendung, die auf einem solchen System aufsetzt, ausstrahlt. Die Hersteller von relationalen Datenbanksystemen versuchen, diese Einschränkungen dadurch zu kompensieren, dass sie die Funktionalität der Datenbanksysteme ständig erweitern und somit ihren Kunden entgegenkommen. Tatsächlich machen sie damit das Problem nur noch größer, weil sie immer stärkere Abhängigkeiten schaffen und somit immer größere Monolithen erzeugen. Die folgende Aufzählung zeigt auf, welche weitreichenden Konsequenzen die *Diktatur* der relationalen Datenbanken hat:

- *Software-Engineering-Prozess:* Relationale Datenbanken legen fest, dass man erst ein Schema definieren muss, bevor man Daten in der Datenbank halten kann. Analog können in Java keine Objekte ohne eine entsprechende Klassendefinition existieren. Manche Menschen möchten jedoch erst einmal Daten sammeln und sich dann Gedanken machen, was sie mit diesen Daten anfangen. Der Aufwand, die Daten zu strukturieren, bevor sie gesammelt werden, ist in solchen Situationen nicht akzeptabel.
- *Softwarearchitektur:* Die heutige Generation relationaler Datenbanksysteme verlangt, dass Daten erst in das Datenbanksystem geladen werden müssen, bevor man sie verarbeiten darf. In vielen Anwendungen besteht jedoch das Bedürfnis, Daten »on-the-fly« zu verarbeiten und der Aufwand, erst durch die Schichten der von der relationalen Datenbank vorgegebenen Architektur zu gehen, erscheint zu hoch, sowohl in Hinsicht auf Performanz als auch Programmieraufwand.
- *Evolution:* SQL und relationale Datenbanken erlauben nur eine grobgranulare Evolution von Daten: Es wird entweder die Struktur aller oder keines Objektes einer Tabelle angepasst.
- *Quality of Service:* In einer Schichtenarchitektur bestimmt das schwächste Glied die Sicherheit, Performanz, Verfügbarkeit, Vorhersagbarkeit usw. der kompletten Anwendung. Unglücklicherweise ist trotz großer Anstrengungen der Datenbankhersteller das Datenbanksystem häufig eben dieses schwächste Glied. Viel schlimmer ist, dass die heutige Generation von Datenbanksystemen keinerlei Flexibilität erlaubt: Zum Beispiel mehr Sicherheit für weniger Performanz einzutauschen ist mit einem heutigen Datenbanksystem nicht möglich. Die heutige Generation an Datenbanksystemen deckt nur einen Punkt im weiten Spektrum an möglichen QoS-Anforderungen ab.

6.2 Vorteile von XML

XML vermeidet die im vorherigen Abschnitt aufgeführten Einschränkungen von aktuellen relationalen Datenbankprodukten. Ein Grund ist, dass XML noch jung ist und einige architektonische Fehler (noch) nicht bei der Implementierung von XML-Datenverarbeitungssystemen gemacht wurden. Andere Gründe und Vorteile liegen allerdings immanent in XML drin.

Streng genommen versteht man unter XML einen Standard zur plattformunabhängigen Serialisierung von Daten mittels spitzer Klammern [Bray et al. 2006]. XML ist eine Untermenge von SGML, und der entscheidende Unterschied zu SGML liegt darin, dass XML wesentlich einfacher ist, aber für die meisten Anwendungen eine hinreichende Mächtigkeit besitzt. Offensichtlich ist die Möglichkeit, Daten (und somit Business-Objekte) plattformunabhängig zu serialisieren und damit auch unabhängig von einer Datenbank zu speichern und zu verarbeiten, ein entscheidender Vorteil von XML zur Realisierung von Software Mass Customization (Abschnitt 4). Ansonsten könnte eine lose Kopplung mittels einer serviceorientierten Architektur (SOA) gar nicht implementiert werden. Dieser Abschnitt versucht jedoch, drei etwas subtilere Vorteile von XML, die bisher in der Literatur nicht so ausführlich diskutiert wurden, herauszuarbeiten. Diese Vorteile sind nicht XML-spezifisch, und es ist strittig, ob man sie überhaupt als Kerneigenschaften von XML ansehen darf. Doch der XML-Technologiestack ist derzeit der einzige Technologiestack, der diese Eigenschaften vereinigt.

