Diplomarbeit

Entwurf und Entwicklung eines generischen Content Management Systems für eXtended Business Objects

von
Daniel Bleisteiner
geboren am 04.06.1975

Fakultät IV - Elektrotechnik und Informatik
Technische Universität Berlin

Berlin 2003

Table of Contents

1 Einleitung

Die hier vorliegende Arbeit ist der Entwurf und die Realisierung eines "generischen Content Management Systems für eXtended Business Objects". Dieses generische Content Management System ist ein wesentlicher Bestandteil des "eXtended Business-Objects-Frameworks für eCommerce-Systeme", das in der gleichnamigen Dissertation vorgestellt wird. Die vorliegende Arbeit wurde im Rahmen und in enger Abstimmung mit dieser Dissertation angefertigt und trug wesentlich dazu bei, das dort gesteckte Ziel zu erreichen.

Das Ziel der Dissertation war der Entwurf und die Realisierung eines neuartigen Frameworks, mit dem individuelle eCommerce-Systeme deutlich schneller und kostengünstiger als bisher erstellt werden können. Im Folgenden wird auf die Beweggründe für diese Arbeit detailiert eingegangen.

1.1 Motivation

Das gemeinsame Ziel der Dissertation und dieser Arbeit ist die Entwicklung eines neuartigen Frameworks, des "eXtended Business-Objects-Frameworks für eCommerce-Systeme", der es ermöglichen soll, leistungsfähige individuelle eCommerce-Systeme deutlich schneller und kostengünstiger als es bisher möglich war, entwickeln zu können. Diese Zielstellung und der angestrebte ganzheitliche Lösungsansatz sollen anhand der nachfolgenden Fragestellungen erläutert werden:

Warum individuelle eCommerce-Systeme?

Verglichen mit der Anzahl von existierenden, unterschiedlichen Geschäfts-modellen, die in der real existierenden Wirtschaft zu finden sind, gibt es nur eine verschwindend kleine Zahl von "passenden Standardanwendungen". Die bisher entwickelten Standard-Lösungen im Bereich des eCommerce sind meist

abgeschlossene Anwendungen, die höhere Anforderungen an Ausbaufähigkeit, Flexibilität und Integrierbarkeit in vorhandene IT-Systeme nicht erfüllen können. Nur für die wenigsten Geschäftsmodelle, wie z.B. B2C-Shop-Lösungen existieren überhaupt Standard-Lösungen.
Trotz hoher Kosten wird in den meisten Fällen die Entwicklung von eigenen Lösungen bevorzugt, da Standard-Lösungen nur selten auf die gegenwärtigen und zukünftigen Geschäftsprozesse anwendbar sind. Das moderne Geschäftsleben zeichnet sich durch ein stetiges Anwachsen von Angeboten, globalem Wettbewerb und Kundenerwartungen aus. Der ständig steigende globale Konkurrenzdruck erfordert von den Unternehmen in zunehmendem Maße Leistungen, die über den "Standard" hinausgehen. Als Antwort darauf sind Unternehmen in der ganzen Welt damit beschäftigt, Lösungen zu entwickeln, die über den "Standard" hinausgehen und die es ihnen erlauben, entsprechende Produkte und Dienstleistungen anbieten zu können. Individuelle Lösungen sind dabei auch für Geschäftsbereiche bzw. Geschäftsmodelle gefragt, für die es bereits Standard-Lösungen gibt, besonders dann, wenn diese Standard-Lösungen nur unzureichend erweiterbar und anpassbar sind.
Die Erfolgspotentiale von eCommerce-Anwendungen, wie Zeiteinsparung, Kosteneinsparung, Qualitätserhöhung und die Ausnutzung strategischer Potentiale können nur durch individuelle, auf das jeweilige Geschäftsmodell eines Unternehmens angepasste Lösungen maximiert werden.

Warum ein Framework?

Ein grosser Teil der grundlegenden Entitäten (wie z.B. Kunde, Rechnung, Lieferadresse, Zahlungsadresse, Produkt, Produktkategorie usw.) und ein grosser Teil der Anwendungsfunktionen (wie z.B. Content-Management, Produktpräsentation, Benutzernavigation, Suche, Eingabevalidierung usw.) sind in den meisten Geschäftsmodellen identisch. Durch das Zusammensetzen von fertigen, wiederverwendbaren, grundlegenden Komponenten aus einem "Baukasten" ist man in der Lage, in minimaler Zeit zu einem Basissystem zu gelangen, das dann beliebig ausgebaut und den individuellen Bedürfnissen angepasst werden kann.
Framework- bzw. "Baukasten"-Systeme für eCommerce sind schon von ihrem grundlegenden Konzept her für flexible Erweiterungen ausgelegt. Dies prädestiniert sie für das heutige Geschäftsleben, das durch immer schneller wechselnde Anforderungen und Geschäftsstrukturen geprägt ist. Die erhöhte Flexibilität wirkt sich dabei positiv auf alle heutigen und zukünftigen

Erweiterungs-, Veränderungs- und Integrationsanforderungen aus. Durch die Wiederverwendung von fertigen, erprobten Komponenten werden die Entwicklungszeiten drastisch verkürzt, die Herstellungskosten herabgesetzt und die Zuverlässigkeit und Qualität erhöht.

Warum für eCommerce-Systeme?

Kaum ein anderer Bereich hat in den letzten Jahren eine solch rapide Entwicklung vollzogen wie der eCommerce. Obwohl der Begriff "Electronic Commerce" erst in der jüngeren Geschichte des World Wide Web in den Mittelpunkt des öffentlichen Interesses gerückt ist, existieren bereits heute hunderte von Analysen und Prognosen über die Entwicklungspotentiale von eCommerce. In der folgenden Abbildung ist eine der zahlreichen Prognosen für die Entwicklung des Weltmarktes für eCommerce dargestellt.

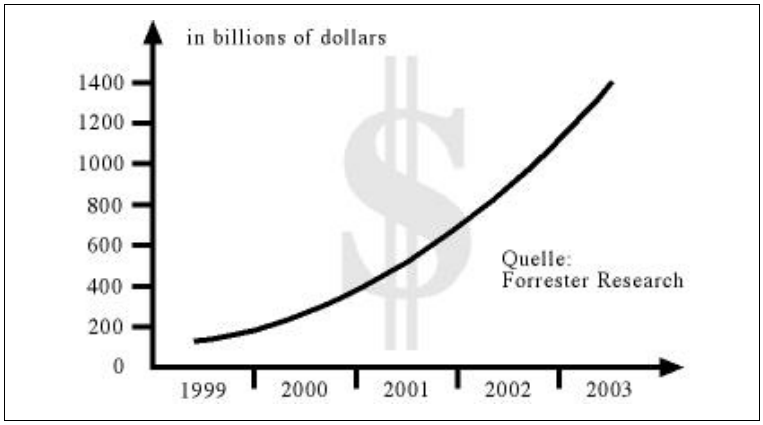

Abb 1: Prognostizierter Weltmarkt für eCommerce

Viele der Prognosen haben eine Gemeinsamkeit. Sie sagen dem eCommerce nicht nur eine gute, sondern eine sehr gute Zukunft voraus. Eine Zukunft, in der mit und durch den eCommerce exponentielle Umsatzsteigerungen zu erwarten sind. Eine Zukunft in der der gesamte Bereich des eCommerce als eine treibende Kraft, nicht nur für die Entwicklung der Software-Industrie, sondern für die Weiterentwicklung der gesamten Weltwirtschaft angesehen wird.
Darüberhinaus wird eCommerce mittlerweile von vielen Nationen als ein wesentlicher Schlüssel für die internationale Wettbewerbsfähigkeit betrachtet. Die wirtschaftlichen und gesellschaftlichen Auswirkungen des Internet im Allgemeinen und des Electronic Commerce im Speziellen sind so weit- und

tiefgreifend, dass die Entwicklung von elektronischen Lösungen ein wesentlicher Zukunftsfaktor bzw. sogar ein überlebensnotwendiger Faktor für die Mehrheit der existierenden Unternehmen darstellen wird.
In der Praxis existieren verschiedenste technologische, gesellschaftliche und wirtschaftliche Eintrittsbarrieren, die vor allem die kleinen und mittelständigen Unternehmen bisher daran hindern, die vorhandenen Potentiale des eCommerce nutzen zu können. Die wichtigsten von ihnen sind zu lange Entwicklungs- und Einführungszeiten und die damit verbundenen zu hohen Kosten bei der Einführung von eCommerce-Systemen. Die damit einhergehende lange Zeitspanne bis zur Amortisation von Investitionen in Rationalisierungs-maßnahmen durch eCommerce-Lösungen stellt vor allem für kleine und mittlere Unternehmen ein meist unzumutbares Risiko dar.

Warum Business-Objects?

Die Betrachtung von Geschäftsobjekten, von Objekten also, die ihre Entsprechung in der real existierenden Geschäftswelt haben, ermöglicht die Annäherung zwischen der betriebswirtschaftlich anwendungsbezogenen Sichtweise von Auftraggebern und Managern und der technisch implementierungsbezogenen Sichtweise von Entwicklern.
Eine schnelle und korrekte Umsetzung von Anforderungen aus der Geschäftswelt in einsatzfähige eCommerce-Systeme erfordert - heute mehr denn je - eine enge Zusammenarbeit von Software-Ingenieuren und Experten aus den jeweiligen Geschäftsbereichen.
Durch die Betrachtung von Geschäftsobjekten können geschäftliche Konzepte, Prozesse und Elemente so beschrieben werden, dass sie von beiden Welten gleich gut und eindeutig verstanden werden. Auf diese Weise helfen Geschäftsobjekte dabei, Fehler in der Design- und Planungsphase von eCommerce-Systemen zu minimieren, woduch die Grundlage für eine schnellere und korrekte Umsetzung von Geschäftsmodellen in elektronische Lösungen geschaffen wird.

Warum eXtended Business-Objects?

Die in dieser Arbeit vorgestellten eXtended Business Objects (eXBO) erweitern die Konzeption von herkömmlichen Business-Objects in zwei wesentlichen Punkten:

eXtended Business-Objects besitzen zusätzliches "Wissen" über ihre Darstellung. Dies ermöglicht den Aufbau von eCommerce-Systemen, die gleichzeitig für eine Vielzahl von verschiedensten Endgeräten spezialisiert werden können.
Die eXtended Business-Objects besitzen zusätzliches "Wissen" über ihre Bearbeitung. Dies ermöglicht es ihnen mit sogenannten "eXBO-Facility-Komponenten" bearbeitet zu werden. Die Kombination von eXBO und eXBO-Facilities ermöglicht bisher unerreichte Entwicklungsgeschwindigkeiten bei der Konstruktion von mittleren bis grossen eCommerce-Anwendungen. Auf die möglichen Einsparungspotentiale wird in einem späteren Kapitel detailiert eingegangen.

Im Gegensatz zu herkömmlichen Business-Objects, deren Attribute in den meisten Fällen nur ausreichen, um sie in normalen Web-Anwendungen darstellen zu können, verfügen die eXtended Business-Objects über spezialisierte Attribute, die eine professionelle Darstellung und Bearbeitung mit unterschiedlichsten, heutigen und zukünftigen Endgeräten ermöglichen.

Durch den Einsatz der eXBO-Facilities können die Objekte auf verschiedenen Endgeräten dargestellt werden. Die XAttributes liefern genügend Meta-Informationen, mit denen sich die Oberfläche optimal an die gegebenen Umstände anpassen kann. In dieser Diplomarbeit werden die eXBO-Facilities für das WebInterface entworfen und realisiert. Der gleiche Funktionsumfang ist aber auch leicht abgewandelt auf anderen Geräten denkbar. So könnte durch eine Erweiterung des Frameworks beispielsweise auch eine Darstellung auf einem PDA realisiert werden, die sich an den dort vorhandenen Einschränkungen orientiert und die notwendigen Optimierungen vollautomatisch umsetzt. Die XAttributes bringen die dafür notwendigen Eigenschaften bereits mit.

Die eXBO sind darüber hinaus ebenfalls in der Lage, für jedes einzelne Attribut mitzuteilen, wie es in Content-Management-Komponenten bearbeitet werden soll.

Beispiele: Für ein Attribut "longDescription" soll in der Benutzeroberfläche ein grosses Textfeld vorgesehen werden, ein Attribut "shortDescription" benötigt nur eine einzelne Textfeldzeile, für das Attribut "detailPicture" soll im User-Interface eine Upload-Funktionalität existieren, für ein Attribut "Anrede" soll anstatt dem bekannten Textfeld eine Choice (Auswahlbox) verwendet werden, die die Auswahl aus bestimmten vorgegebenen Werten (z.B. Herr, Frau oder Doktor) ermöglicht.

Durch diese Fähigkeit wird die Konstruktion von sogenannten eXBO-Facilities ermöglicht. Diese eXBO-Facilities, die die Verarbeitung von eXBO ermöglichen (dazu gehören das Anzeigen, Verändern und Suchen), müssen fortan nur ein einziges Mal implementiert werden und sind dann für alle Business Objekte nutzbar. Dies bedeutet im Kern, dass mit der Konstruktion eines eXBO auch alle Arbeiten an dem dazugehörigen Content-Management-System abgeschlossen sind. Der Entwickler eines eXBO wird also von der Konstruktion entsprechender Verarbeitungskomponenten für das zugehörige Content-Management-System bzw. für den Backoffice-Bereich befreit. Dies ermöglicht die bisher unerreichten Entwicklungsgeschwindigkeiten bei der Konstruktion von mittleren bis grossen eCommerce-Anwendungen.

1.2 Zielsetzung

Das Ziel der Arbeit ist die Umsetzung des vorgestellten Konzeptes für ein generisches Content-Management-System. Die eingeführten eXtended BusinessObjects werden dabei exemplarisch aus den vom Framework zur Verfügung gestellten Komponenten aufgebaut und die Funktion des Konzeptes belegt. Der theoretische Teil der Arbeit stellt die getroffenen Überlegungen umfangreich dar und geht auf die in diesem Zusammenhang aufgetretenen Probleme und deren Lösungen ein.

Das Konzept der eXtended BusinessObjects und des dahinter liegenden Frameworks zur Erstellung eines flexiblen und mit zahlreichen komfortablen Features ausgestatteten BackOffice für ein umfangreiches Content-Management stellt eine neue und gegenüber althergebrachten Arbeitsweisen sehr fortschrittliche und vielseitige Lösung dar. Die Möglichkeiten dieses Ansatzes werden in Kapitel 6 “Validierung” an einem Beispiel dargestellt.

1.3 Aufbau

Dieses erste Kapitel “Einleitung” beschreibt die grundlegende Motivation hinter dieser Arbeit und zeigt das gesteckte Ziel auf.

Im zweiten Kapitel “Bestandsaufnahme” wird auf den aktuellen Stand bei der Entwicklung von eCommerce-System eingegangen. Dazu gehört eine Analyse vorhandener Systeme mit ihren Stärken und Schwächen.

Das dritte Kapitel “Design & Entwurf” stellt den grundlegenden Aufbau des Frameworks dar. Zusätzlich wird auf mögliche Plattformen eingegangen, die aufgrund ihrer Architektur besonders gut für die Umsetzung des Konzeptes geeignet sind.

Im vierten Kapitel “Spezifikation” werden die sogenannten eXtended Attributes vorgestellt. Die eXtended Attributes sind der grundlegende Bestandteil der eXtended Business Objects. In den späteren Kapiteln wird im Detail gezeigt, wie eXtended Business Objetcs aus den eXtended Attributes aufgebaut werden können.

In Kapitel 5 - “Implementierung” - werden die verschiedenen Bearbeitungs- und Darstellungskomponenten für die eXtended BusinessObjects detailiert beschrieben und erläutert. Sie sind ein integraler Bestandteil des Frameworks und bilden einen weiteren Grundstein für die Flexibilität und Leistungsfähigkeit des Konzeptes.

Das 6. Kapitel mit dem Titel “Validierung” beschreibt anhand eines Beispiels den Einsatz der eXtended BusinessObjects und der dazugehörigen eXBO-Facilities bzw. Content-Management-Komponenten. Es wird die Umsetzbarkeit der zuvor beschriebenen Konzepte belegt und verifiziert. In diesem Zusammenhang wird auch auf eine kleine, aber trotzdem aussagekräftige, Beispielanwendung auf Basis des vorgestellten Frameworks eingegangen. Bei der Auswahl von Geschäftsobjekten – die natürlich nur unvollständig sein kann – wurden zwei typische, besonders häufig in elektronischen Umsetzungen von Geschäftsmodellen anzutreffende Objekte gewählt.

Das letzte und siebente Kapitel gibt eine abschließssende Zusammenfassung und einen Ausblick.

Im Anhang folgen das Abbildungs- sowie das Literaturverzeichnis und Source-Listings zum implementierten Framework.

2 Bestandsaufnahme

Ein neues Konzept kann nur dann vernünftig mit anderen Systemen verglichen werden, wenn man diese auch zuvor ermittelt und zumindest grundlegend einführt. Dieses Kapitel soll sich mit diesen anderen, bereits auf dem Markt befindlichen Systemen auseinandersetzten und deren Vor- und Nachteile offenlegen.

Das eXBO-Framework besitzt ein breites Einsatzspektrum, das es ermöglicht, Anwendungen für verschiedenste Bereiche des eCommerce entwickeln zu können. Die unterschiedlichen Bereiche bzw. Marktsegmente des eCommerce, die mit dem Framework erreicht werden können, werden im folgenden Abschnitt "Einsatzbereiche des eCommerce" dargestellt.

Die in diesem Abschnitt unter anderem erwähnten fertigen Standardsysteme "von der Stange" sind nur sehr bedingt mit einem Framework für die Entwicklung von unterschiedlichsten, individuellen, leistungsfähigen eCommerce-Systemen vergleichbar, denn sie sind zumeist abgeschlossene Anwendungen, die nur für ein einziges Geschäftsmodell einsetzbar sind und zudem höheren Anforderungen, z.B. an Integrationsfähigkeit und Erweiterungsmöglichkeiten, nicht gewachsen sind. Im Abschnitt "Standardanwendungen des eCommerce" wird das eXBO-Framework aus diesem Grund exemplarisch mit nur einer eCommerce-Standanwendung verglichen.

Im nachfolgenden Abschnitt "Frameworks für eCommerce-Systeme" werden die eigentlichen "Konkurrenten" des eXBO-Frameworks, nämlich andere Frameworks für die Entwicklung von eCommerce-Systemen, betrachtet.

Abschließend wird in diesem Kapitel noch auf die Standardisierungsbemühungen der OMG im Bereich der BusinessObjects eingegangen.

2.1 Einsatzbereiche des eCommerce

Wie bereits erwähnt kann das eXBO-Framework für Entwicklungen in den verschiedensten Bereichen des eCommerce eingesetzt werden. Dies sind insbesondere die Bereiche Business-to-Business (B2B), Business-to-Consumer (B2C), Consumer-to-Consumer (C2C), eGovernment und innerbetriebliche Rationalisierung durch elektronische Lösungen.

Innerhalb dieser Kategorien können wiederum verschiedenste Geschäftsmodelle bzw. Ausprägungen identifiziert werden, die ständig durch neue Geschäftsmodelle bzw. Geschäftsideen erweitert werden. Nachfolgend wird kurz auf den aktuellen Entwicklungsstand der verschiedenen Bereiche eingegangen.

Consumer to Consumer (C2C)

Beispielhafte Vertreter dieser Kategorie von Electronic Commerce sind Gebrauchtwarenmarktplätze wie z.B. "Zweite Hand" und "AutoScout 24" oder auch Versteigerungen wie z.B. "eBay", "Ricardo" oder "QXL". Mischformen von C2C und B2C stellen z.B. Ratgeberplätze dar, bei denen einerseits Produkte von Händlern angeboten werden, aber gleichzeitig auch die Meinungsbildung über die öffentliche Diskussion von Verbrauchern zur Kaufempfehlung führt. Weiterführende Literatur zu verschiedenen Ausprägungen des C2C findet man unter anderem in [1] und [2].

Business to Consumer (B2C)

Der Business to Consumer-Bereich wird von eCommerce-Shop-System dominiert. Zu den bekanntesten (und erfolgreichsten) Vertretern gehören beispielsweise *www.dell.com* und *www.apple.com,* die jeweils mehrere Millionen Dollar Umsatz pro Monat erzielen.

Neben den normalen eCommerce-Shop-Systemen gibt es weitere Ausprägungen des Business to Consumer-Bereichs. Zu ihnen gehören beispielsweise Marktplätze bei denen mehrere Shops in einem Portal zusammengefügt werden. Durch erweiterte Dienste, z.B. einer übergreifenden Suche können dadurch Vorteile bzw. Mehrwerte für den Kunden entstehen, die sich dann natürlich auch beim Shopanbieter niederschlagen. Eine weitere Anwendung im Bereich des B2C stellt das sogenannte "Power-Shopping" (Gruppenkauf) dar. Bei dieser Variante

sammeln sich Kaufinteressierte für ein bestimmtes Produkt und bestellen später – ähnlich wie ein Grosshändler – grössere Mengen mit entsprechendem Mengenrabatt. Weitere Literatur zu den verschiedenen Geschäftsmodellen im Business to Consumer Bereich findet man in [1-5].

Business to Business (B2B)

Innerhalb des Bereichs Business to Business haben sich bis jetzt die Rollen Buyer, Seller und Marketmaker herauskristallisiert. Zu den wichtigsten Anwendungen in diesem Bereich gehören Buyer-Managed-, Seller-Managed- und Marketmaker-Managed- Marktplätze. Neben diesen drei etablierten Anwendungsfällen, die nachfolgend noch einmal näher beschrieben werden, existieren zahlreiche andere Geschäftsmodelle und verschiedenste Entwicklungsperspektiven. In diesem Zusammenhang sei besonders auf [2-4] verwiesen.

a) Buyer-Managed-Marktplätze

Beispiele für Buyer-Managed-Marktplätze sind die Handelsplattformen der US-Auto-Riesen Ford und General Motors. Über diese Plattformen sollen die Geschäftsbeziehungen mit den jeweils bis zu 30.000 verschiedenen Geschäftspartnern abgewickelt werden. Ziel ist es, das gesamte Zulieferergeschäft besser, schneller und effizienter zu gestalten. Bei Auto-Riesen, wie z.B. Ford, deren Zulieferergeschäft auf ca. 300 Milliarden Dollar geschätzt wird, bedeutet die erhoffte Senkung der durchschnittlichen Transaktionskosten von 100 $ auf 10 $ eine Einsparung von bis zu 16 Milliarden Dollar pro Jahr.

Andere Beweggründe für grosse Einkäufer, Buyer-Managed-Marktplätze zu etablieren, bestehen in der Möglichkeit, erhöhte Transparenz über alle Vorgänge bei den Zulieferern zu bekommen. Dies kann z.B. dafür genutzt werden, die Lieferzeiten weiter zu verkürzen und erhebliche Kostensenkungspotentiale im Bereich der Lagerhaltung auszunutzen.

Industriezweige, wie z.B. die Automobilindustrie, die durch wenige aber sehr grosse Hersteller und viele, aber kleinere Zulieferer geprägt werden, sind besonders für Buyer-Managed-Markplätze prädestiniert. Weitere Literatur zum Thema Buyer-Managed-Marktplätze findet man in [1-2].

b) Supplier-Managed-Marktplätze

Industrien, die für Supplier-Managed-Marktplätze prädestiniert sind, weisen wenige grosse Hersteller und viele kleinere Abnehmer auf. Beispiele für Supplier-Managed-Marktplätze finden sich z.B. in der Stahlindustrie oder in der Medizintechnik. Wenngleich zwischen B2B- und B2C-Anwendungen erhebliche Unterschiede bestehen (besonders in Umfang und Aufwand der getätigten Transaktionen, vgl. [1] und [2]), weisen einfache Supplier-Managed-Marktplätze ähnliche Funktionalitäten wie B2C-Shop-Systeme auf. Weitere Literatur zum Thema findet man in [1-5].

c) Marketmaker-Marktplätze

Marketmaker-Marktplätze sind Marktplätze, die weder vom Zulieferer (Supplier), noch von Käufer (Buyer) beherrscht werden. Sie verstehen sich vielmehr als "... intersection of demand and supply chain." [2] - also der Schnittstelle zwischen verschiedensten Anbietern und Herstellern. Zur Verfügung gestellt werden sie von sogenannten "Marktplatzbetreibern" (Marketmaker), die auf verschiedenste Weisen (z.B. durch prozentuale Beteiligung an Transaktionen, monatlichen Pauschalbeträgen, Werbung, Vermittlungsprovision, etc.) profitieren können. Weitere Literatur zum Thema findet man in [1-3].

eGovernment

Der gesamte Bereich des eGovernment ist noch weniger erschlossen als der Bereich des eCommerce. Es gibt so gut wie keine Standardanwendungen, oder gar etablierte Rollen. Einflussfaktoren sind regionale und nationale Unterschiede in den Verwaltungsstrukturen, und erhebliche Sicherheitsbedenken. Das sehr frühe – erst beginnende – Stadium des eGovernment wird in die Kategorien Government to Business, Government to Government und Government to Consumer unterteilt, auf die nachfolgend kurz eingegangen wird.

a) Government to Business (G2B)

Diese Kategorie beinhaltet alle Formen von Transaktionen zwischen Unternehmen und öffentlichen Verwaltungen. Diese können sowohl von geschäftlicher Natur sein (wie z.B. bei öffentlichen Ausschreibungen für Bauaufträge) oder zur Erfüllung des gesetzlichen Ordnungsrahmens dienen (z.B. Gewerbeanmeldungen

oder Steuerbescheide). Weiterführende Literatur zum Thema Government to Business und insbesondere den möglichen Überschneidungen von Business to Business und Government to Business findet man in [1-5].

b) Government to Government (G2G)

Diese Form in der Beziehung beinhaltet den elektronischen Austausch von Informationen zwischen verschiedenen öffentlichen Verwaltungen und die Vereinheitlichung und Zusammenführung von Datenbeständen. Der gegenwärtige Stand ist durch verschiedenste, teilweise komplexe Verwaltungsstrukturen und Verwaltungsvorschriften und durch grösstenteils inkompatible, historisch gewachsene Datenverarbeitungssysteme geprägt. Sollen die Bereiche Government to Business und Government to Consumer durch elektronische Verfahren schneller und effizienter gestaltet werden, so muss auch die innerbehördliche Kommunikation den gestiegenen Leistungsanforderungen angepasst werden. Bestrebungen, wie die der Europäisierung, erfordern darüber hinaus eine effiziente und geregelte Kommunikation über Landes- und Regierungsgrenzen hinweg. Weiterführende Literatur findet man in [1-5].

c) Government to Citizen (G2C)

Diese Kategorie des eGovernment beinhaltet alle Formen der Interaktionen zwischen Bürgern und öffentlichen Verwaltungen. Der heutige, zögerliche Beginn der Kategorie Government to Citizen ist durch statische Informations-bereitstellungen im World Wide Web gekennzeichnet. Viele aktuelle Projekte die sich mit der Vereinheitlichung und Zusammenführung von Datenbeständen beschäftigen, ermöglichen in einem ersten Schritt die Schaffung von "Bürgerbüros", die es ermöglichen, verschiedene öffentliche Verwaltungs-vorgänge von einem physikalischen Ort aus durchzuführen. Zukünftige Anwendungsfälle sind Bürgerportale im World Wide Web, in denen Bürger selbständig, z.B. für die Unterstützung von Lebenslagen (Heirat, Umzug, etc.) Verwaltungsvorgänge initiieren und ausführen können. Konkrete Projekte sind z.B. „eGOV", „VeZuDa" oder „ProBüd". Weitere Literatur zu diesem Thema findet sich in [46-47].

Wie auch bei der Kategorie eCommerce sind nicht alle Modelle sauber in die drei Bereiche G2B, G2G und G2C einzuordnen. Aktuelle Ansätze, wie z.B. die "Virtuellen Kommunen", in denen Städte oder ganze Regionen sich selbst, ihre Unternehmen, ihre Geschäfte, ihre Bürger und die öffentlichen Institutionen präsentieren, können Elemente aus allen eGovernment- und eCommerce-Kategorien beinhalten.

Weitere Ausprägungen des eCommerce

Bevor nachfolgend auf die grundlegenden Anforderungen und die verschiedenen Transaktionsphasen von eCommerce eingegangen wird, muss an dieser Stelle noch einmal betont werden, dass die obige Auflistung nur einige der wichtigsten und allgemein anerkannten Kategorien und Ausprägungen von eCommerce aufzeigt und keineswegs den Anspruch auf Vollständigkeit hat. Der Bereich des eCommerce weist viele weitere interessante Bereiche auf, die bisher noch nicht einmal erwähnt werden konnten. Dazu gehören z.B. eFinance (die elektronische Abwicklung von Zahlungsverkehr, z.B. mit Banken und Kreditinstituten) und die innerbetriebliche Optimierung und Rationalisierung von Geschäftsmodellen durch elektronische Lösungen, um nur zwei weitere wesentliche Bereiche zu erwähnen. Die weitere ausführliche Betrachtung aller Ausprägungen des eCommerce würden den Rahmen dieser Arbeit sprengen. Weitere Literatur zu den Themen eCommerce, eGovernment, eFinance, Rationalisierung und Optimierung durch eCommerce, eBusiness B2C, B2B, B2C, G2G, G2B, G2C usw. findet man in [1-5].

2.2 Frameworks für eCommerce-Systeme

2.2.1 Apple WebObjects

Die WebObjects Entwicklungs- und Deploymentumgebung liefert eine Vielzahl von Werkzeugen und Frameworks für die Entwicklung von Internet- und eCommerce-Anwendungen.

Wesentliche Bestandteile der WebObjects Umgebung sind das Enterprise Objects Framework (EOF), das WebObjects Framework (WOF) und das Foundation Framework, die im folgenden kurz vorgestellt werden.

a) Enterprise Objects Framework (EOF)

Die Objekte und Komponenten dieses Frameworks sind darauf spezialisiert, aus beliebigen Datenquellen, meist relationalen Datenbanken, automatisiert Objekte zu erzeugen bzw. die erzeugten Objekte persistent zu halten. Die Objektebene und die Datenbankebene werden dabei durch Adapter-Objekte sauber voneinander getrennt. Über die Verwendung von J2EE-konformen JDBC-Adaptoren zu Datenbanken hinaus wird auch die Entwicklung von zusätzlichen Adaptor-Komponenten unterstützt mit denen der Zugriff auf unterschiedlichste Datenhaltungssysteme (z.B. Altsysteme, wie hierarchische Datenbanken, historisch gewachsene Systeme wie z.B. Flugabfragesysteme mit einer eigenen Abfragesprache, Warenwirtschaftssysteme, Enterprise Resource Planungssysteme usw.) ermöglicht werden kann. Weitere Informationen über das Enterprise-Objects-Framework sind z.B. in [27-29] sowie [30-32] zu finden.

b) WebObjects-Framework (WOF)

Das WebObjects-Framework (von dem der aktuelle Name für die gesamte Entwicklungsumgebung und auch für den Applicationserver abgeleitet wurde) ist spezialisiert für den Aufbau von Internetanwendungen. Eine der wesentlichsten Komponenten in diesem Framework ist die flexible und höchst leistungsfähige, serverseitige Session- bzw. Zustandsverwaltung. Ausserdem existieren zahlreiche Objekte und Komponenten für die Konstruktion von dynamischen Internetseiten in Form von JSP-Tag-Libraries. Die Wiederverwendung von Objekten und Komponenten beim Aufbau von dynamischen Internetseiten wird massiv unterstützt. Mehr Informationen zum WebObjects Framework sind in [28-29] zu finden.

c) Foundation Framework

Der Foundation Framework beinhaltet Basisklassen für die Programmierung, wie z.B. Arrays, Timer usw. Mehr Informationen zum Foundation Framework sind in [25-29] zu finden.

Die vorgestellten Frameworks enthalten zahlreiche sehr nützliche Klassen für die Entwicklung von Internet-Anwendungen. Spezialisierte Business-Objekte und -Komponenten enthalten diese Frameworks jedoch nicht.

Pro:

- sehr gute Skalierbarkeit
- geringer Preis von US $ 900 (vormals US $ 50.000)
- Unterstützung von verschiedenen Unix-Derivaten und Windows
- umfangreiche Frameworks
- flexible Anbindung von Datenquellen aller Art
- umfangreiche Referenzliste erfolgreicher Grosskunden-Projekte

Contra:

- keine speziellen Business- oder Geschäftsobjekte
- keine selbst konfigurierenden Komponenten (die mit den eXBO-Facilities vergleichbar wären)
- keine vorgefertigten Komponenten für Business Aufgaben

Fazit:
WebObjects ist eine hervorragende Umgebung für die Entwicklung von Internetanwendungen aller Art. Es fehlt jedoch an spezialisierten Objekten und Komponenten für eCommerce Aufgaben. Solche Objekte und Komponenten müssen aufbauend auf die vorhandenen Frameworks von Hand selber erstellt werden. Weitere Alleinstellungsmerkmale, wie sie etwa das eXBO-Framework mit den selbst konfigurierenden eXBO-Facilities bietet, sind ebenfalls nicht vorhanden. WebObjects ist allerdings nicht nur als Konkurrenzsystem anzusehen: Die gleichzeitige Nutzung von WebObjects Frameworks und des eXBO Frameworks schliessen sich nicht aus. Das eXBO-Framework kann ebenfalls als Erweiterung von WebObjects genutzt werden, wodurch die vorhandenen Defizite von WebObjects im eCommerce-Bereich ausgeglichen werden können.

2.2.2 BEA WebLogic Portal Server

Aufbauend auf dem Bea Weblogic Application Server bietet Bea Weblogic Portal Server Komponenten, die die Entwicklung von eCommerce-Systemen, insbesondere von B2B- und B2C-Portalen, unterstützen. Die Komponenten des Bea Weblogic Logic Portal-Server unterstützen dabei insbesondere die beim Aufbau von eCommerce-Portalen typischen Problembereiche wie Personalisierung und Workflow.

Die Bea Weblogic-Plattform hat weitere Stärken bei der Integration in vorhandene IT-Umgebungen und der sehr flexiblen Anbindung von unterschiedlichsten Datenquellen und Anwendungen.

Bea Weblogic Portal Server unterstützt dabei vollständig die objektorientierte Programmierung und Standardtechnologien wie Java und J2EE und ist für verschiedenste Betriebssysteme erhältlich. Der Bea Weblogic Portal Server gehört zu den am weitesten verbreiteten Application Servern insbesondere im Bereich der "Fortune 1000" - Unternehmen.

Pro:

- starke Verbreitung
- Standard konform (Java und J2EE)
- gut in verschiedene IT-Landschaften integrierbar
- Portal-spezifische Businesskomponenten wie z.B. Personalisierung

Contra

- hoher Preis (> US $ 50.000)
- auf Portal-Funktionalitäten spezialisierte, nicht allgemein einsetzbare Business-Komponenten
- komplexe Installation und lange Einarbeitungszeiten
- keine mit den eXBO-Facilities vergleichbaren innovativen Ansätze, die ähnliche Einsparungspotentiale versprechen

Fazit:
Nicht nur aufgrund der Standard konformen Technologie, der guten Integrierbarkeit in unterschiedlichste IT-Umgebungen und der Verfügbarkeit für verschiedene Plattformen ist das Bea Weblogic Portal-System eines der leistungsfähigsten Konkurrenzprodukte. Aufgrund des hohen Preises ist der Zielmarkt des Bea Weblogic Portal-Servers eindeutig bei grösseren Unternehmen zu suchen. Für kleine und mittlere Unternehmen bietet der Bea Weblogic Portal-Server keine wirtschaftlichen Alternativen, insbesondere bei deren Einstieg in den eCommerce-Bereich.

Die vorgefertigten Komponenten des Bea Weblogic Portal-Servers sind zudem für den Aufbau von Portal-Systemen spezialisiert und nur bedingt an vollkommen andere Bedürfnisse bzw. individuelle Geschäftsprozesse von Unternehmen anzupassen.

2.2.3 IBM WebSphere Commerce Suite

Aufbauend auf den, noch auf C++ basierenden Vorgängern IBM Websphere Commerce Suite V 4.1 und dem auf C++ und Lotus Domino basierenden IBM Netcommerce, präsentiert IBM mit der IBM Websphere Commerce Suite Pro eine mittlerweile vollständig auf Java basierte Suite für den Aufbau von eCommerce Anwendungen.

Die besonderen Stärken liegen in den konsequenten Unterstützung von Standard Technologien wie Java und J2EE, von Standard-Protokollen, wie CGI, XML, HTML und SSL, sowie der Unterstützung von unterschiedlichen Webservern, wie z.B. dem Apache-Domino oder Netscape-Server (neben dem eigenen IBM HTTP-Server).

Die Unterstützung verschiedenster Betriebssystem-Plattformen (NT, AIX, Solaris, AS/400, S/300) und die Verfügbarkeit in mittlerweile zehn verschiedenen Sprachen sind ebenfalls als Stärken zu nennen.

Als nachteilig sind die (aus Marktgründen) selbst auferlegte Beschränkung auf nur zwei relationale Datenbanksysteme als mögliche Datenquellen (Oracle und IBM´s DB2) und die noch immer nicht optimale Performanz einiger Frameworkteile, die schon bei mittleren Zugriffszahlen durch entsprechende Skalierungs- und Hardwareausgaben ausgeglichen werden müssen, anzusehen.

Das von IBM im Rahmen der IBM Websphere Business Components integrierte San Francisco Framework, das wie das eXBO-Framework auf der Idee von Geschäftsobjekten bzw. Business-Objekten basiert, stellt für die Entwicklung von eCommerce Anwendungen spezialisierte Objekte und Komponenten zur Verfügung. Deren Fähigkeiten sind allerdings nicht mit den hier vorgestellten Konzepten der selbstkonfigurierenden eXBO-Facilities und der extended BusinessObjects vergleichbar, die darüber hinaus gehende Einsparungspotentiale besitzen.

Pro:

- Unterstützung von Standardtechnologien wie Java und J2EE
- Standardprotokolle wie CGI, SSL, HTTP ermöglichen den Austausch von Komponenten, wie z.B. von Webserver oder Applicationserver (keine Abhängigkeit von IBM Produkten)
- Unterstützung verschiedenster Betriebssystemplattformen

- spezialisierte Businessobjekte und Komponenten
- umfangreiche Suite mit vielen Komponenten und Werkzeugen

Contra:

- Beschränkung auf zwei relationale Datenbanken
- Framework-Teile mit mäßiger Performanz
- sehr komplexe und zeitaufwendige Installation
- lange Einarbeitungszeiten aufgrund der grossen Komplexität
- hohe Kosten (Einstiegspreis bei > US $ 50.000 per CPU; voll skalierbare Mehrprozessor-Varianten bis zu US $ 500.000)

Fazit:
Die IBM Websphere Commerce Suite ist eine auf die Entwicklung von eCommerce-Anwendungen spezialisierte Umgebung mit zahlreichen Features und speziellen Business-Objekten und Business-Komponenten. Allerdings ist dieses Produkt aufgrund der sehr hohen Kosten vorrangig den grossen und grössten Unternehmen vorbehalten. Die IBM Websphere Commerce Suite bietet bei der Herabsetzung der wirtschaftlichen Einstiegsbarrieren in den eCommerce insbesondere für kleinere und mittlere Unternehmen keine wirkliche Alternative.

2.2.4 Microsoft Commerce Server

Aufbauend auf den Vorgängern Microsoft Site Server und Microsoft Merchand Server bietet Microsoft mit dem Microsoft Commerce Server eine auf der .net-Technologie aufbauende eCommerce Suite für den Aufbau von eCommerce Systemen an. Diese Suite beinhaltet verschiedene Teil-Anwendungen bzw. grössere vorgefertigte Komponenten für verschiedene eCommerce typische Problembereiche wie z.B.: Product Catalogue System, Profile System, Business Processing Pipeline, Business Analytic System, Data Warehousing, usw.. Die einzelnen, früher zum Teil eigenständigen Systeme, sind durch zusätzliche .net-Programmierung miteinander gekoppelt.

Bei der Erstellung von individuellen, auf das jeweilige Geschäftsmodell eines Unternehmens zugeschnittenen eCommerce-Anwendungen sind die enthaltenen Teilanwendungen bzw. vorgefertigten Komponenten des MS-Commerce-Server nur dann produktivitätssteigernd einsetzbar, wenn die Geschäftsmodelle bzw. -prozesse zu denen des Unternehmens passen. Geschäftsmodelle bzw. -Prozesse

von Unternehmen, die sich nicht mit dem vorgegebenen Schema abbilden lassen, können nur schlecht von den vorgefertigten Komponenten profitieren, da für eine Wiederverwendung die Granularität vieler Komponenten zu gross ist.

Die Microsoft Commerce Suite ist zudem nur denjenigen Unternehmen vorbehalten, die sich ganz in die Abhängigkeit des Microsoft-Unternehmens begeben können oder wollen: Als Programmiersprache wird das Microsoft eigene C# präferiert, erstellte Einzelkomponenten sind nicht J2EE Standard konform, also nicht in anderen Umgebungen wieder verwendbar, als Application Server ist man an den Microsoft eigenen Application Server gebunden, auch im Bereich des HTTP-Servers ist nur der Microsoft eigene IIS-Webserver einsetzbar. Grosse Teile der Funktionalität sind direkt von dem Microsoft eigenen SQL Server-Datenbanksystem abhängig, so dass auch bei der Wahl der Datenquelle keine wirklich freie Entscheidung bzw. Flexibilität vorhanden ist.

Der Einstiegspreis ist mit ca. US $ 12.000 pro Prozessor deutlich unter dem des Bea Weblogic Portal-Servers und der IBM Websphere Commerce Suite. Auch bei anspruchsvolleren Deployment-Installationen liegen die Preise für die notwendigen Microsoft Commerce Server Lizenzen noch deutlich unter denen der Konkurrenzprodukte (eine noch nicht voll, zumindest aber auf acht Prozessoren skalierbare Microsoft-Lösung liegt bei „nur" US $ 200.000). Nachteilig ist natürlich auch die vollkommene Abhängigkeit von der Microsoft Produkt- und Lizenzpolitik. Sollten sich die Preise, aus welchem Grund auch immer, verändern, ist man schon technisch nicht in der Lage, beispielsweise zu einem anderen Application-Server (mit vielleicht wesentlich günstigerem Preis-Leistungsverhältnis) zu wechseln.

Pro:

- Hoher Vorfertigungsgrad von Komponenten bis hin zu fertigen Teilanwendungen für einige eCommerce-Bereiche
- Bekanntheit und Grösse von Microsoft

Contra:

- Die vorgefertigten Teilanwendungen bzw. Komponenten sind sehr weit an fertige Produkte angelehnt, wodurch die Wiederverwendung für individuelle Business-Lösungen eingeschränkt wird.
- Keine freie Wahl des Datenbanksystems
- Keine freie Wahl oder Austauschmöglichkeit des Webserver

- Keine freie Wahl oder Austauschmöglichkeit des Applicationservers
- Keine Unterstützung des J2EE Standards
- Vollkommene Abhängigkeit von einem Unternehmen
- Abhängigkeit von Microsofts Produkt- und Preispolitik

Fazit:
Für die Umsetzung von individuellen eCommerce Systemen, von elektronischen Lösungen also, die meist eine Abbildung der real existierenden Geschäftsprozesse darstellen, ist die Suite immer dann produktiv einsetzbar, wenn die Teilanwendungen der Microsoft Commerce Server Suite die real existierenden Geschäftsprozesse abbilden können. Sind die Geschäftsmodelle und -prozesse eines Unternehmens nicht deckungsgleich bzw. "nicht kompatibel", so können die enthaltenen Teilanwendungen und Komponenten nicht bzw. nur mit unverhältnismäßig grossem Aufwand wiederverwendet werden.

Die Lizenzierungspolitik mit Preisen weit über 200.000 US $ für voll skalierbare Lösungen liegt zwar weit unter denen von Bea und IBM, stellt jedoch für kleinere und mittlere Unternehmen noch immer eine erheblich Einstiegshürde dar.
Die Einzelkomponenten des Microsoft Commerce Server Systems verschliessen sich zudem Standards wie beispielsweise J2EE, die einen Austausch von Komponenten ermöglichen würden, wodurch Unternehmen, die entsprechende Investitionen tätigen, in eine vollkommene Abhängigkeit von Microsofts Produkt- und Lizenzierungspolitik geraten würden.

2.3 Standardanwendungen des eCommerce

Stellvertretend für existierende Standardanwendungen im Bereich des eCommerce wird hier der wohl bekannteste Vertreter aus dem B2C-Shop-Bereich, "Intershop", betrachtet.

Fertige Standardsysteme "von der Stange", wie z.B. Intershop sind nur sehr bedingt mit den bisher betrachteten eCommerce-Suiten und Frameworks für die Entwicklung von leistungsfähigsten, individuellen eCommerce-Systemen vergleichbar. Intershop ist (wie die meisten der Fertig-Anwendungen) eine in sich abgeschlossene Anwendung die nur für ein einziges Geschäftsmodell (nämlich als Shop-System) einsetzbar und auch innerhalb dieses Anwendungsfalls kaum an

eigene Bedürfnisse anpassbar ist. Gleiches gilt für die Integrationsfähigkeit in vorhandene IT-Umgebungen und insbesondere für die Ausbaufähigkeit des Systems.

Pro:

- Geringer Preis
- Sofort Einsetzbar

Contra:

- Auf ein einziges Geschäftsmodell beschränkt
- Nur schlecht an eigene Bedürfnisse anpassbar
- Schlechte Integrationsfähigkeit in vorhandene IT-Umgebungen
- Keine wieder verwendbaren Komponenten für den eCommerce-Bereich
- Abhängigkeit von mitgelieferter Datenbank
- Keine objektorientierte Programmierung
- Keine Business Objekte
- Keine Unterstützung von Standards, wie Java oder J2EE
- Nicht ausbau- und entwicklungsfähig

Fazit:
Intershop beschränkt sich alleine auf das Geschäftsmodell B2C-Shop. Es existieren weder Business Objekte noch wiederverwendbare Komponenten. Das System ist primär für die Verwendung "out of the Box" gedacht. Anpassungen oder Erweiterungen z.B. an firmeneigene Geschäftsmodelle und -prozesse sind nicht vorgesehen. Für die Entwicklung von individuellen, leistungsfähigen, auf die jeweiligen Geschäftprozesse eines Unternehmens angepasste eCommerce-Systeme stellt Intershop keine Alternative dar.

2.4 OMG Business Objects

Die Object Management Group (OMG) hofft durch ihre Bemühungen im Bereich der Business-Objects, die Entwicklung von Geschäftsanwendungen wesentlich zu verbessern. Diese Verbesserungen beruhen hauptsächlich auf zwei Punkten.

Erstens verspricht man sich durch die Definition der Business-Objects eine Annäherung zwischen der betriebswirtschaftlich-anwendungsbezogenen Sichtweise und der technisch-implementierungsbezogenen Sicht der Entwickler. Die schnelle und effiziente Entwicklung von spezialisierten Geschäftsanwendungen erfordert eine enge Zusammenarbeit von Software-Ingenieuren und Experten aus den jeweiligen Geschäftsbereichen. Für die schnelle und korrekte Umsetzung von Anforderungen aus der Geschäftswelt in einsatzfähige Softwaresysteme benötigt man einen Weg, um geschäftliche Konzepte, Prozesse, Elemente und Zusammenhänge so zu beschreiben, dass sie von beiden Welten gleich gut und eindeutig verstanden werden. Dies ist eine der Hauptintentionen der Business-Objects.

Der zweite Punkt ist die Bereitstellung von wiederverwendbaren, getesteten, erprobten und standardisierten Komponenten, mit denen es möglich ist, Geschäftsanwendungen mit wesentlich geringeren Kosten und mit wesentlich kürzeren Entwicklungszyklen zu erstellen. Zukünftige Geschäftsanwendungen sollen ausserdem die Fähigkeit besitzen, auch komplexere Aufgaben, wie zum Beispiel die firmenübergreifende Kooperation durch Business-Objects, zu bewältigen.

Was aber sind Business-Objects?

Unter Business-Objects versteht man Objekte, die reale Dinge aus dem Geschäftsleben repräsentieren, zum Beispiel einen Kunden, eine Rechnung oder einen Auftrag.

Die OMG definiert ein Business-Object sinngemäß als “die Repräsentation eines Objektes, das in einem Unternehmensbereich aktiv ist. Es umfasst seinen Namen und seine Definition innerhalb des Unternehmens, Attribute, Verhalten, Beziehungen, Regeln, Vorgehensweisen und Randbedingungen. Ein Business-Object kann zum Beispiel eine Person, einen Ort, ein Ereignis, einen Geschäftsprozess oder ein Konzept repräsentieren”.