6.2.1 Trennung von Schema und Daten

Die erste Eigenschaft ist, dass XML Daten von ihrer Interpretation und ihrem Schema trennt. Diese Eigenschaft kann man in keiner anderen heutigen Softwaretechnologie finden. Zum Beispiel können Daten in eine relationale Datenbank erst geschrieben werden, nachdem man die Tabellen (das Schema) für die Daten angelegt hat. Die Daten leben auch nur so lange wie die Tabellen. Analog ist die Existenz von Java-Objekten an die entsprechenden Java-Klassen gebunden. XML-Dokumente können unabhängig von ihrem Schema existieren und man kann dasselbe XML-Dokument mit unterschiedlichen Schemata »interpretieren«. Das heißt, je nach Schema erhält das XML-Dokument eine andere Bedeutung.

Diese Eigenschaft von XML ist aus zwei Gründen bedeutend. Zum einen erlaubt diese Eigenschaft eine Abweichung von der klassischen Software-Engineering-Methodik, in der Entwickler zuerst ein Datenmodell entwickeln, bevor sie irgendetwas implementieren können. Mit XML können – wie in den vorherigen Abschnitten mehrfach gefordert – die strengen Grenzen zwischen den einzelnen Phasen der Softwareentwicklung besser verwischt werden.

Zum anderen erlaubt die Trennung von Daten und Schema, dass unterschiedliche Anwendungen (insbesondere »customized applications«) auf dieselben Daten zugreifen. Genau aus diesem Grund ist XML so beliebt geworden zur Integration von Anwendungen und als Datenaustauschformat zwischen Anwendungen. Des Weiteren wird aus diesem Grund XML so bevorzugt zur Archivierung von Daten verwendet. Es ist nicht sicher, ob es in zweitausend Jahren noch Microsoft Word als Anwendung gibt. Es gibt jedoch eine berechtigte Hoffnung, dass man in zweitausend Jahren noch das XML-Format der neusten Version von Word wird lesen können (sofern die Word-Dokumente permanent auf moderne Hardware umkopiert wurden).

6.2.2 Einheitliche Repräsentation von unstrukturierten, strukturierten und gemischten Daten

XML hat das Potenzial, die »Mutter aller Datenformate« zu werden. Kein anderes heute bekanntes Datenformat erlaubt die Repräsentation aller Arten von Daten: unstrukturierte Daten wie Text oder Multimediadaten, strukturierte Daten wie zum Beispiel Kontoinformation und gemischte Daten wie zum Beispiel Text mit Annotationen. Diese Eigenschaft macht XML zum potenziellen Kandidaten für ein einheitliches Datenformat aller Daten, eine wesentliche Forderung zur Reduzierung der Komplexität heutiger Software. Hierdurch werden auch XML-Technologien (zum Beispiel XQuery [Boag et al. 2006]) natürliche Kandidaten, um eine unstrukturierte Suche à la Google mit einer strukturierten Suche à la SQL zu veredeln. Wäre es nicht schön, wenn man in Google nach dem Supermarkt im Raum Zürich mit dem besten Sonderangebot für Lindt-Schokolade fragen könnte? Die Eigenschaft, das gesamte Spektrum von unstrukturierten zu strukturierten Daten repräsentieren zu können, macht XML auch attraktiv zur Speicherung von Ergebnissen von wissenschaftlichen Experimenten. Am Anfang werden Messdaten (zum Beispiel eine Genomsequenz) unstrukturiert sein. Mit zunehmender Forschung und Analyse der Messdaten wird man mehr Strukturen (Annotationen) hinzufügen. Bei wertlosen Daten (zum Beispiel Vorlesungsmitschriften eines Studierenden) wird man sich vielleicht nie die Mühe machen, Strukturen hinzuzufügen. Diesen Ansatz nennt man »pay as you go along« und er entspricht der menschlichen Erwartungshaltung, wie man mit Daten umgeht: In wertvolle Daten investiert man – wertlose Daten lässt man einfach herumliegen und hofft, dass sie nicht im Weg stehen.

6.2.3 Einheitliche Repräsentation von Daten, Metadaten und Programmcode

Wie in Abschnitt 4 aufgeführt, sind Metadaten und Programmcode auch Daten und sollten als solche behandelt werden. Die Grenzen zwischen den unterschiedlichen Arten von Daten verwischen. Als Beispiel soll XMI dienen [XMI 2005]. XMI ist die Repräsentation von Modellen (z.B. UML-Klassendiagrammen) in der Eclipse-Plattform. Aus Sicht von Eclipse sind XMI-Dateien ganz gewöhnliche Daten, die in Eclipse verarbeitet und visualisiert werden. Aus Sicht eines Programmierers sind XMI-Dateien Metadaten, die ein Softwaresystem beschreiben und bei der Evolution des Softwaresystems oder anderen Aufgaben helfen. In jüngster Zeit wurde verstärkt an »executable UML« geforscht [Mellor & Balcer 2002]. Aus dieser Sicht sind XMI-Dateien ausführbarer Programmcode. Erneut ist XML das derzeit vielversprechendste Datenformat, um eine einheitliche Repräsentation für diese unterschiedlichen Sichtweisen auf Daten zu schaffen.

6.3 Wieso nicht SQL+Java+XML?

Eine mögliche Lösung, um die Vorteile von XML auszunützen, liegt scheinbar in der Formel »SQL+Java+XML«. Die Hoffnung ist, die Vorteile aller drei Tech-

nologien zu vereinbaren. Tatsächlich wird dieser Ansatz in der Industrie aktuell propagiert, indem fast alle gängigen Programmiersprachen (inklusive SQL und Java) um XML-Schnittstellen erweitert werden [Florescu & Kossmann 2006]. Diese Lösung bietet den Vorteil, dass man das Rad nicht neu erfinden muss. Allerdings ist bei einer solchen Vermischung unterschiedlicher Technologien und Datenmodelle mit Reibungsverlusten zu rechnen. Der berühmte Impedance Mismatch zwischen SQL und objektorientierten Programmiersprachen wie Java wird durch die Hinzunahme von XML-Daten noch verschärft.

Die Alternative zu »SQL+Java+XML« wären – gemäß der Prognose aus Abschnitt 1 – ein oder mehrere neue Programmiermodelle. Zur effizienten und robusten Implementierung von neuen Programmiermodellen würden natürlich wesentliche Komponenten und Erfahrungen der heutigen Softwarelandschaft übernommen werden. Neue Programmiermodelle bergen Chancen (z.B. Aufräumen mit Altlasten, Bereitstellung eines höheren Abstraktionsniveaus u.a. für Kommunikation und Datenhaltung) und Gefahren (z.B. Schulung von Programmierern, Kinderkrankheiten). Ob solche Modelle tragbar sind, können wir heute nicht beantworten. Doch wir sind der Meinung, dass es Aufgabe der Wissenschaft ist, zumindest mit solchen neuen Modellen zu experimentieren und in Zusammenarbeit mit der Industrie über Implementierungen und Migrationspfade nachzudenken.