Diese Definition lässt erkennen, dass der Begriff der “Business-Objects” sehr allgemein gehalten wird. Um eine bessere Kategorisierung von den verschiedenen Business-Object-Typen zu erreichen wird von Mitgliedern der BODTF (Business Object Domain Task Force) der OMG eine Klassifizierung der Business-Objects vorgeschlagen. Diese Klassifizierung soll das Verständnis unter

den Anwendungsentwicklern verbessern, die in Zukunft per Katalogauswahl auf vorhandene Komponenten zugreifen sollen. Die OMG Business-Objects werden dabei nach drei Typen unterschieden: Entity-Business-Object (Entität bzw. Gegenständlichkeit), Process-Business-Object (Ablauf) und Event-Business-Object (Ereignis).

Beispielhafte Vertreter der ersten Kategorie von Business-Objects (Entity-Business-Objects) sind reale oder abstrakte Phänomene, die in Fach-spezifikationen meist als Substantive auftauchen: Kunde, Bestellung, Adresse oder Vertrag. Eine weitere Hilfe für das Aufspüren von Entity-Business-Objects ist die Tatsache, dass ihre Attribute in herkömmlichen Anwendungen oft und gerne in Datenbanken gespeichert werden.

Process-Business-Objects sind im Gegensatz dazu stets zeitbehaftete Geschehnisse bzw. Geschäftsabläufe, wie zum Beispiel Verkaufen, Beschaffen, Unterstützen, Abrechnen oder Liefern.

Event-Business-Objects sind die Abbildung von Ereignissen. Bei diesen kann es sich um Ereignisse handeln, die zeitabhängig sind (wie zum Beispiel "Steuererklärung anfertigen, weil Kalendermonat vorbei ist"), um Ereignisse die vom System selbst herbeigeführt werden (wie zum Beispiel "Mindestlagermenge unterschritten" oder "Kontobuchung ausgeführt") oder auch um Ereignisse die vollständig unabhängig vom System sind (wie zum Beispiel "Tank leer" oder "Apple-Aktie > 100 $").

Die Vorgehensweise der OMG ist dabei so geregelt, dass man versucht, zunächst domain-spezifische Business-Objects zu finden. Durch die Etablierung derartiger Objekte, mit denen grundlegende und typische Aufgabenstellungen in den jeweiligen Branchen abgedeckt werden können, soll vor allem die Kooperation verschiedener Firmen einer Branche gefördert werden. Die domain-spezifischen Business-Objects können ihrerseits noch in Unternehmens- bzw. Enterprise-spezifische Business-Objects verfeinert werden. Für einige Business-Objects ist es notwendig, regional bzw. geopolitisch bedingte Variationen einzuführen. Beispiele für diese Variationen sind Business-Objects, die Steuersätze oder Währungen repräsentieren. Nachdem domain-spezifische Business-Objects identifiziert werden können, wird versucht, durch den Gedankenaustausch mit den anderen OMG Domain Task Forces eine grössere Übereinstimmung zu finden, um so ein allgemeingültiges und somit am besten wiederverwendbares "Common Business Object" (CBO) zu erhalten.

Da der OMG Interface Definition Language (IDL) die Ausdrucksmittel fehlen, um eine ausreichende semantische Beschreibung der Schnittstellen und Eigenschaften von komplexen Business-Objects zu beschreiben, wird innerhalb der Business Object Domain Task Force (BODTF) der OMG versucht, die Entwicklung und Standardisierung der sogenannten Component Definition Language (CDL) voranzutreiben. Die Component Definition Language ermöglicht eine semantische Spezifikation von Objekten, deren Beziehungen zu anderen Komponenten und eine formale Beschreibung des Verhaltens, inklusive etwaiger Vor- und Nachbedingungen. Die CDL ist eine Erweiterung der IDL.

In den Diskussionen der BODTF kann man zwei verschiedene Stadien von Business-Objects unterscheiden. In einem ersten Stadium wird versucht, Business-Objects aus einem Business-Modell abzuleiten. Die dabei identifizierten Business-Objects werden textuell (zum Beispiel mit der Component Definition Language) oder graphisch (zum Beispiel mit der Unified Modelling Language (UML)) beschrieben. In diesem Stadium existieren die Business-Objects unabhängig von Informationsverarbeitungssystemen, Anwendungen, Software-Design oder Programmcode. Ausserdem wird nicht davon ausgegangen, dass jedes Business-Object in diesem Stadium in einer späteren Anwendung als solches implementiert werden muss.
Business-Objects im zweiten Stadium sind "reale" Software-Objekte mit allen "normalen" Objekteigenschaften. Zusätzlich können die Business-Objects Eigenschaften haben, die über die Beschreibung des OMG-Objekt-Modells hinausgehen. Diese Eigenschaften können zum Beispiel durch die CDL oder die UML beschrieben werden:

Behavior: Verhaltensbeschreibungen, wie zum Beispiel: "immer wenn ein Jahreswechsel vollzogen wird, wird veranlasst, dass..."

Business rules: exakte Beschreibung von Geschäftsregeln die vom Business-Object eingehalten werden müssen

Business identity: eindeutige Identifikation jeder Instanz des Business-Objects, um es der entsprechenden Entität in der realen Welt zuordnen zu können

Integrity of instances and inter-instance relationships: um die Vollständigkeit einer Instanz zu garantieren; im Falle eines Vertrages darf zum Beispiel nicht die Hälfte fehlen

Persistence: Business-Objects sollen solange existieren, wie die „realen" Phänomene, die sie abbilden

Security: ein Business-Object "Rechnung" darf zum Beispiel nicht unauthorisiert verändert werden

Interoperability: Business-Objects verschiedener Hersteller sollen sich untereinander verständigen können

Transactability: die Vollständigkeit von Transaktionen soll garantiert werden können; im Falle eines Fehlers oder Abbruchs soll ein Rollback durchführbar sein

Die ursprüngliche Vision der OMG ging davon aus, dass Entitätstypen wie Kunde, Artikel, Verkauf oder Rechnung relativ schnell als Common Business Objects (CBOs) definierbar sind. Inzwischen ist man erheblich zurückhaltender, zeigte sich doch in der Diskussion, dass selbst bei diesen, auf den ersten Blick einfach erscheinenden Business-Objects grosse Verständigungsschwierigkeiten bestehen. Mit erheblich reduziertem Anspruch wird nunmehr in einem Bottom-Up-Ansatz versucht, nach und nach wenigstens eine Normierung elementarer Objekttypen (wie zum Beispiel Decimal oder Time) zu erreichen. Aktuelle Arbeiten der BODTF betreffen die Umrechnung, Verwaltung und Darstellung von Währungs- und Datumseinheiten, Objekte zur Repräsentation von Geschäftsadressen sowie Kalenderfunktionen.

Das Konzept der Business-Objects ist nach Einschätzung der Experten ein Schritt in die richtige Richtung. Sollte es gelingen, auch nur einen Teil der Konzepte und grundlegenden Ideen umzusetzen, könnte dies einen evolutionären Schritt für die gesamte Software-Entwicklung, vor allem aber für den Bereich der Entwicklung von Geschäftsanwendungen bedeuten.

Zusammenfassend lässt sich also sagen, dass die Konzepte der OMG Business-Objects sehr vielversprechend sind. Der gegenwärtige Stand der Entwicklung im Bereich der OMG Business-Objects erlaubt es jedoch nicht, auf eine ausreichend grosse Anzahl von ausgereiften, standardisierten Business-Objects zurückzugreifen, mit denen man schon heute konkrete eCommerce-Anwendungen erstellen könnte. Es ist angebracht, die Entwicklung der OMG Business-Objects weiterhin zu verfolgen um schon jetzt von den aufgezeigten Lösungsvorschlägen und Ideen zu profitieren und sie in eigene Ansätze einfliessen zu lassen. Weitere Informationen zum Thema Business-Objects sind [13-24] zu entnehmen.

Einer der Ursprünge der vorliegenden Arbeit liegt in der Beobachtung der Business Object Domain Task Force der OMG und der Beschäftigung mit den dort vorgeschlagenen Ideen und Lösungsansätzen.

3 Design & Entwurf

3.1 Einleitung

Dieses Kapitel gibt einen Einblick in die "Wurzeln" bzw. "Quellen" der zu Grunde liegenden Ideen des eXBO-Ansatzes und trägt auf diese Weise zu einem besseren Verständnis der nachfolgenden detaillierten Beschreibungen der Einzelkomponenten bei.

Im ersten Abschnitt "Entwurfsmuster" werden zunächst diejenigen grundlegenden Überlegungen vorgestellt, die den Entwurf des eXBO-Frameworks am stärksten beeinflussten.

Im Abschnitt "Notwendige Komponenten für das Framework" werden die daraus folgenden Komponenten des Framework vorgestellt.

Der Abschnitt "Grundsätzliche Entwurfsentscheidungen" stellt grundlegende Entwurfsentscheidungen vor, die beim Aufbau des eXBO-Frameworks (und auch bei allen anderen eCommerce-Systemen) beachtet werden sollten, um wichtige Aspekte wie Integrationsfähigkeit, Ausbaufähigkeit, Skalierbarkeit u.a. optimal zu unterstützen.

Im letzten Abschnitt "Die Entwicklungsplattform" wird auf die gewählte Entwicklungsplattform WebObjects eingegangen.

3.2 Entwurfsmuster

Die methodische Suche nach Entwurfsmustern (englisch: "Design Patterns") gehört zu den besonders interessanten Gebieten innerhalb der objektorientierten Programmierung. Die Suche nach Entwurfsmustern umfasst dabei mehr als die Suche nach möglichst "vollkommenen", wiederverwendbaren Klassen bzw.

Komponenten. Entwurfsmuster zielen vielmehr darauf ab, gemeinsame Verhaltens- und Interaktionsmuster zu erkennen, die über die Ebene von Klassen hinausgehen.

Die Untersuchung von Mustern ist nicht auf den Bereich der Software-Technik beschränkt. Die Bedeutung von Mustern ist schon viel früher von anderen technischen Wissenschaften erkannt worden, wie etwa der Biologie, Chemie, Physik oder der Architektur. Die Bedeutung für die Software-Entwicklung, insbesondere bei der Konstruktion grosser und komplexer Software-Systeme wurde erst zu Beginn der letzten Dekade dieses Jahrhunderts erkannt.

Zu den wesentlichen Veröffentlichungen auf dem Gebiet der Entwurfsmuster (Design-Patterns) gehören die Arbeiten von Grady Booch, Bruce Anderson, Erich Gamma, Kent Beck oder J. Vlissides [34-45], um nur einige zu nennen. Grundlage vieler Arbeiten über Entwurfsmuster sind die Aufzeichnungen von Christopher Alexander, einem Gebäudearchitekten aus Berkeley, der schon in den 70er Jahren eine "neue", auf Entwurfsmustern basierende Architektur propagierte. Insbesondere sein Buch "The Timeless Way of Building" fand grossen Anklang bei vielen "Software-Architekten" von heute und wird entsprechend häufig zitiert [34].

In seinen Untersuchungen "der Komplexität" beobachtet er beispielsweise, dass "komplexe Systeme in den meisten Fällen aus einer kleinen Zahl von Teilsystemen bestehen, die in unterschiedlichen Verbindungen und Anordnungen organisiert sind". Mit anderen Worten: komplexe Systeme haben gemeinsame Muster. Diese Muster können die Wiederverwendung kleiner Komponenten einschliessen, wie etwa die Zellen, die sowohl in Pflanzen als auch in Tieren zu finden sind. Ebenso können auch grössere Strukturen, wie z.B. das Gefäßsystem, das Pflanzen und Tieren gemeinsam ist, wiedererkannt bzw. wiederverwendet werden.

In den oben genannten Arbeiten wird beschrieben, wie ähnliche "Muster" in gut strukturierten Software-Systemen wiederzufinden sind. Systeme, die auf den ersten Blick komplex und undurchdringbar erscheinen, werden durch das Suchen bzw. Auffinden von Mustern plötzlich verständlich und überschaubar. Hat man einmal die grundlegenden Muster erkannt, so können auch komplexeste Systeme als blosse Anordnung von verschiedenen, aber immer wiederkehrenden Mustern betrachtet werden. Die Anzahl der dabei voneinander verschiedenen Grundmuster ist oft erstaunlich gering.

Folgendes Zitat von Grady Booch [34] gibt die Einschätzung der Experten zu Entwurfsmustern wieder: "... ferner wird veranschaulicht, wie diese Muster die eingewurzelte Komplexität von Systemen per se zu vereinfachen vermögen."

In den nächsten Abschnitten dieses Kapitels werden diese Eigenschaften weiter beschrieben und anhand eines konkreten Beispiels gezeigt, wie zunächst komplex anmutende eCommerce-Software-Systeme durch die Identifikation von Mustern zu übersichtlichen bzw. überschaubaren Systemen werden.

Zuvor wird im folgenden Abschnitt auf die unterschiedlichen Kategorien von Entwurfsmustern und ihren Bezug zur vorliegenden Arbeit eingegangen.

Kategorien von Entwurfsmustern

Entwurfsmuster für Software existieren in unterschiedlichen Grössenordnungen und Abstraktionsniveaus. Sie werden im allgemeinen in die folgenden Kategorien eingeordnet:

- *Architekturmuster*: Architekturmuster drücken fundamentale Strukturkriterien und Architekturkonzepte eines Software-Systems aus. Sie beschreiben eine Menge von vordefinierten Subsystemen, deren Verantwortlichkeiten und Regeln sowie Hinweise, die zwischen ihnen bestehenden Beziehungen zu realisieren.

- *Entwurfsmuster*: Entwurfsmuster bieten ein Schema zur Verfeinerung von Subsystemen oder Komponenten. Ein Entwurfsmuster beschreibt häufig vorkommende Strukturen kommunizierender Komponenten.

- *Idiome*: Idiome sind in der Regel spezifisch für eine Programmiersprache. Sie beschreiben, wie man eine bestimmte Komponente, ihre Funktionalität oder Beziehungen zu anderen Komponenten realisieren kann.

Die Grenzen zwischen den Kategorien sind fliessend und vom konkreten Anwendungsfall abhängig. Die Software-Konstruktion mit Entwurfsmustern hat eine professionelle Unterstützung der Erstellung und Pflege grosser und komplexer Systeme zum Ziel. Zur Erreichung dieses Ziels werden sowohl die funktionalen als auch die nicht-funktionalen Eigenschaften explizit betrachtet.

Die erste Kategorie von Entwurfsmustern, die Architekturmuster, gehen häufig auf die nicht-funktionalen Eigenschaften eines Systems ein. Zu diesen, in der Software-Entwicklung immer wichtiger werdenden (und für eCommerce-Systeme besonders wichtigen) "nicht funktionalen" Eigenschaften, gehören z.B.: Wiederverwendbarkeit, flexible Änderbarkeit, Erweiterbarkeit, Anpassbarkeit, Wartbarkeit, Portierbarkeit, Zuverlässigkeit, Robustheit, Verfügbarkeit, Entwicklungsgeschwindigkeit, Testbarkeit oder die Integrationsfähigkeit in vorhandene IT-Systeme.

Diese "nicht-funktionalen" Eigenschaften sind wichtige Qualitätskriterien für Software-Systeme im allgemeinen und für eCommerce-Systeme im speziellen. Auf diese "nicht-funktionalen" Eigenschaften, die im grossen Maße von den gewählten Architekturmustern abhängen, wird im Abschnitt "Grundsätzliche Entwurfsentscheidungen" eingegangen.

In diesem und dem Kaptitel "Notwendige Komponenten für das Framework" wird auf die funktionalen Eigenschaften im Sinne von "Entwurfsmustern" und "Idiomen" eingegangen.

Nachfolgend wird anhand eines konkreten Beispiels näher auf Entwurfsmuster für eCommerce-Systeme eingegangen. Als Beispiel dafür wird ein eCommerce-Shop-System gewählt.

Entwurfsmuster am Beispiel eines eCommerce-Shop-Systems

Rollenverteilung

Analog zu den real existierenden Rollen (Käufer und Betreiber bzw. Shop-Inhaber) können eCommerce-Shop-Systeme in zwei verschiedene Bereiche aufgeteilt werden:

- die für den Kunden zugängliche "Storefront"
- den für den Betreiber und seine Mitarbeiter zugänglichen "Backoffice".

Gemäß der Aufgabenverteilung in grösseren Unternehmen kann der Backoffice in weitere Unterrollen aufgesplittet werden, wie z.B. "Kundenmanager", "Produktmanager", "Lagerverwaltung", "Bestellabwicklung" usw..

In der nachfolgenden Abbildung wird diese Rollenverteilung dargestellt:

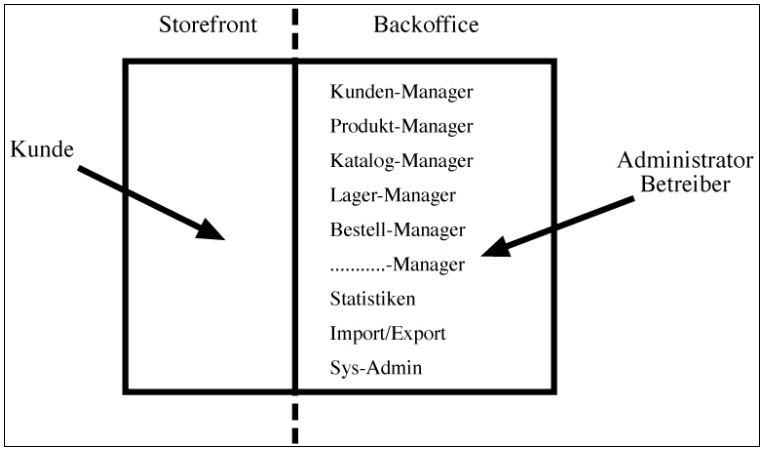

Abb 2: Rollenverteilung in eCommerce-Systemen (z.B. Shop)

Bereits die Betrachtung unterschiedlicher Rollen vereinfacht die Darstellung von eCommerce-Shop-Systemen zunächst in der Art, dass das System in zwei funktional voneinander getrennte Teile zerlegt werden kann (“Storefront” und “Backoffice”).

Diese Trennung könnte, wenn konsequent fortgeführt, sogar dazu führen, eine grössere Anwendung in mehrere überschaubare Teilanwendungen zu zerlegen. Im folgenden wird der Bereich “Backoffice” isoliert von dem Bereich “Storefront” betrachtet. Auf die Darstellung des im Normalfall wesentlich einfacher strukturierten “Storefront-Bereichs” wird im nachfolgenden verzichtet, da die eXBO-Konzepte ausreichend mit Hilfe des komplexeren “Backoffice”-Bereichs erläutert werden können.

Funktionen Backoffice

Der Backoffice eines eCommerce-Shop-Systems muss es ermöglichen, Warengruppen und Produkte einzupflegen und zu ändern, Kunden und Bestellungen zu verwalten, Zahlungssysteme und Lieferbestimmungen festzulegen, allgemeine Systemeinstellungen vorzunehmen und vieles mehr.

Anhand des Beispiels eines “Produktmanagers” werden zunächst die notwendigen Funktionalitäten und Bedienoberflächen, die für das Anlegen, Ändern, Kopieren, Löschen und Suchen - also für das Arbeiten mit Produkten - notwendig sind, erläutert. Ziel ist es, an diesem Beispiel ein “Muster” aufzuzeigen, nach dessen Vorbild (fast) alle “Manager” aufgebaut sind.

Später (im Kapitel 5 - “Implementierung”) wird dieses immer wiederkehrende Muster erneut aufgegriffen. Dort wird anhand dieses “Manager-Musters” die Konstruktion von generischen Komponenten (den so genannten „eXBO-Facilities“), die sich selbständig “erzeugen” und mit denen beliebige Objekte nach deren Wünschen bearbeitet werden können, erläutert.

In den nachfolgenden Abbildungen sind die am “Manager-Muster” beteiligten Komponenten dargestellt:

Abb 3: List-Page-Komponente

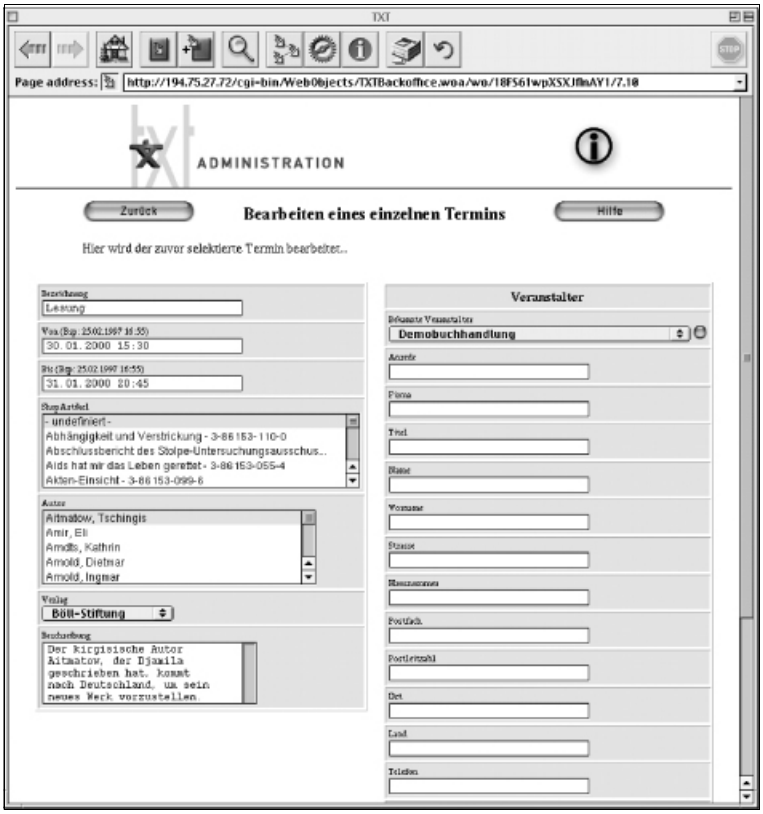

Abb 4: Edit-Page-Komponente

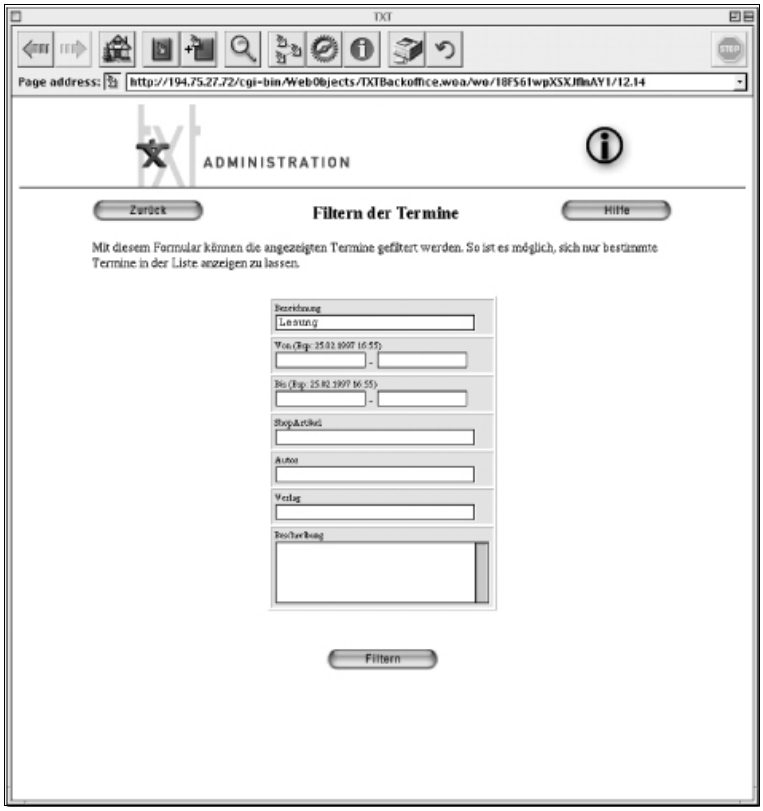

Abb 5: Search-Page-Komponente

Die List-Komponente listet die vorhandenen Produkte auf und ermöglicht das Selektieren einzelner Objekte. In ihr werden Funktionen wie Editieren, Neuanlegen oder Kopieren angeboten, die jeweils einen Wechsel zur Edit-Komponente nach sich ziehen. Komfortable List-Komponenten ermöglichen darüber hinaus die Sortierung nach verschiedenen Attributen (beispielsweise durch Klick auf die Spaltenüberschrift), oder ein "Batch-Processing" (vor- und zurückblättern einer zuvor festgelegten Anzahl von Objekten).

Mit Hilfe der Edit-Komponente kann auf alle Attribute eines Produktes zugegriffen werden. Das Anwählen der Speicherfunktionen in der Edit-Komponente führt zu der persistenten Speicherung des editierten Objektes.

Bei einer grossen Anzahl von Produkten kann das Auffinden eines speziellen Produktes oder einer bestimmten Gruppe von Produkten zu einer anstrengenden und zeitraubenden Aufgabe werden. Deshalb bieten die meisten Systeme Such- bzw. Filterfunktionen an, die es ermöglichen, bestimmte Objekte schneller

auffinden zu können. Die in Abbildung 5 dargestellte Suchkomponente bietet diese entsprechende Funktionalität (vergleichbar mit einer Internet-Suchmaschine).

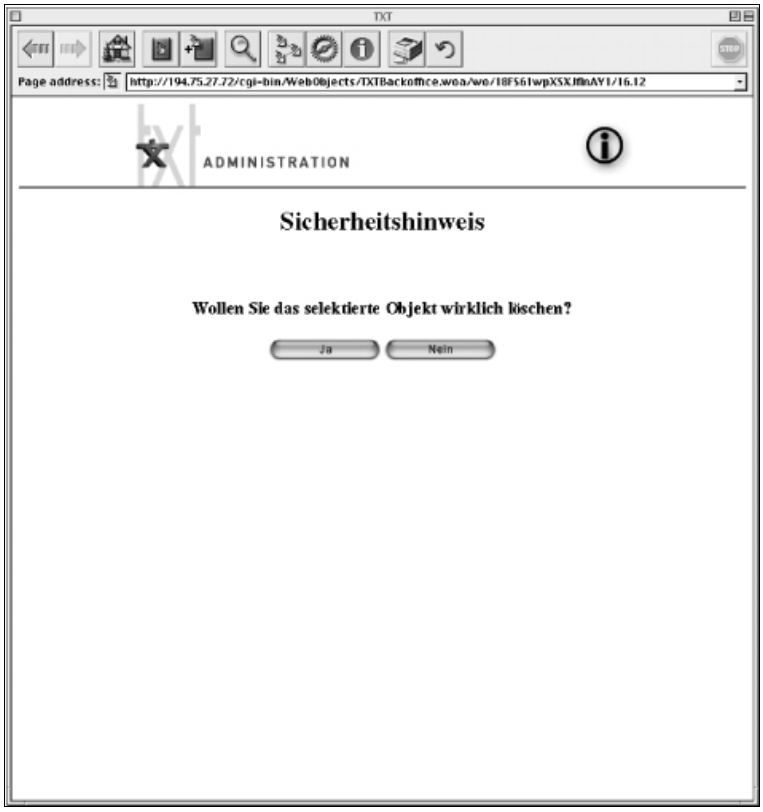

Abb 6: Hinweis-Dialog-Komponente

Abb 7: Info-Komponente

Die in den Abbildungen 6 und 7 dargestellten Komponenten können das "Muster" weiter verfeinern. Eine Hinweis-Komponente ermöglicht einen zusätzlichen Dialog mit dem Benutzer, um ihn z.B. vor den Folgen bestimmter Handlungen zu warnen und ein versehentliches Ausführen von Funktionen (z.B. das Löschen von Objekten) zu vermeiden.

Mit Hilfe einer Info-Komponente können komplexere Geschäftsobjekte, die beispielsweise viele Verbindungen bzw. Beziehungen zu anderen Objekten haben, übersichtlich und zusammenfassend dargestellt werden. Eine Info-Komponente könnte auch abgewandelt werden, um eine "Preview"-Funktionalität umzusetzen, mit der das spätere Erscheinungsbild in der "Storefront" kontrolliert werden kann.

Diese fünf Komponenten (List-, Edit-, Such-, Hinweis- und Info-Komponente) bieten bereits (fast) alle notwendigen Funktionen um Artikel-Objekte innerhalb eines eCommerce-Shop-Systems zu managen. Gelingt es dieses "Muster" auch für die anderen Geschäftsobjekte (innerhalb des Shop-System-Beispiels z.B.:

"Warengruppen", "Kunden", "Bestellungen", "Liefervarianten") fortzusetzen, so hat man bereits einen grossen Teil der gesamten Backoffice-Funktionalität umgesetzt. Dies soll mit der nachfolgenden Abbildung 8, die einen Entwurf für den "Backoffice" eines eCommerce-Shop-Systems darstellt, weiter verdeutlicht werden:

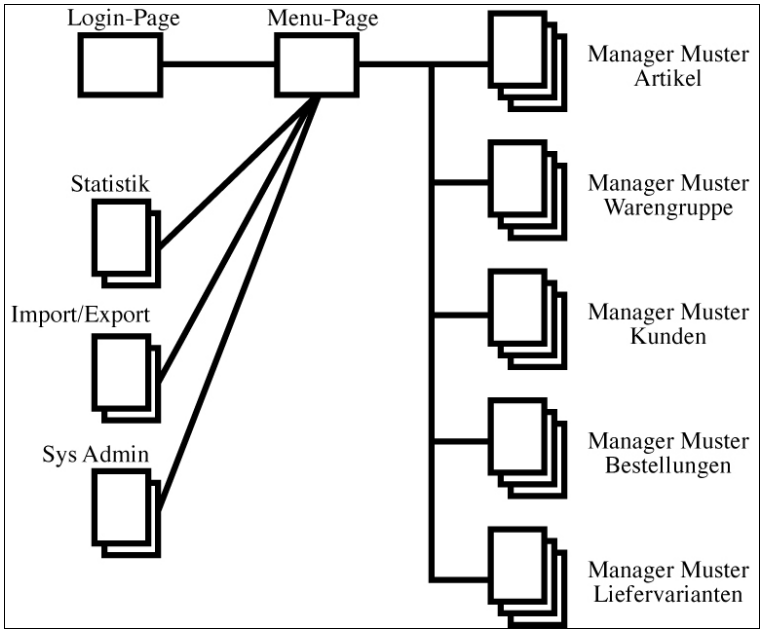

Abb 8: Entwurf für den "Backoffice" eines eCommerce-Systems

Innerhalb dieser Arbeit werden solche Muster, wie z.B. das "Manager-Muster" umgesetzt. Das Besondere an den "eXBO-Facilities", die im Rahmen dieser Arbeit entwickelt werden und Umsetzungen genau solcher Muster darstellen, ist deren "generische" Eigenschaft. Durch diese Eigenschaft wird erreicht, dass ein Muster nur ein einziges mal implementiert werden muss und danach von allen, auch von vorher unbekannten eXBO-Objekten genutzt werden kann.

Um dies genau nach den Wünschen bzw. Erfordernissen des jeweiligen eXBO-Objektes durchführen zu können, bedarf es der Kommunikation zwischen eXBO-Objekt und "eXBO-Facility": Bevor sich eine eXBO-Facility für ein eXBO-Objekt "selber generiert" teilt das eXBO-Objekt exakt mit, wie dies geschehen soll.

Als Beispiel soll an dieser Stelle die Abbildung 3 herangezogen werden. In ihr ist derjenige Teil eines "Manager-Musters" abgebildet, in dem zunächst alle "Artikel"-Objekte in einer Tabelle aufgelistet werden.

Bei der Implementierung einer solchen Seite wird im Normalfall zunächst spezifiziert, welche Attribute des Artikel-Objektes in welcher Spalte der Tabelle angezeigt werden sollen. In einem zweiten Schritt wird dann die Tabelle (bzw. die Seite), die die Artikel-Objekte auflistet, programmiert. Das gleiche geschieht normalerweise nicht nur für ein Geschäftsobjekt sondern für viele verschiedene, so dass für jeden Typ von Geschäftsobjekt (z.B. Artikel, Warengruppe, Rechnung) eine solche Komponente entsteht.

Im Falle von eXBO-Facilities existiert die angegebene Komponente nur ein einziges Mal. Objekte, die in der Tabelle dargestellt werden sollen, teilen z.B. der entsprechenden eXBO-Facility-List-Komponente unmittelbar vorher (zur Laufzeit des Programmes) mit, welche Attribute in der Tabelle, in welcher Reihenfolge, mit welcher Sortierung usw. dargestellt werden sollen. Die eXBO-Facility-List-Komponente generiert sich daraufhin exakt nach diesen Angaben selbst.

Bei längerer Betrachtung (von z.B. den bereits vorgestellten Komponenten des "Manager-Musters") erkennt man, dass die einzelnen Komponenten eines Musters wiederum aus verschiedenen, wiederverwendbaren Einzel-Elementen bestehen können. Zur Veranschaulichung werden in der nachfolgenden Abbildung 9 einige der möglichen Einzel-Elemente vorgestellt.

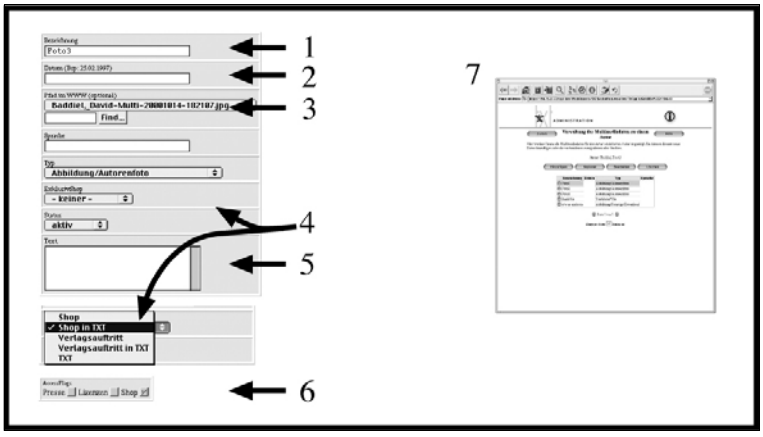

Abb 9: Elementarkomponenten einer Edit-Page

1. Normales Textfeld

2. Textfeld für die Eingabe eines Datums

3. Upload-Funktionalität, die es ermöglicht, Bilder und andere Multimedia-Daten vom lokalen Rechner zum Server zu transportieren

4. Verschiedene “Choices” bzw. “Pop-Up” oder “Pull-Down” - Menüs (aufklappende Auswahlboxen, die vordefinierte Möglichkeiten bieten)

5. Ein grösseres Textfeld für die Eingabe von längeren Beschreibungen

6. Schalter

7. Sub-List-Komponenten für die Bearbeitung von “Arrays” (z.B. verschiedene “Tagespunkte” eines “Veranstaltungs-Geschäfts-Objektes”)

Im Rahmen der bisherigen Arbeiten am eXBO-Framework konnten bisher ca. ein Dutzend solcher Elementarelemente identifiziert werden (vergleiche auch Kapitel 4 - “Spezifikation”).

Durch die eXBO-Konzepte wird es möglich, dass (wie bei dem “generischen” Aufbau der Tabelle im vorhergehenden Beispiel) Objekte nicht nur mitteilen können, wie sie dargestellt oder bearbeitet werden möchten, sondern dass sogar jedes einzelne Attribut eines Objektes diese Fähigkeit besitzt.
Das Attribut eines Artikel-Objektes mit dem z.B. ein Artikel-Bild bestimmt wird (nachfolgend “Picture-Attribut” genannt) kann z.B. selbstständig mitteilen: “Ich bin ein “Picture-Attribut”, wenn ich in einer Edit-Komponente dargestellt werde, dann möchte ich mit einer Upload-Funktionalität (die das Aufspielen von entsprechenden Bildern von Client-Rechnern zum Server-Rechner erlaubt) bearbeitbar sein”.

Eine entsprechende Edit-Page-Facility-Komponente berücksichtigt dann diesen “Wunsch” und generiert sich so, dass für dieses Attribut entsprechend eine funktionierende Upload-Funktionalität bereitgestellt wird.

Im nächsten Abschnitt "Notwendige Komponenten für das Framework" werden die grundlegenden Komponenten (die eXtended Attributes, die eXBO-Facilities, die eXtended BusinessObjects) vorgestellt, die zur Realisierung des eXBO-Konzeptes notwendig sind.

3.3 Notwendige Komponenten für das Framework

3.3.1 eXtended Attributs

Für alle gängigen Verarbeitungskomponenten (wie zum Beispiel normale oder formatierte Eingabefelder, Passwortfelder, Auswahllisten, Upload-Funktionen für Multimediadaten, Radio-Buttons usw.) und Verknüpfungsmöglichkeiten mit anderen Objekten (1-1, 1-n oder n-n Verknüpfungen, wie sie aus der Datenbanktheorie bekannt sind) sollen entsprechende, vorgefertigte eXtended Attributes implementiert werden.

Diese eXtended Attributs werden im nachfolgenden Kapitel "Spezifikation" detailiert dargestellt.

3.3.2 eXtended Business-Objects (eXBO)

eXBO sind wiederum Objekte, die mit Hilfe der "Elementarkomponenten" (den eXtended Attributs) aufgebaut werden. eXBO-Objekte enthalten, genau wie andere Geschäftobjekte auch, Geschäftsdaten und Businesslogik. Der wesentliche Unterschied liegt darin, dass ihre Attribute nicht aus einfachen Objekttypen (wie z.B. string, float, array, date) sondern aus eXtended Attributes bestehen.
Durch die Wiederverwendung von fertigen eXtended Attributes aus dem entsprechenden "eXtended Attributes Framework" kann der, bei der Implementierung der eXBO eventuell anfallende Mehraufwand sehr gering gehalten werden.

Die entscheidende Verbesserung bei der Konstruktion von Softwaresystemen, die das Potential der angesprochenen Steigerung der Entwicklungsgeschwindigkeit in sich birgt, wird durch einen zweiten Schritt ermöglicht:

Durch den Umstand, dass jedes eXtended Business Object, genauer betrachtet sogar jedes einzelne Attribut eines eXtended Business Objects, selbständig mitteilen kann, wie es dargestellt und verarbeitet werden möchte, wird es möglich "generische" Verarbeitungs- und Darstellungskomponenten zu konstruieren, die in der Lage sind, jedes beliebige eXtended Business Object, optimal und exakt nach dessen Wünschen anzeigen und verarbeiten zu können. Das gilt sowohl für vorher bekannte Objekte als auch für unvorhersehbare Objekte, die zum Beispiel erst im nachhinein entworfen werden.

In Kapitel 6 "Validierung" werden typische Beispiele für eXtended BusinessObjects vorgestellt.

3.3.3 eXBO-Facilities

Diese im vorhergehenden Abschnitt als "generische Verarbeitungs- und Darstellungskomponenten" bezeichneten Komponenten, werden im folgenden als eXBO-Facilities (Facilities == "Möglichkeiten" für eXtended Business-Objects) bezeichnet.

Diese eXBO-Facilities sind zum Teil hoch komplexe Komponenten, die in der Lage sind, sich selber nach den Wünschen von beliebigen (auch von vorher vollkommen unbekannten, z.B. erst im nachhinein entwickelten) eXBO-Objekten zu generieren.

Die entsprechenden Informationen, die angeben, wie dies geschehen soll, bekommen die eXBO-Facilities entweder:

- von den einzelnen Attributen (den eXtended Attributs)
- von einem Objekt (einem eXBO-Objekt) oder
- von übergeordneten Session- oder Applikationsweiten Objekten bzw. "der Anwendungslogik" des Programms; die Anwendungslogik könnte z.B. der in Abbildung 3 dargestellten List-Komponente mitteilen, dass sie für den gerade aktiven Nuzer die Delete-Funktion ausblenden soll

Diese eXBO-Facility-Komponenten werden im fünften Kapitel „Implementierung" detailiert dargestellt.

3.4 Grundsätzliche Entwurfsentscheidungen

In diesem Abschnitt werden grundlegende Überlegungen bzw. Kriterien vorgestellt, die bei der Entwicklung des eXBO-Frameworks Beachtung fanden . Diese Grundlagen und Rahmenbedingungen sollten bei der Entwicklung von eCommerce-Systemen im Allgemeinen beachtet werden, da an eCommerce-Systeme besondere Anforderungen, z.B. bezüglich Integrationsfähigkeit in vorhande IT-Systeme, Sicherheit, Skalierbarkeit usw., gestellt werden.

Unabhängigkeit von Datenquellen

Bei vielen vorhandenen eCommerce-Systemen ist die Kopplung zur Datenbank meist so eng, dass ein Wechsel zu einem anderen Datenbankmanagementsystem sehr schwer fällt oder gar unmöglich ist. Diese starke Abhängigkeit soll beim Entwurf des eXBO-Frameworks vermieden werden.

Ursachen für eine starke Abhängigkeit ist beispielsweise ein häufiges Absetzen von SQL-Statements direkt aus dem Source-Code. Der Wechsel zu einer anderen Datenbank (und dementsprechend zu einer anderen SQL-Dialektik) würde Änderungen im gesamten Source-Code nach sich ziehen.

Eine andere Ursache kann darin liegen, dass zuviel Logik (z.B. in Form von Stored-Procedures) in die Datenbank gelegt wird. Die entsprechende Funktionalität geht bei einem Wechsel des Datenbankherstellers verloren.

Am leistungsfähigsten erwiesen sich bisher Konzepte von Adaptern, wie sie aus der objektorientierten Programmierung bekannt sind. Im Source-Code sieht dabei der Datenbankzugriff bzw. das Ansprechen des Objektes immer gleich aus. Die spezialisierten Adapter-Objekte für die jeweiligen Datenbanksysteme erzeugen daraus automatisch korrektes SQL für das jeweilige Datenbanksystem. Für einen Wechsel der Datenbank genügt das Austauschen des Datenbank-Adapters.

In der Praxis genügt es für eCommerce-Systeme häufig nicht sich auf relationale Datenbanksysteme als Datenspeicher bzw. Datenquelle zu begrenzen. Einer der wichtigsten Aspekte für eCommerce-Systeme ist ihre Möglichkeit, sich in bestehende IT-Infrastrukturen einzupassen. Unter diesen gibt es eine Vielzahl

von Altsystemen (z.B. hierarchische Datenbanken), historisch gewachsene Systeme (z.B. Flugabfragesysteme mit einer eigenen Abfragesprache), Warenwirtschaftssysteme, Enterprise Resource Planungssysteme (ERP) usw..

eCommerce-Systeme sollten eine Vielzahl von Datenquellen von Haus aus unterstützen und auf die Integration von verschiedensten Datenquellen vorbereitet sein. Die Leistungsfähigsten unter ihnen sollten es darüber hinaus erlauben, mehrere verschiedene Datenquellen gleichzeitig einzubinden. Sie dazu [1-2] und [33].

Kapselung der Business-Logik

Die konsequente Kapselung der Business-Logik gegenüber der Benutzeroberfläche und gegenüber der Datenquelle hat Auswirkungen auf die Wiederverwendbarkeit, die Flexibilität, die Unabhängigkeit von Endgeräten, die Skalierbarkeit, die Entwicklungsgeschwindigkeit und die Möglichkeit, Design und Technik parallel zu entwickeln.

Wird beispielsweise die Business-Logik zu grossen Teilen in die Benutzeroberfläche gebracht (typisch z.B. für Java-Script, JScript, ASP-Anwendungen) sind grosse Mengen Source-Code im HTML enthalten. Eine durchschnittliche Design-Agentur kann dann nicht mehr damit umgehen, um z.B. die Anwendung den “Corporate-Identity”-Vorgaben des Unternehmens anzupassen. Umgekehrt wäre es für die Entwicklungsabteilung fatal, wenn unbedarfte Designer aus Versehen auch nur ein einziges Zeichen aus dem Source-Code löschen oder verändern würden.

Mit Systemen, die statt Source-Code nur kleine “Tags” im HTML-Code aufweisen, wird schon heute eine parallele Entwicklung von Design und Anwendung betrieben, was die gesamte Entwicklungszeit (“Time To Web”) verkürzt. Soll eine vorhandene eCommerce-Anwendung später verschiedenste Endgeräte (siehe z.B. die aktuellen Entwicklungen im UMTS-Umfeld in Bezug auf mobile Geräte oder die neuesten Bestrebungen für ein digitales, internetfähiges Fernsehen) und somit verschiedenste Oberflächen unterstützen, so müsste die Business-Logik ebenfalls nicht neu implementiert werden.

Umgekehrt führt eine zu enge Bindung an die Datenbank (durch direkte SQL-Aufrufe oder Stored-Procedures) zu einer zu starken Abhängigkeit von einem Datenbankenhersteller. Das eCommerce-System wird unflexibel bezüglich der möglichen Datenquellen.

Sind die Bereiche Oberfläche, Business-Logik und Datenspeicherung nicht konsequent durch entsprechende Zugriffsschichten voneinander getrennt, fällt auch die Aufteilung auf verschiedene voneinander getrennte Rechner zwecks Skalierung schwerer.

In der Praxis preisen viele Systeme ihre Mehrschichten-Architektur an, meistens ist dabei aber die Kopplung zwischen den einzelnen Ebenen zu starr (z.B. durch die schon angesprochenen direkten SQL-Aufrufe), so dass die Flexibilität in beide Richtungen (Datenquellen und Oberflächen) entsprechend eingeschränkt wird.

Für das eXBO-Framework soll eine Kapselung der Business-Logik gewählt werden, die es gestattet die "Kapseln" nach Bedarf auszutauschen, um so flexibel mit verschiedensten Datenquellen und Endgeräten arbeiten zu können.

Architekturen für eCommerce-Systeme

Abgeleitet aus den vorangegangenen Forderungen bezüglich der Kapselung der Business-Logik, dem Wunsch nach Unabhängigkeit von Datenbankherstellern bzw. gänzlicher Unabhängigkeit von Datenquellen und dem Anspruch, verschiedenste Endgeräte unterstützen zu können, wird eine Drei-Ebenen-Architektur mit konsequenter Trennung der einzelnen Schichten für den Aufbau von eCommerce-Anwendungen vorgeschlagen.

Nur wenn die drei grundlegenden Ebenen (Datenhaltung, Business-Logik und User-Interface) durch zusätzliche „Trennschichten" (die z.B. in Form von Adapter-Objekten) konsequent voneinander getrennt werden, ist eine maximale Flexibilität bezüglich verwendeter Datenhaltungssysteme und eingesetzter Endgeräte möglich.

Die Beachtung dieser grundlegenden architekturellen Überlegungen hat entscheidenden Einfluss auf die Integrationsfähigkeit des eXBO-Systems in bestehende und zukünftige IT-Infrastrukturen.

Zustandsverwaltung

Spricht man heutzutage über eCommerce- oder eGovernment-Anwendungen, so sind implizit fast immer Webanwendungen gemeint, die orts- und plattform-unabhängig mit einem Webbrowser bedient werden können. Alle diese Webanwendungen haben eines gemeinsam: sie nutzen das Internet. Das Internet wurde primär für die Präsentation und den Austausch von Daten entwickelt und ist vollkommen zustandslos, bzw. kennt von Haus aus keine Sessions oder gar Zustandsverwaltungen für voneinander getrennte Sessions.

Die eCommerce-Anwendungen müssen also eine eigene Session- bzw. Zustandverwaltung besitzen um Dinge, wie z.B. getrennte Warenkörbe für viele parallele Nutzer, zu ermöglichen. Um einen Request (Anfrage eines Nutzers) eindeutig zu einer Session zuordnen zu können müssen zwei Dinge gewährleistet sein. Eine Session muss eindeutig identifizierbar sein, d.h. sie muss eine eindeutige Kennung (ID) besitzen und sie muss dem Client (Webbrowser) und dem Server (Webanwendung) bekannt sein. Neben dieser Session-ID müssen, für jede Session getrennt, weitere relevante Daten (wie z.B. die oben angesprochenen Artikel im Warenkorb) gespeichert werden.

Die Speicherung bzw. Verwaltung aller sessionrelevanten Informationen kann auf verschiedene Weisen und an verschiedenen Orten vorgenommen werden. Grob lässt sich zwischen Sessionverwaltung beim Client und Sessionverwaltung beim Server unterscheiden. Bei der Sessionverwaltung im Client werden die session- bzw. zustandsrelevanten Daten über das Internet zum Client übertragen und auf dem Rechner des Benutzers gespeichert. Die Speicherung beim Client kann z.B. in Cookies, in versteckten Formularfeldern oder in Proxy-Objekten (z.B. Java-Applets) vorgenommen werden. Bei der Zustandverwaltung im Server bleiben alle Daten im Serverrechner und werden entweder in der Anwendung oder in einer Datenbank gespeichert.

Beide Verfahren haben Vor- und Nachteile. Die Zustandverwaltung im Client, die in der Praxis häufig durch Cookies realisiert wird, ist technisch einfach zu realisieren und wird deshalb vor allem von “low-end”-Systemen eingesetzt. Diese Methode hat aber mehrere entscheidende Nachteile:
Zustandsrelevante Daten, die eventuell sicherheitsrelevante Informationen enthalten, werden ständig über das Netz hin und her übertragen.