7 Fazit

Dieser Artikel begann mit einem Zitat und soll mit einem Zitat von Bill Gates enden: »Man überschätzt den Fortschritt der nächsten zwei Jahre. Man unterschätzt den Fortschritt der kommenden zehn Jahre.« Alle Aussagen dieses Artikels sind mit Vorsicht zu genießen. Aus Sicht der Datenbank-Community macht dieser Artikel vier wesentliche Aussagen:

- Software Mass Customization wird die Softwareindustrie und insbesondere die Rolle von Datenbanksystemen grundsätzlich verändern. Mass Customization ist keine technische Notwendigkeit, sondern ein wirtschaftlicher und gesellschaftlicher Trend, der sich auch die Softwareindustrie unterordnen muss.
- Datenbanken werden in größere IT-Infrastrukturen aufgehen. Die Datenbank wird unsichtbar. Das heißt, dass es in Anwendungsprogrammen keine sichtbare Datenbankschnittstelle mehr geben wird. (Dies wird auch für Kommunikationsschnittstellen gelten.)
- Die heutige Generation hat einen wesentlichen technologischen Schlüssel zur Bereitstellung der IT-Infrastruktur der nächsten Generation in der Hand. Nicht zufällig sind bereits heute die großen Datenbankhersteller auch Marktführer im Bereich Middleware. Sie werden auch ein bedeutendes Wort bei der »Post-Middleware«-Infrastruktur mitsprechen.
- Datenmodellierung wird ein zeitloses Problem der Informatik bleiben. Auch in zehn Jahren wird die Informatik in den Kinderschuhen stecken, und es wird jede Menge in der Forschung und Anwendung zu tun geben.

Die große, generelle Vision liegt darin, die Komplexität von Software zu reduzieren. Dies wird auf zwei Ebenen versucht:

a. Es muss eine höhere Abstraktionsstufe gefunden werden, die von den technologischen Eigenheiten heutiger Softwarearchitekturen abstrahiert (z.B. Kommunikation, Persistenz, Sicherheit, Verteilung). Hierzu sind die aktuellen Programmiermodelle und Softwarearchitekturen ungeeignet.
b. Die Welt ist kompliziert und somit wird auch realistische Software inhärent komplex sein. Diese Komplexität kann man nur beherrschen, indem man sie zu den Anwendern schiebt – also zu denen, die diese Komplexität verstehen. Genau das ist die Idee von Mass Customization, die in diesem Artikel für die Softwareindustrie propagiert wird. Zumindest erspart man sich damit den Stille-Post-Effekt in der Kommunikation von Anwendern zu Softwareentwicklern. Outsourcing ist eindeutig die falsche Lösung, da es diesen Stille-Post-Effekt verschärft. Mass Customization könnte man auch als eine besonders aggressive Form des Insourcings bezeichnen. Die IT-Abteilung wandert nicht zu IBM, Accenture oder nach Indien, sondern sie wandert zur Fachabteilung.

Den Autoren ist bewusst, dass viele Aussagen kontrovers sind und wesentlich differenzierter betrachtet werden müssen. Das Ziel dieses Artikels ist es jedoch, eine Diskussion in Gang zu setzen, und dafür ist eine solch undifferenzierte Sichtweise häufig besser. Die Autoren freuen sich auf Feedback.

Danksagung

Einer der Autoren ist derzeit im Rahmen eines Forschungsfreisemesters an der Stanford University. Dieses Umfeld hat diesen Artikel sehr geprägt. Die Autoren möchten sich bei folgenden Personen für die vielen Diskussionen und Ideen bedanken: Roger Bamford (Oracle), Adam Bosworth (Google), Manfred Broy (TU München), Daniela Florescu (Oracle), Dieter Gawlick (Oracle), Hector Garcia-Molina (Stanford), Daniel Kossmann (Unilever), Alexander Kreutz (i-TV-T), Stefan Pröls (i-TV-T), Jennifer Widom (Stanford).

Referenzen

[Boag et al. 2006] *Boag, S.; Chamberlin, D.; Fernandez, M.; Florescu, D.; Robie, J.; Simeon, J.:* XQuery 1.0: An XML Query Language. W3C Candidate Recommendation. Juni 2006.