Obwohl die Mehrheit der Cookies vergleichsweise harmlos ist, sind Cookies schon des öfteren in den Mittelpunkt von Sicherheitsdiskussionen gerückt. Denn Cookies sind immerhin Dateien bzw. Informationen, die von "Fremden" auf dem eigenen Rechner "eingepflanzt" werden und die ihre Informationen bei der nächsten Verbindung zum Internet wieder preisgeben können.

Die Cookie-Methode funktioniert nur wenn der Nutzer auf der Client-Seite Cookies akzeptiert, was ein grosser Teil von Internet-Nutzern aber ablehnt. Ohne den Cookie allerdings ist die Anwendung dann nicht mehr funktionsfähig. "Man kann es fast schon als peinlich bezeichnen, wenn eine Website gänzlich auf diese Methode baut und somit für den Benutzer, der – aus welchen Gründen auch immer – Cookies ablehnt, unbrauchbar ist" [49].

Die Sessionverwaltung im Server stösst weder auf Akzeptanzprobleme noch auf Sicherheitsbedenken. Dafür ist sie wesentlich schwieriger umzusetzen. Ist die Sessionverwaltung nicht Bestandteil einer leistungsfähigen Middleware und muss selbst programmiert werden, ist ein ganzes Team von erfahrenen Entwicklern notwendig, um eine stabile und hochskalierbare Zustandverwaltung zu implementieren. Eine serverseitige Zustandverwaltung ist demnach selten bei "low-end"-Systemen anzutreffen und eher dem professionellen bzw. High-End-Bereich zuzuordnen.

Um den Mehraufwand einer Eigenimplementierung zu vermeiden, wird bei der Entwicklung des eXBO-Frameworks eine entsprechend leistungsfähige Middleware bzw. ein entsprechendes, leistungsfähiges Entwicklungs- und Deployment-System ausgewählt, welches den Anforderungen gerecht wird.

Hardware- und Betriebssystem-Plattformen

Bezüglich der Untersuchung verschiedener Betriebssysteme gibt es, wie für viele andere Kriterien auch, kontroverse Diskussionen. Bei den einen ist z.B. die Unterstützung von Windows ein absolutes Muss, um der grossen Verbreitung dieses Systems gerecht zu werden. Andere wiederum, z.B. solche mit extremen Sicherheitsanforderungen lehnen Systeme, die ausschliesslich auf Windows basieren, von vornherein ab, da Rechner mit diesen Betriebssystemen in der Regel nicht sicher genug sind oder gar Hintertüren für Geheimdienste aufweisen [6-10].

Allgemein kann man beobachten, dass vor allem kostengünstige Low-End-Systeme oft nur auf der Windows-Plattform verfügbar sind, während High-End-Systeme eine modulare Architektur aufweisen, welche die Verteilung der einzelnen Komponenten auf verschiedene Betriebssysteme und Hardwareplattformen erlaubt. Zusätzlich gilt noch immer die Einschätzung, dass Windows basierte Systeme preisgünstiger, auf Unix basierte Systeme hingegen wesentlich ausfallsicherer und besser zu skalieren sind [50].

Es wäre demzufolge von Vorteil, wenn das eXBO-Framework auf mindestens zwei verschiedenen Plattformen (Windows und Unix) lauffähig ist.

Performance

Die Performance bzw. die Antwortzeiten eines Systems sind ein wesentliches Kriterium für Web-Anwendungen. Je kürzer die Antwortzeit desto besser. Sinkt die Performance unter ein gewisses Mindestmaß ist mit einer Akzeptanz-Einbuße bis hin zum Totalverlust von Kunden zu rechnen.

Die Performance von einem System verschlechtert sich mit zunehmender Zahl von parallelen Usern und Grösse des Datenbankinhalts (Gegenmaßnahmen siehe Skalierung). Allerdings unterscheiden sich verschiedene Systeme von Haus aus (auch bei gleicher Userzahl und DB-Grösse) stark in ihrer Performance. Hierzu sei z.B. auf die Vergleichsstudie zu Applicationservern vom Französischen Institut SQLIngenierie hingewiesen [51].

Entsprechend sorgfältig ist die Auswahl des Applicationservers und der übrigen Komponenten durchzuführen. Gleiches gilt für die Implementierungsarbeiten und die Tests. Im Idealfall sollten alle Einzelkomponenten bereits bei der Spezifikation, der anschliessenden Implementierung und bei jedem Einzeltest auf ihre Performance hin geplant und überprüft werden.

Skalierbarkeit

Die Skalierbarkeit eines Systems beschreibt die Möglichkeit, sich bei wachsendem Erfolg den ständig steigenden Anforderungen an Performance und Inhalt anzupassen. Es kann mitunter fatal für ein Unternehmen sein, wenn es seinem eigenen Wachstum nicht durch ein leistungsfähiges System entgegenkommen kann. Für jeden eCommerce-Anbieter kann ein instabiles System oder zu lange

Ladezeiten den Todesstoß bedeuten. Plant man ein eCommerce-System, das im Erfolgsfall von sehr vielen Anwendern genutzt wird, sollte man sich von vornherein im klaren darüber sein, dass von web-basierten eCommerce-Systemen ein Höchstmaß an Skalierbarkeit gefordert wird. Die eCommerce-Systeme müssen dabei in zwei verschiedenen Richtungen skalierbar sein:

Skalierbarkeit bezüglich der Antwortzeiten des Systems - auch bei Hunderten oder Tausenden von parallelen Zugriffen soll sich die Antwortzeit nur minimal verschlechtern.

Skalierbarkeit bezüglich der Inhalte - genau wie eine grosse Anzahl von parallelen Zugriffen kann auch eine grosse Anzahl von z.B. Artikeln in der Datenbank zur erheblichen Verlängerung von Antwortzeiten führen.

Die Qualität der Entwickler und die Ausführlichkeit von diesbezüglichen Tests bei der Systementwicklung sind leider nicht die einzigen Faktoren für eine gute Skalierbarkeit. Hochskalierbare eCommerce-Systeme fordern optimale Performance von jeder einzelnen Komponente: Webserver, Applicationserver, Datenbankserver, Betriebssystem, Hardware und Implementierung.

Für fast alle Komponenten gilt dabei, dass höhere Performance durch einen höheren Preis eingekauft werden muss.

Die zuvor angesprochene Kapselung der eigentlichen Programmfunktionalität zur Oberfläche und zur Datenquelle kommt auch hier, bei den Skalierungs- bzw. Wachstumsmöglichkeiten eines Systems, voll zum Tragen. Sind die Ebenen sauber voneinander getrennt, kann ein Unternehmen mit einer freien oder preisgünstigen Datenbank starten und später, bei entsprechendem Erfolg, durch den Austausch einiger weniger Komponenten zu einer professionellen, hochperformanten Datenbanklösung wechseln.

Die Anschaffung eines entsprechenden Datenbanksystems, wie z.B. eines Oracle-Enterprise-Datenbank-Managementsystems für Webanwendungen (welche mit mehreren hunderttausend Euro Lizenzgebühren einhergeht) sind für die Startphasen der meisten Unternehmen unrealistisch. Wurde aber zu Beginn auf ein System gesetzt, das nicht genügend ausbaufähig ist, muss eine kostspielige Neuentwicklung gestartet werden.

Viel schlimmer noch ist die Tatsache, dass die Engpässe des Altsystems bis zur Inbetriebnahme des neuen hingenommen werden müssen. Ein Problem, das in der schnelllebigen Internet-Welt, "in der die Konkurrenz nur einen Klick weiter ist", zu Image- und Akzeptanz-Problemen führt, und letztendlich den Verlust von vielen Nutzern bedeuten kann.

Durch die Beachtung der bereits zuvor erwähnten architekturellen Grundlagen und der Verwendung eines leistungsfähigen und voll skalierbaren Applicationservers wird die Entwicklung des eXBO-Frameworks von vornherein auf hohe Skalierbarkeit ausgelegt.

Eingesetzte Technologien und Programmiersprachen

Perl, PHP, Java, C++, ObjectiveC, Java-Script, Virtual-Basic, JScript, ASP, WebObjects und Java-Servlets sind nur eine Auswahl von Programmiersprachen bzw. Technologien, die in der Praxis für die Entwicklung von dynamischen Webseiten eingesetzt werden. Eine genaue Analyse der verschiedenen Technologien würde den Rahmen dieses Kapitels sprengen.

In diesem Abschnitt soll nur auf einige allgemeine Einschätzungen hingewiesen werden. Die seit Jahren anerkannten Vorteile der objektorientierten Programmierung, wie z.B. bessere Wartbarkeit, Wiederverwendung, grössere Zuverlässigkeit, und nicht zuletzt erheblich kürzere Entwicklungszeiten (vor allem bei grösseren Projekten) sind nur durch die Wahl einer echten objektorientierten Programmiersprache bzw. Programmierumgebung gegeben.

Ausserdem ist zu beachten, dass die im Abschnitt "Kapselung der Business-Logik" angesprochenen architekturellen Grundsatzentscheidungen durch die Wahl der verwendeten Technologie beeinflusst, bzw. oft sogar vorweggenommen werden. Wird z.B. Java-Script (das ins HTML hineingeschrieben wird) eingesetzt, um Teile der Business-Logik zu implementieren, ist eine saubere Trennung zwischen Oberfläche und Logik nicht mehr möglich.

In diesem Zusammenhang bleibt anzumerken, dass eine der Hauptursachen für den Erfolg des World Wide Web in der "Demokratisierung" der Informationsgesellschaft liegt. Die Grundidee des World Wide Web ist es, dass Jeder von jedem Ort, mit jedem Rechner und jedem Betriebssystem gleichberechtigt und unkompliziert, Zugang zu Informationen bekommt.

Entwicklungen, wie z.B. Java-Script- oder JScript-Anwendungen, die von vielen Webbrowsern nicht interpretiert werden können, verletzen die grundlegenden Ideen des World Wide Web [11-12], [48].

Bei der Planung eines eCommerce-Systems sollte auch an die Kosten gedacht werden, die entstehen, wenn das Unternehmen grossen Erfolg hat und die Einzelkomponenten: Webserver, Applicationserver, Datenbankserver, Betriebssystem und Hardware dementsprechend aufgerüstet bzw. skaliert werden müssen. Nach Möglichkeit sollte die zur Systementwicklung herangezogene Anwendungs- und Programmierumgebung eine entsprechende Aufrüstung flexibel unterstützen, damit die getätigten Investitionen in die eCommerce-Anwendung gesichert sind und man nicht zu einer Neuentwicklung gezwungen wird.

Die Entwicklung des eXBO-Framework soll vollständig objektorientiert erfolgen. Die zum Einsatz kommenden Benutzeroberflächen werden dabei dem HTML Standard entspechen und ohne proprietäre Technologien auskommen.

3.5 Die Entwicklungsplattform

Um ein ehrgeiziges Projekt wie das eXBO-Framework, effizient und zukunftssicher umsetzen zu können, bedarf es zum einen einer modernen, vielseitigen, leistungsfähigen und vor allem erprobten Entwicklungsplattform. Um dies auch sicher und fehlerfrei realisieren zu können, bedarf es darüber hinaus eines nicht unerheblichen Anteils an Erfahrungen und Sicherheit im Umgang mit diesem System.

Aus diesen Gründen wird als Entwicklungsumgebung im Rahmen dieser Arbeit die Entwicklungsplattform “WebObjects” der Firma Apple eingesetzt.

Die Entwicklungsumgebung

Die WebObjects Entwicklungsumgebung stellt mehrere verschiedene Frameworks zur Verfügung. Zu diesen Frameworks gehören z.B. das “Enterprise Objects Framework”, das “WebObjects-Framework”, das “Foundation-Framework” und das “AppKit-Framework”. Insbesondere von dem “WebObjects-Framework”, dem

“Enterprise Objects Framework” und dem “Foundation-Framework” werden zahlreiche sehr nützliche Klassen zur Verfügung gestellt, die bei der Erstellung des eXBO Frameworks genutzt bzw. eingebunden werden können. Die einzelnen Frameworks wurden bereits im Kapitel 2 “Bestandsaufnahme” im Abschnitt “Frameworks für eCommerce-Systeme” genauer vorgestellt.

WebObjects ist eine der bewährtesten und industriell erprobtesten objektorientierten Anwendungs- und Entwicklungsumgebungen überhaupt. Die Ursprünge der WebObjects-Technologie liegen bei der Firma NeXT, die sich jahrelang als technologischer Vorreiter und Pionier auf dem Gebiet der objektorientierten Programmierung auszeichnete. Nach dem Zusammenschluss der Firmen Apple und NeXT wurde aus der ständig weiterentwickelten NeXT-Step-Entwicklungsumgebung schliesslich WebObjects.

Während einer tiefergehenden Recherche im eCommerce-Umfeld, besonders bei der Untersuchung von anspruchsvollen und erfolgreichen eCommerce-Anwendungen im World Wide Web stösst man nach wie vor zwangsläufig auf den Begriff WebObjects.

Grosse und grösste eCommerce-Anwendungen, wie z.B. die Anwendungen von Deutsche Bank (Online-Banking und Online-Brokering), Apple, Consors, Deutsche Telekom, Reiseveranstalter TUI, American Stock Exchance, eTrade, Schweizerischer Bankenverein, mehrere B2B-Portale, führende B2B-Anwendungen (wie z.B. Ariba), NASA, US-Department of Defense oder dem United States Postal Service, um nur einige zu nennen - sie alle wurden mit Hilfe von WebObjects entwickelt und werden auch mit der WebObjects-Deployment-Umgebung betrieben [52].

Für eCommerce-Systeme wichtige Punkte wie z.B. die strikte Trennung der Datenhaltungs-, Anwendungs-Logik- und User-Interface-Ebenen, optimale Skalierbarkeit oder die Wiederverwendbarkeit von Objekten und Komponenten werden von WebObjects optimal unterstützt. Funktionalitäten, wie das für Internetanwendungen besonders wichtige serverseitige Session-Management oder auch die Anbindung von relationalen Datenbanken an objektorientierte Anwendungen, werden durch fertige, höchst leistungsfähige Framework-Komponenten zur Verfügung gestellt.

Die Deploymentumgebung

Die WebObjects-Deploymentumgebung bietet folgende Komponenten, die die Deploymentphase unterstützen:

WebObjects-Applicationsserver

Die wohl wichtigste Komponente jeder Deploymentumgebung ist der Applicationsserver, der eine oder mehrere parallele Instanzen einer Anwendung unterstützt.

Skalierungs- und Lastverteilungskomponenten

Skalierungs- und Lastverteilungskomponenten werden z.B. eingesetzt, wenn eine eCommerce Anwendung gleichzeitig von mehreren hunderten oder tausenden von Usern benutzt wird und die Antwort- bzw. Reaktionszeiten eines Systems unzureichend werden. Bei Anwendungen die intern intensivste Rechenoperationen durchführen, kann es auch schon bei wenigen parallelen Usern zu Engpässen kommen. Um diesen Engpässen entgegenzutreten, wird deshalb versucht, die einzelnen Komponenten eines Deployment-Systems (insbesondere Webserver, Applicationsserver und Datenbankserver) auf verschiedene Rechner aufzuteilen. In einem weiteren Schritt können mehrere parallele Rechner mit Applicationsservern (die jeweils wieder mehrere parallele Instanzen der eigentlichen Anwendung betreiben) installiert werden. Die Skalierungs- und Lastverteilungskomponenten, die die Zuordnung und Verteilung vom HTTP-Requests und -Responses managen, werden im Fall der WebObjetcs-Umgebung als fertige Komponenten mitgeliefert und müssen nicht selbst programmiert werden.

Automatische Restart- und Recovery-Komponenten

Diese Komponenten sorgen z.B. dafür, dass Instanzen einer Anwendung, die z.B. Aufgrund von Software- oder Hardwarefehlern “abgestürzt” sind, wieder automatisch hochgefahren werden und für neue Requests zur Verfügung stehen.

Überwachungs- und Monitoring-Komponenten

Mit diesen Komponenten bzw. "Werkzeugen" können sämtliche Vorgänge des Deployments überwacht und aufgezeichnet werden. Zum Beispiel: Zeit und Anzahl von "Abstürzen" einzelner Anwendungsinstanzen, die Systemlast oder auch die Antwortzeiten des Systems zu verschiedenen Zeitpunkten.

Last-Test-Komponenten

Mit diesen Komponenten, mit denen eine grosse Zahl von parallelen Usern simuliert wird, kann eine fertige Deployment-Installation z.B. schon vor dem eigentlichen Wirkbetrieb mit entsprechenden Lasten getestet und "vermessen" werden.

Im Fall von WebObjects sind alle genannten Komponenten bereits Bestandteil der WebObjects-Deployment-Umgebung. Vor allem, um der weit überlegenen Marketing-Macht von einigen Konkurrenzprodukten (z.B. IBM Websphere oder BEA Weblogic) entgegenzutreten wurde der Verkaufspreis für die voll skalierbare "unlimited" Version von über US $ 50.000 auf unter US $ 900 gesenkt.

4 Spezifikation

4.1 Einleitung

Das hier folgende Kapitel beschäftigt sich mit der genauen Spezifikation der zu entwickelnden Komponenten. Es wurden insgesamt 10 verschiedene Interface-Elemente ausgemacht, die der Framework für eine möglichst umfassende Umsetzung zur Verfügung stellen sollte. Diese werden hier einzeln spezifiziert und mit ihren Daten und Methoden beschrieben.

4.2 Allgemeine Attribut-Eigenschaften

Jedes der hier aufgeführten Attribute hat die folgenden Eigenschaften mit allen anderen XAttribute gemeinsam. Damit diese nicht bei jedem erneut aufgeführt werden, werden sie hier in diesem Abschnitt zusammengefasst. Es handelt sich dabei um generelle Eigenschaften der Objekte, die für die Funktion des Frameworks Objekt-übergreifend notwendig sind.

Implementiert sind diese Eigenschaften in einer gemeinsammen Superklasse namens XAttribute. Das folgende UML-Diagramm beschreibt die Variablen und Methoden der Klasse.

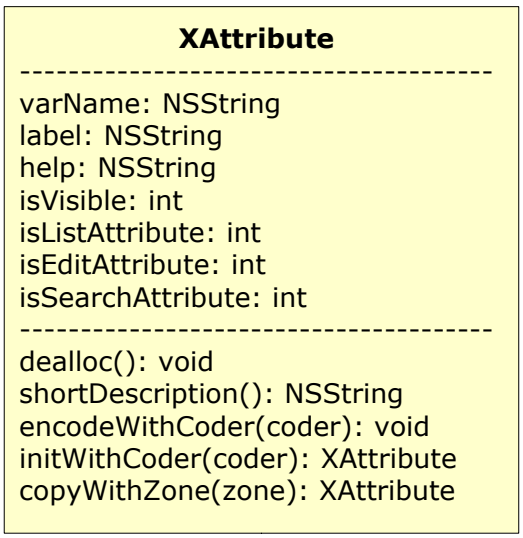

Abb 10: UML-Diagramm XAttribute

Die erste Variable mit den Namen "varName" steht für den Namen selbiger. Er ist notwendig, damit über die WebObjects-typischen Accessor-Methoden auf die Variable zugeriffen werden kann.

Das "label" ist eine kurze textuelle Bezeichnung des Attributes. Sie wird üblicherweise als Name im Interface angezeigt und erlaubt die bessere Identifizierung durch den Benutzer.

Der String "help" ist für eine optionale Hilfe zu dem Attribut gedacht. Sie könnte beispielsweise als Mouse-Over-Effekt im Interface angezeigt werden und würde dem Benutzer so eine weitere Hilfestellung bei seiner Arbeit bieten.

Die folgenden vier Variablen sind als int angelegt, obwohl man sie eigentlich als boolean charakterisieren sollte. In Objective-C hat sich aber im Laufe der letzten Jahre bei mir gezeigt, dass der dort übliche BOOL auf SUNs Solaris zu Problemen führt und unerwartete Reaktionen zeigt. Da man Wahrscheitswerte generell auch als 0 oder 1 interpretieren kann, habe ich mich daher entschieden, generell int statt BOOL zu verwenden. In anderen Sprachen spricht nichts gegen die Verwendung von boolean (beispielsweise in Java).

Unter "isVisible" versteht man die aktuelle Sichtbarkeit des Attributes. Dieses Flag ist dafür gedacht, dass man das entsprechende Eingabefeld im Interface zur besseren Übersicht auch selektiv ausblenden kann.

Bei "isListAttribute" wird festgelegt, ob das Attribut in der normalen Listendarstellung erscheinen soll. Ähnliches gilt für die beiden Variablen "isEditAttribute" und "isSearchAttribute", welche das Auftreten in der Edit- und Search-Komponente im Interface regeln. So lassen sich verschiedene Attribute in bestimmten Kombinationen editieren und/oder anzeigen. Biepielsweise wird das Attribut mit dem Datum der letzten Bearbeitung zwar in der Liste angezeigt, aber nicht zum editieren angeboten. Andersherum wird die lange Textbeschreibung zwar beim Editieren und der Suche einbezogen, in der Listendarstellung wird dieses aber ausgeblendet.

Soweit zu den allgemeinen Variablen jedes XAttribute. Dazu gesellen sich einige Standardmethoden, welche ebenfalls allen gemein sind. Jede dieser Methoden muss aber von der Unterklasse erweitert und an ihre Variablen angepasst werden. Es genügt nicht, die bereits definierte Methode aufzurufen – es sollte aber auch nicht vergessen werden. Die Implementierung in der jeweiligen Klasse sollte sich um die jeweils eigenen Variablen und Zustände kümmern, während durch den Aufruf der Supermethode auch die Variablen der Superklasse berücksichtigt werden.

Die Methode "(void) dealloc" dient der Speicherverwaltung und ist eine für Objective-C typische Methode. Andere Sprachen implementieren die Speicherverwaltung auf andere Art und Weise. Wann immer in Objective-C ein Objekt freigegeben wird, erfolgt der Aufruf dieser Methode. Sie sollte daher dafür sorgen, dass auch die Referenzen auf andere Objekte, die dann nicht mehr benötigt werden, freigegeben werden.

Hinter "(NSString *) shortDescription" verbirgt sich eine Methode, welche eine umfassende Beschreibung des Objektes mit all seinen Variablen zurückgibt. Diese Beschreibung dient in erster Linie, um Veränderungen an dem XAttribute festzustellen. So kann in einem bestimmten Kontext beispielsweise geprüft werden, ob sich das Attribut im Vergleich zu einem vorherigen Zustand verändert hat und eine entsprechend notwendige Aktion ausgeführt werden. So lassen sich Cache-Mechanismen implementieren, die aufwendige Berechnungen zwischenspeichern und die Geschwindigkeit der Applikation positiv beeinflussen.

Die beiden Methoden "(void) encodeWithCoder: coder" und "(XAttribute *) initWithCoder: coder" dienen zur Archivierung der Attribute. Sie stellen unter Objective-C eine standardisierte Möglichkeit der Serialisierung dar. Wann immer ein Objekt archiviert werden soll - man könnte auch sagen, es wird gespeichert –

dann benutzt der sogenannte “Archiver” die Methode zum kodieren der Daten. Umgekehrt verwendet der sogenannte “Unarchiver” die andere der beiden Methoden, um das Objekt wiederherzustellen. So kann man rekursiv ganze Objekt-Bäume archivieren und zu einem späteren Zeitpunkt wieder laden.

Die vorerst letzte Methode ist “(XAttribute *) copyWithZone: zone”. Sie wird zum Kopieren eines Objektes verwendet. Anders als in Java kann man nicht jedes Objekt einfach per “clone” duplizieren. In Objective-C muss man bei jedem Objekt explizit festlegen, welche Daten in die Kopie übertragen werden sollen und welche nicht.

Soweit zu den allgemeinen Variablen und Methoden. Im Folgenden wird auf die Besonderheiten der jeweiligen XAttribute eingegangen. Je nach Typ werden nun unterschiedliche Variablen und Methoden benötigt, die einzeln beschrieben werden sollen.

4.3 XAttributeText

Das wohl einfachste und grundlegendste Element ist das Text-Element. Es stellt die einfachste Form der der Texteingabe zur Verfügung und wird von der Mehrzahl aller Objekte verwendet werden.

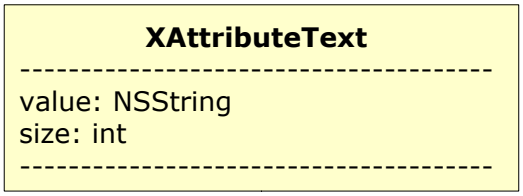

Abb 11: UML-Diagramm XAttributeText

Zu den allgemeinen Variablen und Methoden des XAttribute kommen beim XAttributeText nur diese zwei dazu. Die eine ist das “value”, also der Text, der sich hinter dem Attribut versteckt. Die zweite Variable “size” gibt die gewünschte Grösse des Eingabefeldes in Zeichen an. Dabei wird aber nicht die Länge des Textes begrenzt sondern lediglich das optische Erscheinungsbild beeinflusst.

Bei diesem XAttribute werden keine zusätzlichen Methoden definiert. Es kommt mit den allgemeinen Methoden der Oberklasse aus.

4.4 XAttributeFloat

Ein weiteres oft verwendetes Attribut ist das XAttributeFloat. Es wird für die Eingabe von einfachen Dezimalzahlen verwendet. Gegenüber dem XAtributeText hat es den Vorteil, dass es direkt auf die Dezimalstellen der Zahlen eingehen kann. Intern werden die dahinter liegenden Daten als Float- und nicht als String-Objekte verwaltet. Dadurch lassen sie sich wesentlich effizienter für Berechnungen nutzen.

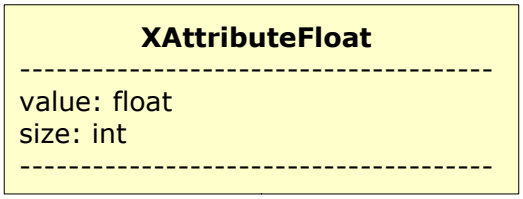

Abb 12: UML-Diagramm XAttributeFloat

Ähnlich wie das XAttributeText ergänzt auch das XAttributeFloat die Superklasse um zwei weitere Variablen. Diesmal ist das "value" aber vom Typ float und repräsentiert somit eine Gleitkommazahl. Die Variable "size" definiert abermals die Länge des dazugehörigen Eingabefeldes im Interface und nicht die maximale Grösse des Wertes.

4.5 XAttributeDate

Für Datumsangaben eignet sich das XAttributeDate am besten. Es formtiert die die Ausgabe in einer gut lesbaren Form und erlaubt die Eingabe des Datums auf verschiedene Art und Weisen. So kann man ein Datum beispielsweise in der Form von "01.08.2002" oder aber auch als "morgen" angeben. Das Attribut sorgt automatisch für eine einheitliche interne Organisation des Datums.

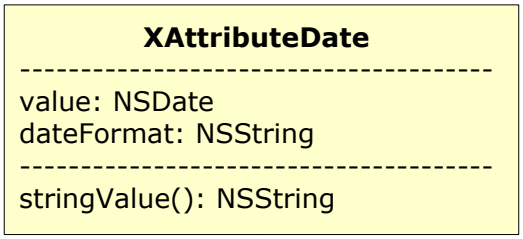

Abb 13: UML-Diagramm XAttributeDate

Das XAttributeDate verwendet für das "value" ein Objekt vom Typ NSDate und repräsentiert somit ein Datum inklusive Uhrzeit. Das dazugehörige "dateFormat" definiert die Formatierung des Datums für die Ein- und Ausgabe im Interface. So lässt sich ein Datum beispielsweise durch "1998/04/12" oder aber auch als "12th of January 1998" darstellen. Die gewünschte Darstellungsweise wird in dieser Variablen hinterlegt.

Das XAttributeDate ergänzt die Klasse zusätzlich durch eine Methode "stringValue". Diese gibt das entsprechend formatierte Datum als String zurück und vereinheitlich so das Erscheinungsbild im Interface.

4.6 XAttributeInteger

Ähnlich dem XAttributeFloat gibt es auch ein XAttributeInteger. Der Unterschied zwischen den beiden Attributen besteht in der Art und Weise, wie sie mit Zahlenwerten umgehen. Während das XAttributeFloat die Nachkommastellen erhält und auch im Interface darstellt, werden diese beim XAttributeInteger vernachlässigt und abgeschnitten. Daraus ergibt sich auch die vorangige Verwendeung des Integer für einfache Zahlenangaben wie die Anzahl als Eigenschaft eines Objektes während der Float eher für Preisangaben oder genaue mathematische Zahlenwerte verwendet wird.

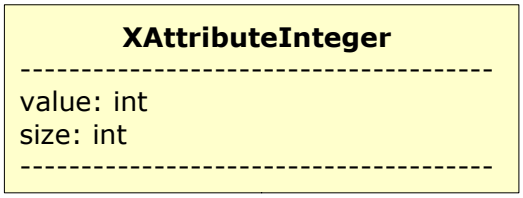

Abb 14: UML-Diagramm XAttributeInteger

Selbstverständlich hat auch das XAttributeInteger wieder ein "value" mit dem entsprechenden Typ – diesmal ein int. Die Variable "size" definiert abermals die Grösse des Eingabefeldes und nicht den maximal möglichen Wert.

4.7 XAttributeTextLong

Das XAttributeTextLong gleicht im grunde dem XAttributeText. Der entscheidende Unterschied besteht in der Art und Weise der Darstellung im Interface. Während das XAttributeText einzeilig dargestellt wird, sieht man beim XAttributeTextLong ein mehrzeiliges Eingabefeld, welches einen wesentlichen längeren Text sowie einige einfache Formatierungen erlaubt.

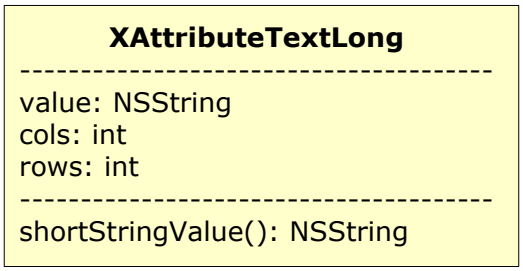

Abb 15: UML-Diagramm XAttributeTextLong

Das "value" ist wie beim XAttributeText auch hier vom Typ NSString. Die beiden int-Values "cols" und "rows" definieren die Grösse des Eingabefeldes im Interface durch die Zahl der Zeilen und Spalten.

Ausserdem bringt das XAttributeTextLong noch eine Methode "shortStringValue" mit, welche eine kurze Version des referenzierten Textes zurückgibt. Sie wird beispielsweise in der tabellarischen Übersicht über die entsprechenden Objekte verwendet.

4.8 XAttributeCheckBox

Die CheckBox ermöglicht das Setzen eines Wahrheitswertes im Interface. Sie wird in Form eines einfachen kleinen Kästchens dargestellt, welches man mit einem Häkchen aktiveren oder deaktiveren kann. Der jeweilige Zustand entspricht dann "true" oder "false".

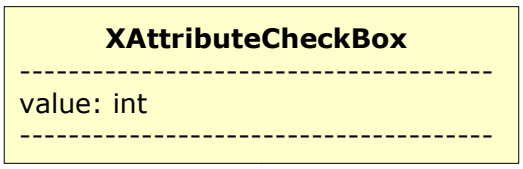

Abb 16: UML-Diagramm XAttributeCheckBox

Das XAttributeCheckBox kommt mit einem einfachen "value" vom Typ int aus. Weitere Variablen werden hier nicht benötigt, da das entsprechende Interface-Element keine nennenswerten Parameter unterstützt.

4.9 XAttributeChoice

Mit dem XAttributeChoice hat der Benutzer die Möglichkeit, aus einer Reihe von vorgegebenen Werten einen einzelnen auszuwählen und zu aktivieren. Ein mögliches Beispiel ist die Auswahl eines Landes aus einer Liste aller existierender Länder.

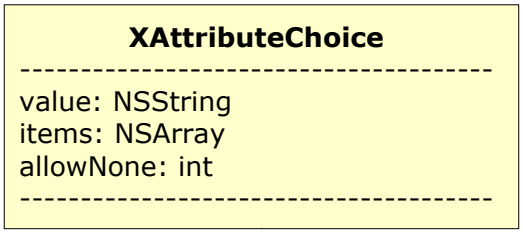

Abb 17: UML-Diagramm XAttributeChoice

Das "value" ist in diesem Fall vom Typ NSString und dient wie bei allen anderen XAttribute als Variable mit dem eigentlichen aktiven Inhalt. Das NSArray "items" hällt beliebig viele Objekte vom Typ NSString, welche alle im Interface als Auswahl angeboten werden. Alle darin enthaltenen Objekte sind potentielle Objekte für die Variable "value". Die dritte Variable "allowNone" vom Typ int legt fest, ob im Interface auch die leere Auswahl möglich ist – also auch keine der vorgegebenen Möglichkeiten gewählt werden darf. Man kann die leere Auswahl auch oft als unbestimmten Wert deuten.

4.10 XAttributeMultiChoice

Während man bei dem einfachen XAttributeChoice immer nur ein Element aus der Liste der Vorgaben auswählen kann ermöglicht die MultiChoice die Auswahl mehrerer Elemente. Dagegen ist jedoch die leere Auswahl nicht möglich. Wie auch bei der einfachen Choice kann man die Vorgaben dabei aber nicht verändern sondern lediglich selektieren.

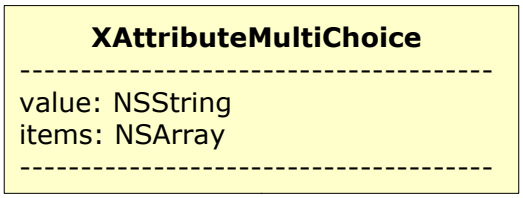

Abb 18: UML-Diagramm XAttributeMultiChoice

Die Variablen sind hier die gleichen wie bei dem XAttributeChoice. Lediglich das nicht unterstützte "allowNone" fällt diesmal weg.

4.11 XAttributeSublist

Das XAttributeSublist stellt eines der komplexesten Attribute dar, die der Framework zur Verfügung stellt. Oft bestehen Daten nicht nur aus einer einfachen Ansammlung von Werten. In vielen Fällen gibt es auch eine Reihe von untergeordneten Objekten, welche in direkter Beziehung zu dem Datensatz stehen. Ein mögliches Beispiel sind die veröffentlichten Alben eines Künstlers, die in der Datenstruktur diesem direkt untergeordnet werden.
Im Interface wird eine Sublist immer durch einen einfachen Button repräsentiert, welcher vom aktuellen Datensatz auf eine Tabelle wechselt, welche die Bearbeitung aller diesem Attribut zugeordneten Daten ermöglicht. Dabei wird in der Datenstruktur eine Ebene abgestiegen und es gelten die gleichen Grundregeln wie für alle anderen Daten. Man kann mit Hilfe des XAttributeSublist auf diese Art und Weise einen beliebig komplexen Baum von Daten bilden.

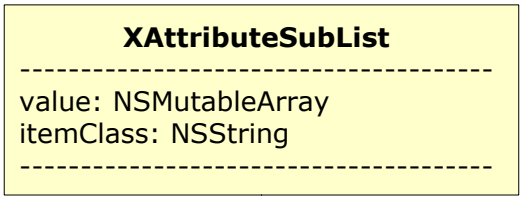

Abb 19: UML-Diagramm XAttributeSubList

Hier haben wir eine bisher noch ungenutzte Klasse für das "value" vom XAttributeSubList. Ein NSMutableArray ist ein Array, dessen Inhalt – also die Referenzen auf die Objekte – zur Laufzeit geändert werden können. Das ermöglicht es uns neue Objekte hinzuzufügen oder bereits existierende Objekte aus der Liste herauszunehmen. Aufgrund der häufigen Änderungen an diesem Attribute sind diese Möglichkeiten von grosser Wichtigkeit.

Die zweite Variable mit dem Namen "itemClass" ist ein String, der den Namen der Klasse beinhaltet, deren Instanzen in dem Array referenziert werden. Auf diese Weise können wir sicherstellen, das nur die gewünschten Objekte im Array bearbeitet und angelegt werden. Ein NSArray und auch das NSMutableArray kann ähnlich dem Vector in Java Objekte verschiedener Klassen aufnehmen, weshalb diese Variable von Bedeutung ist.

Der Umfang und die Vielseitigkeit dieses XAttribute wird in späteren Kapiteln ausführlicher behandelt. In dieser Spezifikation gehe ich daher nur wenig auf die zahlreichen Möglichkeiten vom XAttributeSubList ein.

4.12 XAttributeUpload

Das vorerst letzte Attribute steht für die Möglichkeit eines Uploads von Daten. Angenommen man verwaltet eine Reihe von Büchern in einem entsprechenden Shop, dann bietet dieses Attribute die Möglichkeit, Bilder, Texte oder beliebige andere Dateien zu dem Objekt auf den Server zu übertragen. Auf diese Weise hat der Benutzer die Möglichkeit, ergänzendes Material zum Datensatz abzulegen, welches nicht durch ein direktes Attribut vorgesehen war, aber trotzdem in seinem Shop angeboten werden soll.

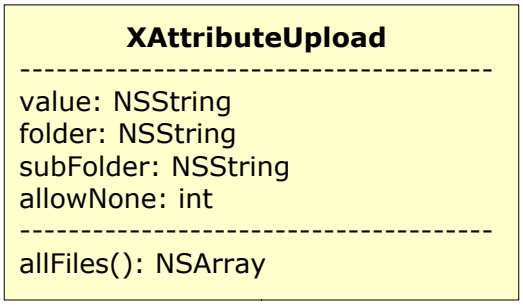

Abb 20: UML-Diagramm XAttributeUpload

Der Upload ist ebenfalls eines der eher komplexeren Attribute. Das "value" ist abermals vom Typ NSString und referenziert die aktuell selektierte Datei. Die Variable "folder" legt fest, in welchem Verzeichnis die verwalteten Dateien auf der Festplatte des Servers abgelegt werden. Dabei wird der "subFolder" noch an diesen angehangen. Die Variable "subFolder" wird von anderen Objekten beeinflusst, die das XAttributeUpload in sich verwenden – also anderen eXtended BusinessObjects. Dadurch kann man die Dateien auf der Festplatte in eine vernünftige Ordnung bringen und auch ausserhalb der Applikation den Überblick behalten. Aus der Sicht des Anwenders spielen diese beiden Variablen aber keine Rolle, da sie für ihn völlig unsichtbar im Hintergrund verwendet werden. Der Benutzer sieht lediglich die Dateien, die mit diesem Attribut verwaltet werden. Die int-Variable "allowNone" steht abermals dafür, ob auch die leere Auswahl erlaubt ist oder nicht. Im Interface wird für die Darstellung unter anderem auch ein PopUp verwendet, der dann die leere Auswahl entsprechend zulässt.

Die Methode "allFiles" gibt direkt alle Pfade zu allen vom XAttributeUpload verwalteten Dateien als Array zurück. Sie wird innerhalb des Frameworks an verschiedenen Stellen benötigt und bietet so einen einheitlichen Zugriff auf die Dateien.

Auch dieses XAttribute wird in späteren Kapiteln ausführlicher beschrieben, da viele Möglichkeiten erst im Umgang mit dem dazugehörigen Interfaceelementen beschrieben werden können.

4.13 Zusammenfassung

Damit ist die Vorstellung der einzelnen Attribute abgeschlossen. Ein wichtiges Kriterium bei der Konzeption des Frameworks bestand unter anderem darin, ein möglichst flexibles und trotzdem umfassendes Werkzeug zur Verfügung zu stellen, dass den Benutzer auch bei einer im WebBrowser ablaufenden Oberfläche möglichst nah an den gewohnten Abläufen der Bedienung hält. So braucht man sich nicht abermals mit neuen Bedienelementen auseinandersetzen und kann ohne eine grössere Eingewöhnung direkt mit dem Interface arbeiten.
Die Liste der XAttribute könnte natürlich noch um weitere ergänzt werden. Die hier vorgestellten Elemente bilden aber bereits eine gute Grundlage für die Umsetzung der Idee. Weitere Spezial-Attribute, beispielsweise für die Eingabe von Farben oder von Telefonnummern, wären denkbar, würden aber verhältnismäßig selten Anwendung finden und werden daher im Rahmen des Diploms nicht konzipiert. Sie liessen sich aber mit den gleichen Mitteln unter Beachtung der vorgegebenen Protokolle problemlos integrieren und könnten dann sofort und direkt in bestehende Anwendungen integriert werden.
Allein mit den hier gegebenen Attributen lassen sich innerhalb kürzester Zeit komplexe und vielseitige Backoffice-Anwendungen realisieren, die keinen Vergleich mit bestehenden Lösungen scheuen brauchen. Der grosse Vorteil besteht in der wesentlich effizienteren Realsierung und den damit verbundenen Kosteneinsparungen bei der Umsetzung von Projekten.

5 Implementierung

Unter Ausnutzung der in den vorangegangenen Kapiteln beschriebenen eXtended Attributes wird im Folgenden auf die Implementierung der bereits in Kapitel 3 "Design und Entwurf" in den Abschnitten 3.2 "Entwurfsmuster" und 3.3.3 "eXBO-Facilities" vorgestellten eXBO-Facilities eingegangen.

5.1 eXBO-Facility List

Bevor auf technische Details eingegangen wird, wird zunächst die Funktionalität dieses Elements vorgestellt. Obwohl in der folgenden Abbildung eine konkrete Implementierung vorgestellt ist, sei an dieser Stelle angemerkt, dass die optische Erscheinung von HTML-Designern jederzeit verändert bzw. angepasst werden kann und an dieser Stelle unerheblich ist. Allein die Funktionalität soll betrachtet werden:

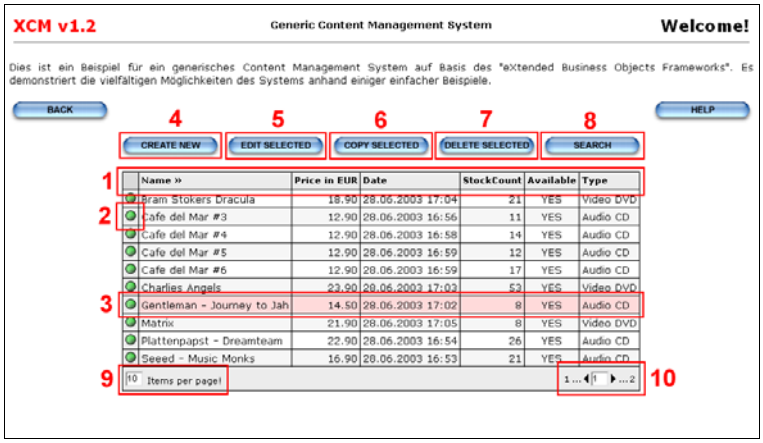

Abb 21: eXBO-Facility List

1. Hier werden die für die Listendarstellung vorgesehenen Attribute der Objekte dargestellt. Das eXBO-Objekt gibt die Spalten und deren Reihenflge vor. Die Bezeichnungen der Spalten können von den tatsächlichen

Variablennamen abweichen und werden ebenfalls durch das eXBO-Objekt festgelegt. Durch einen Klick auf eine beliebige Spaltenüberschrift kann nach dieser sortiert werden. Ein wiederholter Klick sortiert die Objekte in umgekehrter Reihenfolge. In diesem Beispiel sind die Objekte aufsteigend nach ihrem Namen sortiert, was man an dem kleinen Symbol neben der entsprechenden Spalte erkennen kann.

2. Das kleine Symbol in der linken Spalte der Tabelle dient der Auswahl eines Objektes. Die entsprechende Zeile wird daraufhin farblich hervorgehoben. In diesem Beispiel ist sie rot hinterlegt.

3. Eine solche Zeile stellt die ausgewählten Attribute des Objektes dar. Hier ist das Objekt gerade selektiert.

4. Mit diesem Button kann jederzeit ein neues Objekt des entsprechenden Typs erzeugt werden. Es wird automatisch in die Edit-Komponente verzweigt und das Objekt zur weiteren Bearbeitung seiner Eigenschaften dargestellt. Mehr dazu im folgenden Abschnitt.

5. Mit dieser Funktion kann das gerade selektierte Objekt bearbeitet werden. Alle seine aktuellen Eigenschaften werden in der Edit-Komponente dargestellt.

6. Ein selektiertes Objekt kann jederzeit dupliziert werden. In diesem Fall wird eine entsprechende Kopie angefertigt und in der Edit-Komponente dargestellt. Auf diese Weise spart man sich bei sehr ähnlichen Objekten einiges an Arbeit. Die Implementierung der eXBO-Objekte achtet dabei aber darauf, dass bestimmte, möglicherweise eindeutige Attribute, des Objektes in geeigneter Art und Weise verändert oder weggelassen werden.

7. Ein Klick auf diesen Button löscht das gerade selektierte Objekt aus dem Datenbestand. Der von ihm alloziierte Speicher wird freigegeben. Um ein versehentliches Löschen zu vermeiden, wird vor dem eigentlichen Löschen nochmals eine Sicherheitsabfrage eingeblendet.

8. Durch den Button “Search” gelangt man zur eXBO-Facility Search, welche ein detailiertes Filtern aller vorhandenen Objekte ermöglicht. Sie wird in einem der folgenden Abschnitte nochmals genauer vorgestellt.

9. In diesem Feld kann man die gewünschte Zahl der gleichzeitig in der Tabelle dargestellten Objekte festlegen. Vorallem ältere Browser haben mit langen und komplexen Tabellen Schwierigkeiten in der Darstellung. Daher kommt diese Funktion nicht nur dem Überblick sondern auch der Performance entgegen.

10. Sollten mehr Objekte existieren, als in eine einseitige Tabellendarstellung passen, so kann man an dieser Stelle auf die jeweils folgenden Seiten wechseln. Bei sehr vielen Seiten kann man zudem durch die direkte Eingabe einer Seitenzahl diese auch direkt anwählen.

Die Objekte, die in der Tabelle bzw. in der eXBO-List-Komponente dargestellt werden, müssen dieser zuvor übergeben werden. Im Rahmen der hier vorgestellten Implementierung geschieht dies durch ein übergeordnetes Application-Object.

Um die vorgestellten Eigenschaften erfüllen zu können (vor allem um sie für jedes beliebige Objekt erfüllen zu können), benötigt die eXBO-Komponente "List" eine Reihe von Attributen und Methoden. In der folgenden Abbildung ist das UML-Klassendiagramm für diese Listkomponente dargestellt.

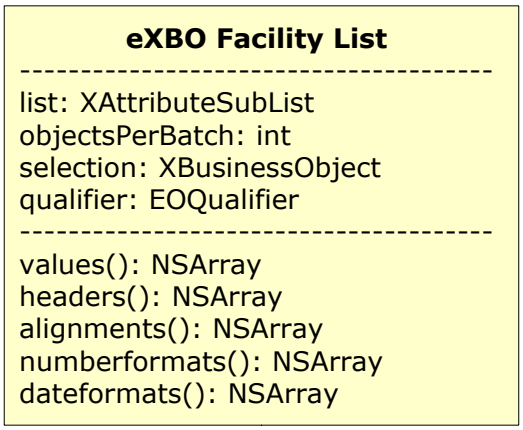

Abb 22: UML-Diagramm eXBO-Facility List

Die erste Variable "list" beinhaltet alle Objekte, die in der eXBO-Facility List dargestellt werden sollen. Es handelt sich dabei durchgehend um eXBO-Objekte, die die entsprechenden Anforderungen für die Darstellung erfüllen.

Unter “objectsPerBatch” wird die Anzahl der gleichzeitig auf einer Seite der Tabelle darzustellenden Objekte abgelegt. Die Variable ist direkt mit dem entsprechenden Eingabefeld auf der eXBO-Facility List verknüpft.

Die Variable “selection” referenziert das aktuell selektierte Objekt der Liste. Es kann immer nur ein Objekt selektiert werden, auf welches sich dann weitere Aktionen wie das Bearbeiten oder Löschen beziehen.

Der “qualifier” ist das aktuell gültige Filterkriterium für die Liste. Per default ist dieses anfangs leer und erlaubt somit die Auflistung aller unter “list” referenzierten Objekte. Wenn über die eXBO-Facility Search jedoch ein Filter definiert wird, so wird dieser der eXBO-Facility List über diese Variable bekannt gemacht. Daraufhin werden nur noch die Objekte dargestellt, die dem Filter entsprechen.

Die Methode “values” dient der Tabelle als Zugriffsmöglichkeit auf alle darzustellenden Objekt-Inhalte. Deren Ergebnis wird in einer entsprechenden Schleife als Tabelleninhalt generiert. Die hier enthaltenen Werte sind werden zuvor mit dem aktuell geltenden Filter bearbeitet, sodass nur die jeweils notwendigen Werte übergeben werden.

Mit der Methode “headers” fragt die Tabelle nach den jeweils darzustellenden Tabellenspalten. In dem zurückgegebenen Array werden die entsprechenden Spaltenüberschriften in der notwendigen Reihenfolge abgelegt.

Jede Spalte der Tabelle erfordert unter Umständen eine eigene Orientierung. Während Strings überlicherweise links ausgerichtet werden, eignet sich bei Zahlen eher eine rechts ausgerichtete Darstellung. Über die Methode “alignments” kann die eXBO-Facility List die Ausrichtung der jeweiligen Spalten erfragen und bei der Darstellung berücksichtigen.