[Bray et al. 2006] *Bray, T.; Paoli , J.; Sperberg-McQueen, C.; Maler, E.; Yergeau, F.:* Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation. August 2006.

[Chamberlin et al. 2006] *Chamberlin, D.; Carey, M.; Florescu, D.; Kossmann, D.; Robie, J.:* Programming with XQuery. Proc. of XIME-P Workshop, Chicago, USA, Juni 2006.

[Cowan & Tobin 2004] *Cowan, J.; Tobin, R.:* XML Infoset 2nd Edition. W3C Recommendation. Februar 2004.

[Delacretaz 2006] *Delacretaz, B.:* Solr: Indexing XML with Lucene and REST. August 2006; *www.xml.com/pub/a/2006/08/09/solr-indexing-xml-with-lucene-andrest.html.*

[Dittrich et al. 2005] *Dittrich, J.; Vaz Salles, M.; Kossmann, D.; Blunshi, L.:* iMeMex: Escapes from the Personal Information Jungle (Demo). Proc. of Int. VLDB Conf., Trondheim, Norwegen, August 2005.

[Fernandez et al. 2006] *Fernandez, M.; Malhorta, A.; Marsh, J.; Nagy, M.; Walsh, N.:* XDM: XQuery 1.0 and XPath 2.0 Data Model. W3C Candidate Recommendation, Juli 2006.

[Florescu & Kossmann 2006] *Florescu, D.; Kossmann, D.:* Programming for XML (Tutorial). Proc of ACM SIGMOD Conf., Chicago, USA, Juni 2006.

[Franklin et al. 2005] *Franklin, M.; Halevy, A.; Maier, D.:* From databases to dataspaces: A new abstraction for information management. SIGMOD Record, Dezember 2005.
[Friedman 2006] *Friedman, T.:* The World is Flat. Penguin Books Ltd, 2006.
[Hamilton 2007] *Hamilton, J.:* Architecture for Modular Data Centers. Erscheint in: Proc. of CIDR Conf., Asilomar, USA, Januar 2007.
[HL7] Health Level 7; *www.hl7.org*.
[Kimball & Strehlo 1995] *Kimball, R.; Strehlo, K.:* Why Decision Support Fails and How to Fix it. SIGMOD Record, September 1995.
[Mellor & Balcer 2002] *Mellor, S.; Balcer, M.:* Executable UML: A Foundation for Model-Driven Architecture. Addison-Wesley, 2002.
[O'Reilly 2005] *O´Reilly, T.:* What is Web 2.0? Design Patterns and Business Models for the Next Generation of Software. September 2005; *www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html*.
[Programmableweb] *www.programmableweb.org*.
[UDDI] *www.uddi.org*.
[Wikipedia] *www.wikipedia.com*.
[XBRL] *www.xbrl.org*.
[XMI 2005] MOF 2.0/XMI Mapping Specification, v2.1. OMG. September 2005.

Donald Kossmann ist Professor für Informatik an der ETH Zürich. Er ist Mitbegründer der i-TV-T AG in München. Seine Forschungsinteressen liegen im Bereich Datenbanken und Informationssysteme, insbesondere webbasierte Informationssysteme.

Gustavo Alonso ist Professor am Institut für Pervasive Computing, Departement Informatik an der ETH Zürich, wo er die Forschungsgruppe für Informations- und Kommunikationssysteme leitet.

Prof. Dr. Donald Kossmann
ETH-Zentrum
Institut für Informationssysteme
Universitätsstr. 6
CH-8092 Zürich
donald.kossmann@inf.ethz.ch
www.dbis.ethz.ch

Prof. Dr. Gustavo Alonso
ETH Zürich
Departement für Informatik
Haldeneggsteig 4
CH-8092 Zürich
alonso@inf.ethz.ch
www.iks.ethz.ch