Zahlen sind nicht gleich Zahlen. Jenachdem ob ich einen Preis, eine Gleitkommazahl oder einfach nur einen ganzzahligen Wert darstellen möchte, erfordert dies in der Tabelle unterschiedliche Formate. Über die Methode “numberformats” kann sich die eXBO-Facility list über die entsprechenden Notwendigkeiten informieren.

Die Methode "dateformats" erfüllt den gleichen Zweck wie die Methode "numberformats" für Datumsangaben. Ein eXBO-Objekt kann über sein XAttributeDate selbst festlegen, in welcher Form ein Datum angezeigt werden soll. Mal ist es nur die Jahreszahl, mal die Uhrzeit und mal ein vollständiges Datum inklusive der Uhrzeit.

Ein weiterer Bestandteil der eXBO-List-Komponente ist deren dynamische HTML-Oberfläche, auf deren Aufbau und Funktionsweise im folgenden eingegangen wird.

Nachdem bereits die zur Verfügung stehenden Methoden, Attribute und deren Herkunft erläutert wurden, wird mit der folgenden Abbildung auf die Funktionsweise der dynamischen HTML-Oberfläche der List-Komponente eingegangen:

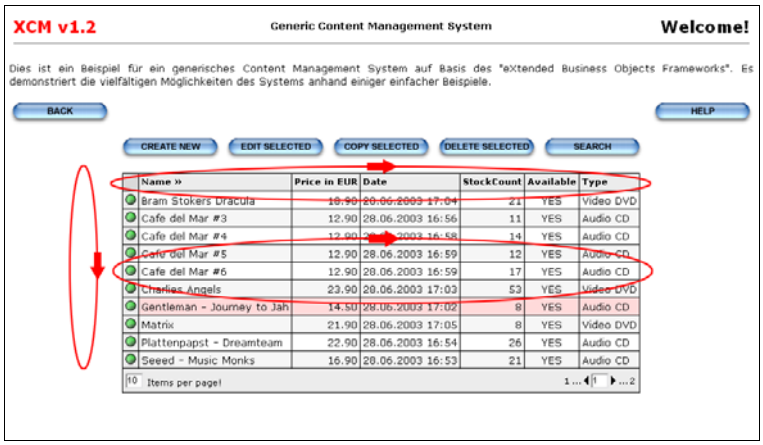

Abb 23: Dynamische HTML-Oberfläche der eXBO-Facility List

Damit die Darstellung der eXBO-Facility List mit allen in Frage kommenden eXBO-Objekten funktioniert, muss sich ihr Inhalt möglichst dynamisch generieren. Bei der Implementierung der Komponente gab es mehere variable Punkte, die nicht von vornherein bekannt sind. Daher baut sich die Darstellung im Wesentlichen aus drei grundlegenden Schleifen auf.

Die erste Schleife erzeugt die notwendigen Spaltenüberschriften der Tabelle. Dabei kommt die Methode "headers" zum Einsatz, die diese in ihrer korrekten Reihenfolge durch direktes Befragen der Objekte ermittelt und zurückgibt.

Daraufhin folgt eine zweite Schleife, die über alle darzustellenden Objekte iteriert und mit diesen jeweils eine dritte Schleife ausführt. Die innere Schleife befragt dann jedes darzustellende Objekte nach seinen Spaltenwerten. Dabei kommt die Methode "values" zum Einsatz. So wird für jedes Objekte eine eígene Tabellenzeile generiert.

An dieser Stelle soll angemerkt werden, dass alle hier vorgestellten eXBO-Facility-Komponenten voll skalierbar sind. Die durchgeführten Testläufe mit den Komponenten des Manager-Musters (mit 10, 100, 1000, 100.000 und 500.000 Business-Objects) ergaben nur unwesentliche Zeitdifferenzen (kleiner als 0,8 Sekunden). Auf die Beziehungen zwischen den einzelnen eXBO-Facilities wird am Ende dieses Kapitels noch einmal gesondert eingegangen.

5.2 eXBO-Facility Edit

Wie zuvor bei der eXBO-Facility List wird zunächst auf die Funktionalität der Komponente eingegangen. Die eXBO-Facility Edit dient dazu, neue oder bereits vorhandene eXBO-Objekte bearbeiten zu können. Als Bestandteil des Manager-Musters (s. Kapitel 3.2 "Entwurfsmuster") wird die eXBO-Facility Edit typischerweise nach der eXBO-Facility List aufgerufen. Das in der eXBO-Facility List selektierte Objekt wird der eXBO-Facility Edit übergeben. Alle Attribute des entsprechenden Objektes werden in der eXBO-Facility Edit dargestellt und können mit den jeweiligen User-Interface-Elementen bearbeitet werden. In der folgenden Abbildung ist ein Beispiel für eine Erscheinungsform der eXBO-Facility Edit dargestellt:

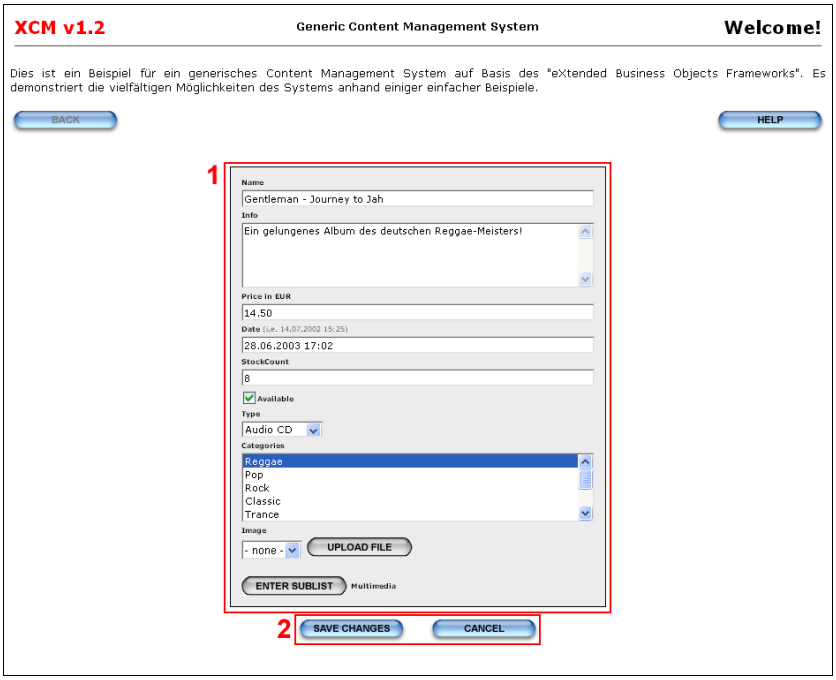

Abb 24: eXBO-Facility Edit

1. In diesem Bereich werden alle zur Bearbeitung vorgesehen Attribute des eXBO-Objektes dargestellt. Je nach Typ des Attributes kommt das entsprechend vorgesehene Interface-Element zum Einsatz. Bestimmte Attribute erlauben zusätzlich auch die Formatierung des Interfaces nach entsprechenden Vorgaben (z.B. das Datumsformat bei einem XAttributeDate), welche von der eXBO-Facility Edit entsprechend berücksichtigt werden.
   In diesem Beispiel sieht man als erstes das Element, welches für ein XAttributeText zum Einsatz kommt. Es ist ein einzeiliges Eingabefeld.
   Das Interface für ein XAttributeTextLong ist als zweites dargestellt. Diesmal wird ein mehrzeiliges Eingabefeld verwendet.
   Das XAttributeFloat wird wieder mit einem einfachen Eingabefeld bearbeitet, wobei diesmal aber die gewünschte Formatierung für den zu bearbeitenden Wert beachtet wird.
   Das Gleiche gilt auch für das XAttributeDate, welches im gewünschten Datumsformat dargestellt und bearbeitet wird.
   Als nächstes folgt ein XAttributeInteger. Auch hier kommt wieder ein einzeiliges Eingabefeld zum Einsatz. Es wäre beispielsweise auch möglich, hier ein Feld mit zwei zusätzlichen Funktionen zum Erhöhen oder Verringern der Zahl anzubieten. Im Beispiel dieser Implementierung wurde sich aber

auf ein einfaches Zahlenfeld beschränkt.
Ein XAttributeCheckBox wird mit einem entsprechenden Häkchen im Interface integriert. Man kann es selektieren oder deselektieren.
Ein XAttributeChoice wird am besten mit einem PopUp-Element bearbeitet. Es ermöglicht die Auswahl eines einzelnen Objektes aus einer Reihe von Vorgaben.
Das XAttributeMultiChoice dagegen wird mit einer Mehrfachauswahl bearbeitet. Auch hier sind die auswählbaren Werte vom Objekt vorgegeben, es ist jedoch auch eine Mehrfachauswahl möglich.
Für das XAttributeUpload wird ein PopUp mit den bereits zum Server übertragenen Dateien dargestellt. Zusätzlich kann man über den daneben befindlichen Button weitere Dateien hochladen, die daraufhin zur Auswahl angeboten werden.
Und das verbleibende Attribut vom Typ XAttributeSubList wird an dieser Stelle mit einem einfachen Button und dessen Bezeichnung dargestellt. Ein Click darauf führt in eine entsprechende eXBO-Facility List, die wiederum die diesem eXBO-Objekt zugehörigen Unter-Objekte des jeweiligen Typs auflistet und deren Bearbeitung ermöglicht.

2. Die beiden Buttons im unteren Teil der Darstellung erlauben das Speichern der Änderungen oder das Abbrechen der Bearbeitung. In beiden Fällen wird zur übergeordneten eXBO-Facility List zurückgekehrt. Die Änderungen am Objekt sind dann direkt und ohne Verzögerung sofort sichtbar.

Die “normale” Implementierung einer Edit-Komponente, wie sie in Abb. 24 dargestellt ist, erfordert schon alleine wegen der Anzahl der zu editierenden Attribute und der verschiedenartigen User-Interface-Elemente einen nicht zu vernachlässigenden Zeit- und Arbeitsaufwand. Das Interessante an der eXBO-Facility Edit ist, dass sie sich ohne jeglichen Programmieraufwand vollautomatisch selbst erzeugt – individuell für jedes Business-Object. Und das ohne vorher wissen zu können, welches Business-Objects als nächstes bearbeitet werden soll, ohne vorher zu wissen, wie viele Attribute dieses Objekt besitzen wird und mit welchen User-Interface-Elementen diese Attribute bearbeitet werden sollen.

Diese Funktionalität wird durch die Verwendung der zuvor beschriebenen eXtended Attributes ermöglicht.

In der folgenden Abbildung, dem UML-Klassendiagramm für die eXBO-Facility Edit, sind die Attribute und Methoden aufgeführt, die mindestens benötigt werden, um die beschriebenen Funktionen der eXBO-Facility Edit erfüllen zu können:

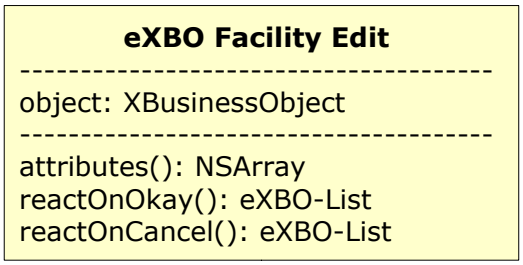

Abb 25: UML-Diagramm eXBO-Facility Edit

Die Variable "object" referenziert das zu bearbeitende eXBO-Objekt. Es wird im Normalfall von der übergeordenten eXBO-Facility List gesetzt und entspricht dort dem aktuell selektieren Objekt.

Mit der Methode "attributes" werden die zur Bearbeitung drazustellenden eXtended Attributes ermittelt. Das eXBO-Objekt gibt sie in der gewünschten Reihenfolge vor.

Die Methoden "reactOnOkay" und "reactOnCancel" werden durch einen Klick auf die entsprechenden Buttons im unteren Teil der Seite ausgeführt. Während bei der ersten die aktuellen Änderungen übernommen und gespeichert werden, verwirft die zweite alle Veränderungen am eXBO-Objekt. In beiden Fällen wird zur übergeordneten eXBO-Facility List zurückgekehrt.

Die nachfolgende Abbildung erläutert die Funktionsweise der dynamischen HTML-Oberfläche der eXBO-Facility Edit:

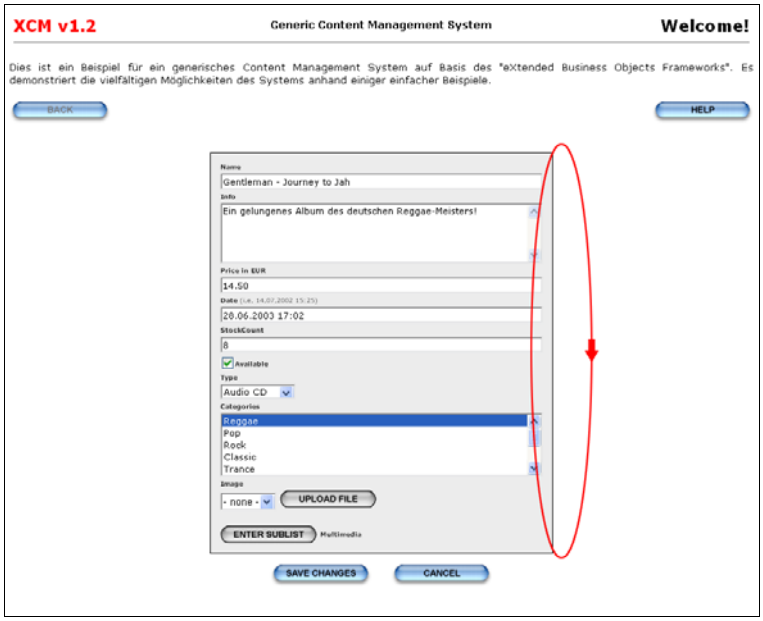

Abb 26: Dynamische HTML-Oberfläche der eXBO-Facility Edit

Im Vergleich zur eXBO-Facility List ist die Generierung dieses Interfaces in ihren Grundzügen weitaus einfacher, da hier nur jeweils ein eXBO-Objekt dargestellt werden muss. Es wird nur noch eine einzelne Schleife benötigt, die über die jeweiligen Attribute iteriert. Die Vielfalt der Darstellung liegt in den unterscheidlichen zum Einsatz kommenden Elemente, die bereits etwas weiter vorne in diesem Kapitel beschrieben wurden. Die eXBO-Facility Edit ermittelt den jeweiligen Typ des Attributes und stellt das vorgesehene Interface-Element dar. Sollte das Element selbst weitere Informationen vom Attribut benötigen (z.B. die in einer List anzubietenden Objekte), dann befragt es dieses direkt und eigenständig.

5.3 eXBO-Facility Search

Die eXBO-Facility Search weist eine HTML-Oberfläche auf, die der eXBO-Edit-Komponente auf dem ersten Blick sehr ähnlich ist. In der nachfolgenden Abbildung ist eine Instanz einer eXBO-Facility Search abgebildet:

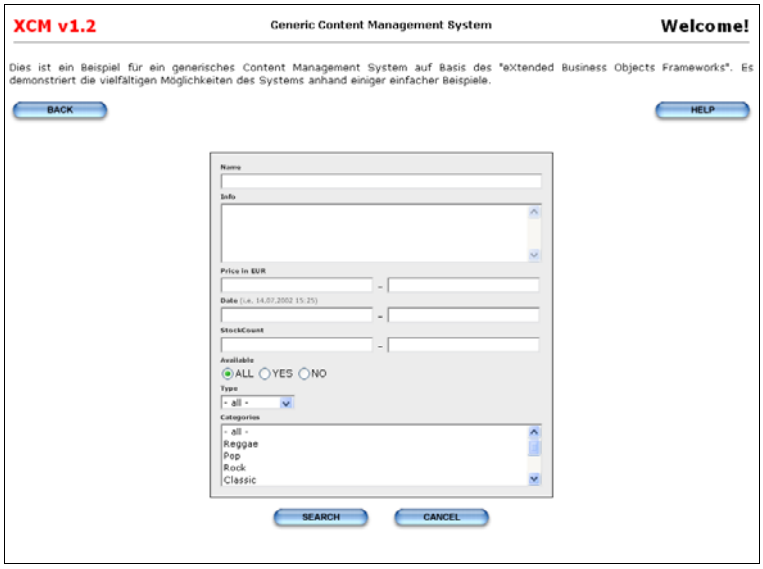

Abb 27: eXBO-Facility Search

Die eXBO-Facility Search weist für jedes Attribut ein Suchfeld auf, das ein gezieltes Suchen bzw. Filtern nach bestimmten Inhalten ermöglicht. Für Attribute, die numerische Grössen oder Zeitpunkte enthalten, werden jeweils zwei Suchfelder eingeblendet, die eine "von-bis"-Suche ermöglichen. Die eXBO-Facility Search kommt meist dann zum Einsatz, wenn der Benutzer in der eXBO-Facility List ein gesuchtes Objekt nicht auf Anhieb findet oder wenn in der dieser nur Objekte mit bestimmten Eigenschaften angezeigt werden sollen. Durch das Anwählen der Funktion "Filtern" bzw. "Suchen" wird das gesamte zuvor von der eXBO-List-Komponente übergebene Array (siehe Abschnitt eXBO-Facility List) nach den entsprechenden Attributen gefiltert. Dabei werden "*" und einige weitere Platzhalter erkannt. Im Falle von mehreren ausgefüllten Suchfeldern werden diese mit einer AND-Funktion verknüpft. Nach dem Abschluss der Suche wird das gefilterte Array in der eXBO-List-Komponente dargestellt. In der folgenden Abbildung wird das UML-Klassendiagramm der eXBO-Search-Komponente vorgestellt:

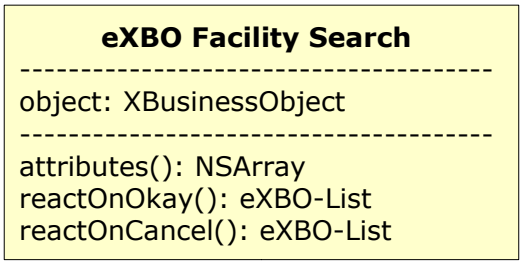

Abb 28: UML-Diagramm eXBO-Facility Search

Die Variable “object” referenziert in diesem Fall das ein beliebiges eXBO-Objekt der zu filternden eXBO-Facility List. Dieses Objekt dient als Referenz für die darzustellenden Filterkriterien.

Die Methode “attributes” ermittelt diese Filterkriterien und ermöglicht der eXBO-Facility, diese entsprechend darzustellen. Je nach Typ kommen für jedes XAttribute wieder unterschiedliche Varianten zum Einsatz. Diese ähneln ihren entsprechenden Gegenstücken aus der eXBO-Facility Edit in den meisten Fällen sehr stark, weshalb sie hier nicht mehr einzeln aufgeführt werden. Attribute, die die zuvor angesprochene “von-bis”-Suche ermöglichen, werden mit den entsprechenden Eingabefeldern dargestellt. Beispiele dafür sind das XAttributeFloat oder das XAttributeDate. Eine weitere Ausnahme stellt das XAttributeCheckBox dar, welches neben “ja” und “nein” auch die Möglichkeit von “alle” bietet. Das gleiche gilt für das XAttributeChoice sowie das XAttributeMultiChoice.

Die beiden Methoden “reactOnOkay” und “reactOnCancel” lösen die notwendigen Aktionen zum Verlassen der eXBO-Facility Search aus und kehren zur übergeordenten eXBO-Facility List zurück. Die erste der beiden Methoden setzt dabei den entsprechend definierten Qualifier bei der eXBO-Facility List, die daraufhin nur die diesem Filter entsprechenden Objekte darstellt.

5.4 Der eXBO-Manager

Um den Wechsel zwischen den verscheidenen eXBO-Facilities zu ermöglichen und die jeweils gemeinsamen Variablen zu verwalten, wurde der sogenannte eXBO-Manager integriert. Dabei handelt es sich um eine kleine einfache Klasse, die die Beziehungen zwischen den eXBO-Facilities organsieriert und eine einfache Navigation zwischen den Komponenten ermöglicht. Das folgende UML-Diagramm beschriebt diese Klasse mit ihren Variablen und Methoden:

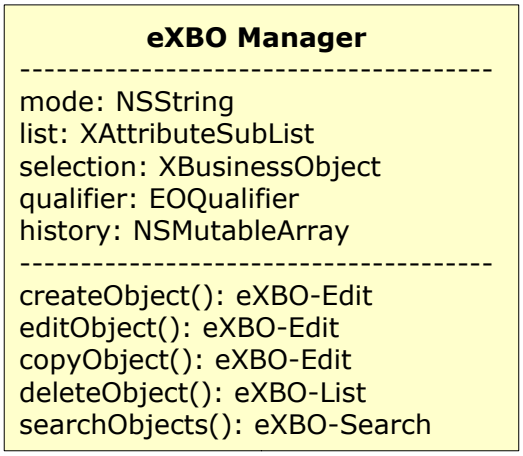

Abb 29: UML-Diagramm eXBO-Manager

In der Variablen "mode" wird der aktuelle Modus dargestellt, der gerade seine Anwendung findet. Dabei kann es sich um "edit", "list" oder auch "delete" handeln. Dadurch wird festgelegt, welche eXBO-Facility gerade dargestellt werden soll.

Die Variablen "list", "selection" und "qualifier" entsprechen ihren Gegenstücken in den jeweiligen eXBO-Facilities und dienen als Quer-Referenz, um diese zwischen den Facilities syncroniseren zu können.

Die Variable "history" vermerkt die Reihenfolge der aufeinanderfolgenden Arbeitsschritte in den eXBO-Facilities. Dadurch wird es möglich, beliebig tief in einen Baum von untergeordneten eXBO-Objekten (unter Verwendung des XAttributeSubList) abzusteigen und anschliessend in der zu erwartenden Reihenfolge der Objekte auch wieder aufzusteigen. Der Benutzer bewegt sich dabei in dem von ihm erwarteten Kontext und landet nicht unerwarteter Weise auf einer anderen eXBO-Facility, als von der er zuvor abgestiegen ist.

Die fünf Methoden des eXBO-Manager sind mit den jeweiligen Buttons verknüpt, die oberhalb der eXBO-Facility List dargestellt werden (vergleiche dazu den Abschnitt "eXBO-Facility List" dieses Kapitels). Sie setzen die jeweiligen Variablen der eXBO-Facilities, initialiseren und stellen diese dann anschliessend dar. Sie ermöglichen also die Navigation zwischen den eXBO-Facilities.

5.5 Zusammenfassung

Wie im vorangegangenen Kapitel 3.2 "Entwurfsmuster" gezeigt, können beim Aufbau von eCommerce-Systemen grundlegende Muster, wie z.B. das Manager-Muster identifiziert werden. Die einzelnen Elemente des Musters (z.B. die List-Page-Komponente oder die Edit-Page-Komponente) bilden dabei die sogenannten "Facility"-Elemente bzw. Komponenten für die Bearbeitung, Darstellung und Verarbeitung von Business-Objekten. In Abbildung 8 ist wiedergegeben, wie ein und das selbe Entwurfsmuster (in diesem Falle das Manager-Muster) für eine Vielzahl von Business-Objekten genutzt wird, sich also ständig wiederholt.

In der Praxis bedeutet dies zunächst, dass für jedes Business-Objekt jede einzelne "Facility" implementiert werden muss. Für ein Objekt "Produkt" eine Produkt-List-Page, eine Produkt-Edit-Page und eine Produkt-Search-Page. Für ein Objekt "Warengruppe" eine Warengruppen-List-Page, eine Warengruppen-Edit-Page und eine Warengruppen-Search-Page usw..

Angelehnt an die Grundsätze der objektorientierten Programmierung wird ein erfahrener Programmierer zunächst versuchen, gemeinsame Funktionalitäten dieser verschiedenen (und doch im wesentlichen gleichen) Facilities zu erkennen und in einer Oberklasse auszulagern. Durch diese Maßnahmen wird die Wartbarkeit erhöht (spätere "Verbesserungen" müssen z.B. nur an einer einzigen Stelle vorgenommen werden und gelten dann sofort für Dutzende von Subklassen) und vor allem der Zeit- und Arbeitsaufwand stark verringert. Trotzdem bleibt ein gewisser Aufwand vorhanden, um die spezialisierten Subklassen zu implementieren und zu instantiieren.

Geht man vom Manager-Muster aus, das lediglich drei Facilities (List, Edit und Search) pro Business-Objekt vorsieht, sind für eine eCommerce-Anwendung die zwanzig verschiedenen Business-Objekte benötigt, insgesamt 60 Facility-Komponenten zu implementieren. Bei einer eCommerce-Anwendung mit 100 verschiedenen Business-Objects würden 300 verschiedene Facility-Komponenten benötigt usw..

Die Idee der eXBO-Facilities ist es, eine einzige universelle Instanz einer Facility (z.B. der List-Page) zu besitzen, die von allen Business-Objects genutzt werden kann. Dadurch verringert sich der Arbeitsaufwand um ein Vielfaches. Dies gilt unabhängig davon, ob es 20, 40 oder 400 Business-Objekte sind. Die Anzahl der benötigten Facilities (z.B. für das Manager-Muster) bleibt konstant.

In der folgenden Abbildung sind die Einsparungspotentiale zur Verdeutlichung noch einmal graphisch aufbereitet. Sie stellt die Anzahl der verschiedenen Business-Objekte in Relation zur Anzahl der benötigten Facilities, bzw. in Relation zum Arbeitsaufwand für die Facilities. Dabei wird versucht, eine Abschätzung des benötigten Arbeitsumfangs bei nicht-objektorientierter Programmierung, bei objektorientierter Programmierung und bei objektorientierter Programmierung und zusätzlichem Einsatz von eXBO-Komponenten zu geben. Im Falle der objektorientierten Programmierung wird die Annahme gemacht, dass ca. 50 Prozent der Programmzeilen in die entsprechenden Oberklassen ausgelagert werden können.

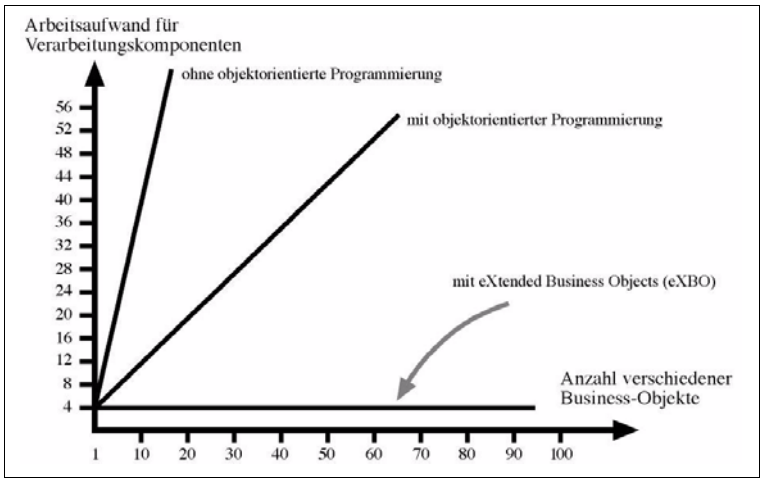

Abb 30: Einsparungspotentiale durch OOP und eXBO-Komponenten

Die Abbildung verdeutlicht am Beispiel der Facilities für das "Manager-Muster" die Einsparungspotentiale, die durch den Einsatz von OO-Programmierung und im zweiten Schritt durch eXBO-Komponenten erzielt werden können. Dies äussert sich in entsprechend stark verkürzten Entwicklungszeiten und der möglichen Senkung der Herstellungskosten.

6 Validierung

6.1 Einleitung

Zur Validierung des Frameworks wurden einige beispielhafte Business-Objekte bzw. eXBO-Objekte implementiert und die Funktion der eXBO-Facilities erprobt.

Für die Beispielanwendung wurden die Objekte Artikel, Multimediadaten und Kategorie gewählt. Diese Objekte eignen sich besonders gut um die Möglichkeiten des Frameworks anschaulich vorzustellen. Gleichzeitig orientieren sie sich an real existierenden Geschäfts-Objekten, die häufig in real existierenden Systemen vorkommen.

Insbesondere das eXBO-Objekt Artikel wurde so konstruiert, dass alle in Kapitel 4 “Spezifikation” eingeführten eXtended Attributes zum Einsatz kommen.

In den nachfolgenden Abschnitten werden die einzelnen Objekte im Detail besprochen. Daran anschliessend wird die aufgebaute Beispielanwendung in Form eines “Rundgangs” vorgestellt.

6.2 eXBO Artikel

Das nachfolgend vorgestellte eXBO-Objekt Artikel erhebt nicht den Anspruch einer vollständigen oder universellen Implementierung einer Artikel-Klasse. Bei der Implementierung des Artikel-Objektes wurde versucht, möglichst typische Attribute zu wählen und gleichzeitig alle vorgestellten eXtended Attributes zu verwenden.

Zu den gewählten Attributen gehört der Name des Artikels, eine ausführlichere Beschreibung, der Preis, das Herstellungsdatum, die Anzahl der noch im Lager befindlichen Artikel, eine allgemeine Typ-Kennzeichnung, eine Liste aller zugehörigen Kategorien und eine Liste der zugehörigen Multimediadaten sowie ein Bild des Artikels.

In dem nachfolgenden UML-Diagramm sind die einzelnen Attribute, deren Bezeichnung und ihre jeweils zugehörigen eXtended Attributes aufgeführt:

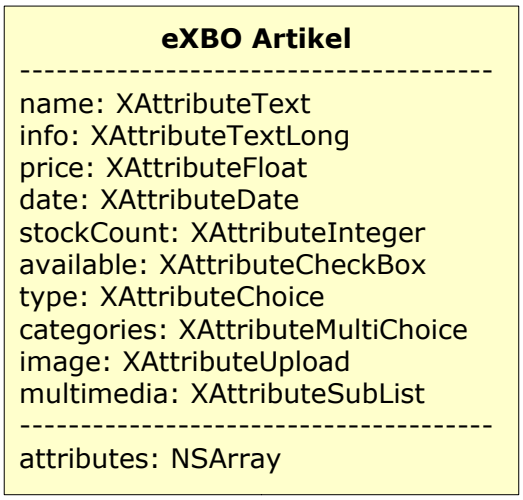

Abb 31: UML-Diagramm eXBO Artikel

In den nächsten beiden Abschnitten werden die Objekte eXBO Multimedia und eXBO Kategorie vorgestellt, die vom eXBO Artikel Objekte in den Attributen "multimedia" und "categories" verwendet werden.

6.3 eXBO Multimedia

Das zuvor vorgestellte eXBO-Objekt Artikel kann mit Hilfe der XAttributeSubList beliebig viele Multimedia-Objekte verwalten. Diese Objekte sind genauso wie der Artikel auch eXBO-Objekte.

Das eXBO-Objekt Multimedia beinhaltet ein Attribut für dessen Namen, ein Info-Attribut für eine Beschreibung von Inhalt und Typ sowie eine Referenz auf die dazugehörige Multimedia-Datei.

In dem nachfolgenden UML-Diagramm sind die einzelnen Attribute, deren Bezeichnung und ihre jeweils zugehörigen eXtended Attributes aufgeführt:

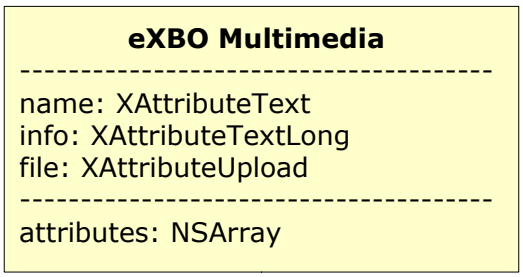

Abb 32: UML-Diagramm eXBO Multimedia

Dieses recht einfach aufgebaute eXBO-Objekt kann in beliebig grosser Zahl vom eXBO-Objekt Artikel referenziert werden.

6.4 eXBO Kategorie

Als drittes Objekt wurde das eXBO-Objekt Kategorie implementiert. Mit diesem Objekt kann ein einfacher Warengruppen-Katalog abgebildet werden. Mit Hilfe des XAttributeMultiChoice-Attributes des Artikels können diesem eine oder mehere Kategorien zugewiesen werden.

Die beispielhafte Implementierung des eXBO-Objekt Kategorie weist die beiden Attribute Name und Info auf. Das folgende UML-Diagramm zeigt den Aufbau des eXBO-Objektes Kategorie:

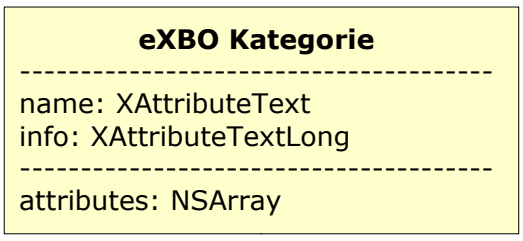

Abb 33: UML-Diagramm eXBO Kategorie

Das Objekt selbst ist recht einfach gehalten, genügt aber den gewünschten Anforderungen.

6.5 Beispielanwendung

Alle bisher vorgestellten Einzelkomponenten, die eXtended Attributes, die eXBO-Facilities und die eXBO-Objekte, wurden im Rahmen der Validierung in einer Beispielanwendung umgesetzt. Mit den Abbildungen auf den folgenden Seiten wird ein Überblick über die Beispielanwendung gegeben.

Die Menü-Page

Nach erfolgreichem Start der Anwendung wird dem Benutzer zunächst die Menü-Page präsentiert:

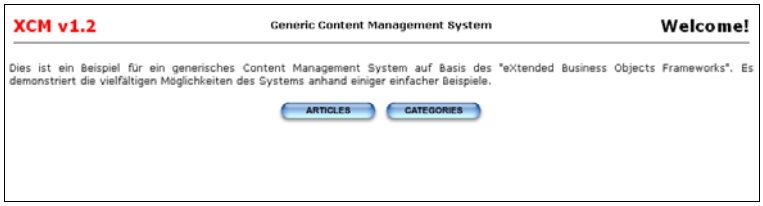

Abb 34: Die Menü-Page

Die Menü-Page bietet im wesentlichten die Möglichkeit auszuwählen, welche der Biespiel-Objekte bearbeitet werden sollen. Durch Click auf den Artikel-Button gelangt man zur Liste aller Artikel. Ein Click auf den Categories-Button ermöglicht dagegen die Bearbeitung der Kategorien.

Die Liste aller Kategorien

Über die Menü-Page gelangt man zur Liste der Kategorien:

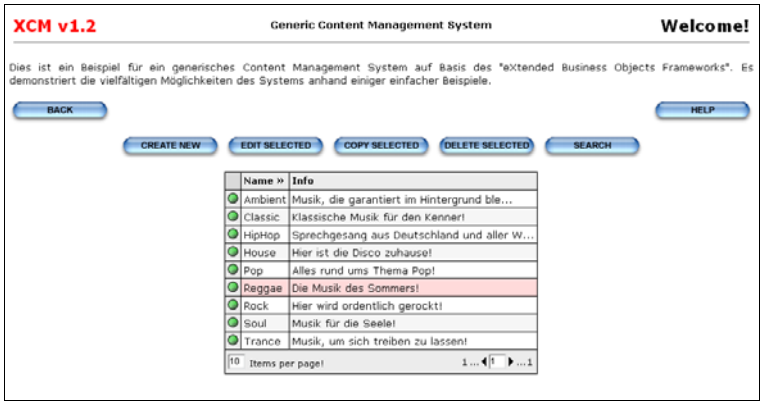

Abb 35: Die Liste aller Kategorien

Die hier verwendete eXBO-Facility List stellt alle vorhandenen Kategorien tabellarisch dar. Durch Anwählen einer Spaltenüberschrift kann die Liste beliebig sortiert werden.

Eine einzelne Kategorie wird durch einen Click auf die in der linken Tabellen-Spalte befindlichen Kugel selektiert. Durch die jeweiligen Menü-Buttons kann eine neue Kategorie erzeugt, die selektierte Kategorie bearbeitet, kopiert oder gelöscht und die Liste aller Kategorien gefiltert werden.

Das Filtern der Kategorien

Durch die Anwahl der Filter-Funktion der eXBO-Facility List gelangt man zur eXBO-Facility Search:

Abb 36: Das Filtern der Kategorien

Mit Hilfe der eXBO-Facility Search kann die Liste aller Kategorien nach den dort angebotenen Eigenschaften gefiltert werden. Durch die Eingabe von Suchkriterien in die jeweiligen Felder wird der zu verwendende Filter definiert. Werden dabei mehrere Felder ausgefüllt so werden diese zu einer UND-Suche verknüpft.

Durch Click auf den Search-Button wird zur Listendarstellung zurückgekehrt und daraufhin die gefilterte Liste der Kategorien darstellt.

Das Bearbeiten einer Kategorie

Mit der eXBO-Facility Edit können die einzelnen Attribute einer Kategorie bearbeitet werden:

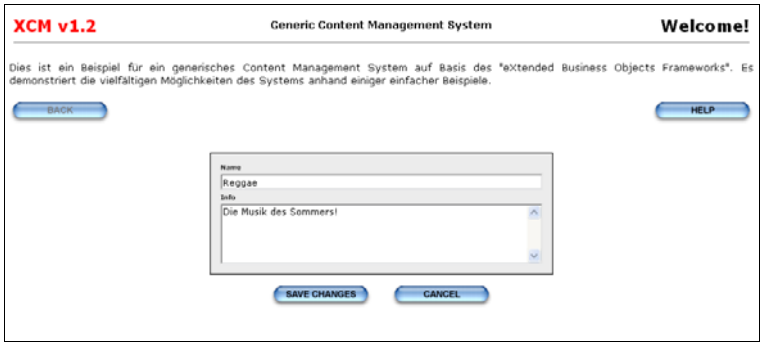

Abb 37: Das Bearbeiten einer Kategorie

In dieser Abbildung sieht man das unterschiedliche Erscheinungsbild der verwendeten eXtended Attribute Text und TextLong. Für das Attribut Name (XAttributeText) wurde ein kurzes Eingabefeld generiert während für das Attribut Info (XAttributeTextLong) ein mehrzeiliges Eingabefeld generiert wurde.

Das Bearbeiten der Kategorie kann mit den entsprechenden Buttons abgeschlossen oder vorzeitig abgebrochen werden.

Die Liste aller Artikel

Über die Menü-Page gelangt man zur Liste der Artikel:

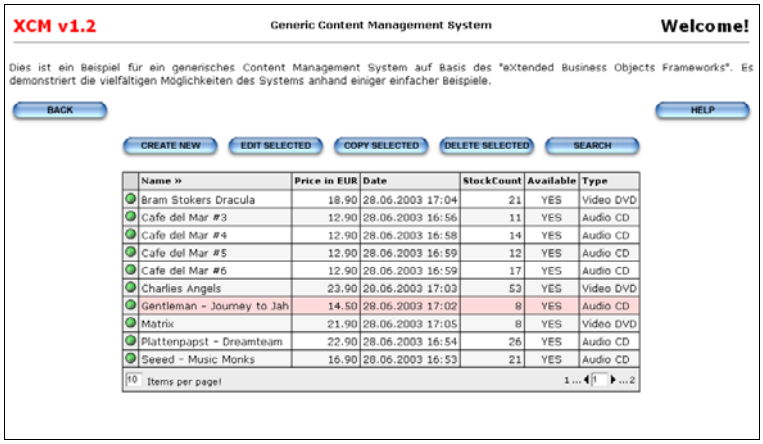

Abb 38: Die Liste aller Artikel

Die hier verwendete eXBO-Facility List stellt alle vorhandenen Artikel tabellarisch dar. Wie schon bei der Liste aller Kategorien kann auch hier durch Anwählen einer beliebigen Spaltenüberschrift die eXBO-Facility List beliebig sortiert werden.

Ein einzelner Artikel wird durch einen Click auf die in der linken Tabellen-Spalte befindliche Kugel selektiert. Durch die jeweiligen Menü-Buttons kann ein neuer Artikel erzeugt, der selektierte Artikel bearbeitet, kopiert oder gelöscht und die Liste aller Artikel gefiltert werden.

Das Filtern der Artikel

Durch die Anwahl der Filter-Funktion der eXBO-Facility List gelangt man zur eXBO-Facility Search:

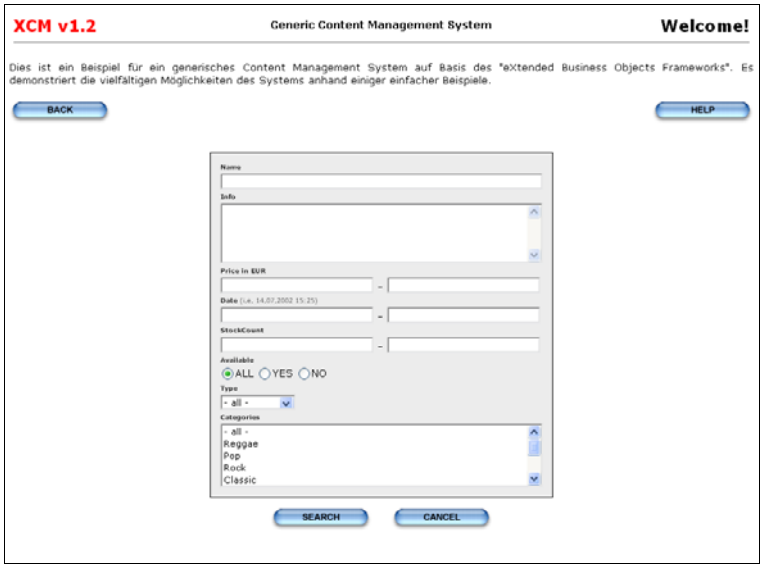

Abb 39: Das Filtern aller Artikel

Mit Hilfe der eXBO-Facility Search kann die Liste aller Artikel nach den dort angebotenen Eigenschaften gefiltert werden. Es gelten die gleichen Filtereigenschaften wie bei den Kategorien, da sie durch die eXBO-Facility Search vorgegeben werden.

Durch Click auf den Search-Button wird zur Listendarstellung zurückgekehrt und daraufhin die gefilterte Liste der Artikel darstellt.

Das Bearbeiten eines Artikels

Mit der eXBO-Facility Edit können die einzelnen Attribute eines Artikels bearbeitet werden.

Durch die bewusst vielfältige Auswahl der Attribute des Artikels können in der folgenden Abbildung fast alle implementierten eXtended Attributes mit ihren zugehörigen Bearbeitungskomponenten in der eXBO-Facility Edit veranschaulicht werden:

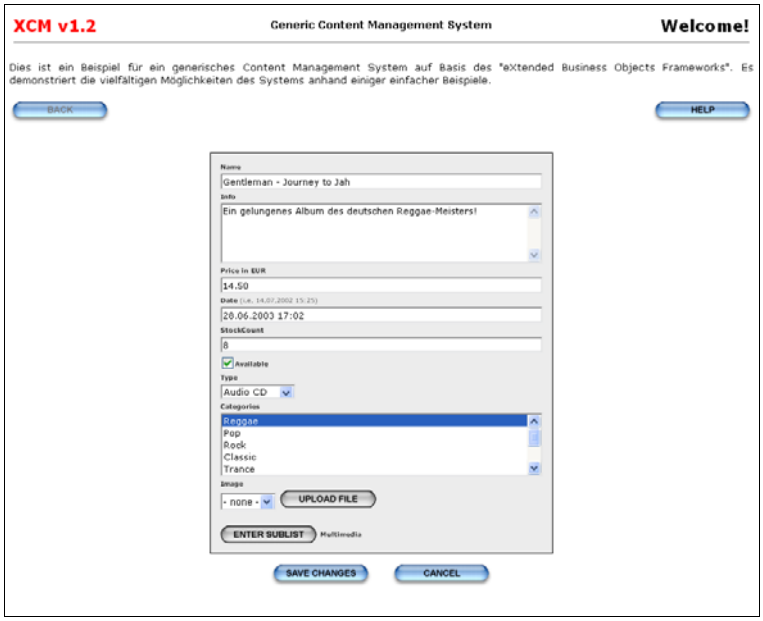

Abb 40: Das Bearbeiten eines Artikels

Für das Attribut Name (XAttributeText) wurde durch die eXBO-Facility Edit ein einzeiliges Eingabefeld generiert. Für das Attribut Info (XAttributeTextLong) wurde ein mehrzeiliges Textfeld erzeugt. Das Attribut Price (XAttributeFloat) wird mit einem speziellen einzeiligen Eingabefeld bearbeitet, welches den Wert des Attributes mit den notwendigen Nachkommastellen darstellt. In ähnlicher Art wird das Attribut Date (XAttributeDate) mit einem die Datums-Formatierung berücksichtigenden Eingabefeld bearbeitet. Das Attribut StockCount (XAttributeInteger) wird von der eXBO-Facility Edit mit einem speziellen einzeiligen Eingabefeld für ganzzahlige Werte dargestellt. Für die boolsche Variable Available (XAttributeCheckBox) wird ein entsprechendes CheckBox-Element generiert. Für das Attribut Type (XAttributeChoice) wird ein Auswahlelement erzeugt, welches es erlaubt, genau eine von mehreren Vorgaben auszuwählen. Für das Attribut Categories (XAttributeMultiChoice) wird ein Auswahlelement erzeugt, welches es erlaubt, eine oder mehrere der vorgegebenen Möglichkeiten zu selektieren. Das Attribut Image (XAttributeUpload) ermöglicht das Übertragen einer Multimedia-Datei von einem Client-Rechner zum Server-Rechner. Die Datei wird auf dem Server dauerhaft gespeichert und steht danach zur Auswahl zur Verfügung. Es können beliebig viele Dateien übertragen werden, von denen jedoch nur jeweils eine aktiv selektiert werden kann. Das letzte Attribut Multimedia (XAttributeSubList) ermöglicht das Verwalten von beliebig vielen Multimedia-Objekten zum gerade bearbeiteten Artikel. In der Edit-Komponente wird für das XAttributeSubList ein Button generiert, der in die Listendarstellung der Multimedia-Objekte verzweigt.

Das Bearbeiten des Artikels kann mit den entsprechenden Buttons abgeschlossen oder vorzeitig abgebrochen werden. Das Bearbeiten der Multimedia-Daten führt ebenfalls zum Sichern der durchgeführten Änderungen.

Die Liste der Multimedia-Daten

Die hier dargestellte Liste der Multimedia-Daten erscheint bei Anwahl der entsprechenden Bearbeitungsfunktion in der eXBO-Facility Edit Komponente des Artikels:

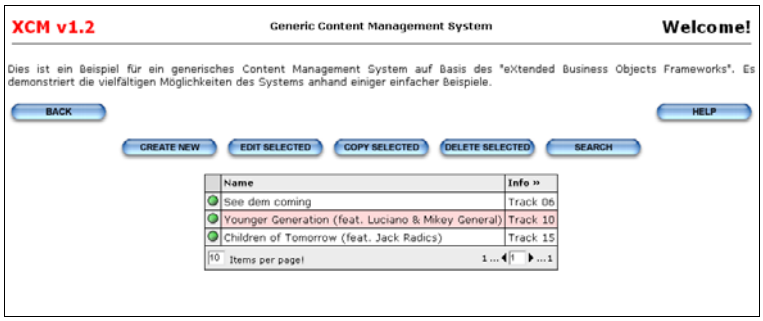

Abb 41: Liste der Multimedia-Daten

Die eXBO-Facility List bietet auch in der Liste der Multimedia-Daten die bereits zuvor geschilderten Funktionen. Die dargestellten Tabellenspalten werden durch das eXBO-Objekt Multimedia vorgegeben.

Das Filtern der Multimedia-Daten

Durch die Anwahl der Filter-Funktion der eXBO-Facility List gelangt man zur eXBO-Facility Search:

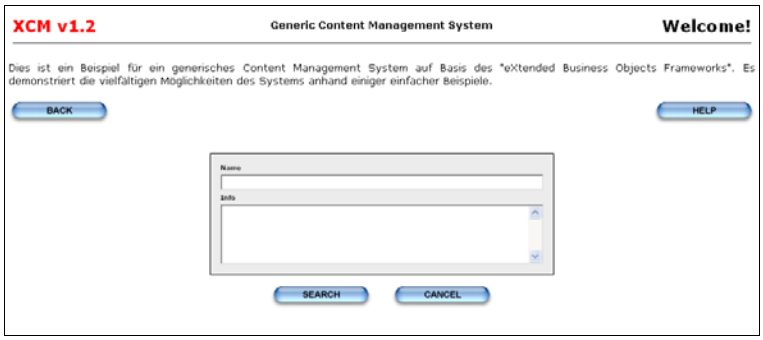

Abb 42: Das Filtern der Multimedia-Daten

Mit Hilfe der eXBO-Facility Search kann die Liste aller Multimedia-Daten nach den dort angebotenen Eigenschaften gefiltert werden. Es gelten die gleichen Filtereigenschaften wie schon bei den Kategorien und Artikeln.

Durch Click auf den Search-Button wird zur Listendarstellung zurückgekehrt und daraufhin die gefilterte Liste der Multimedia-Daten darstellt.

Das Bearbeiten der Multimedia-Daten

Mit der eXBO-Facility Edit können die einzelnen Attribute eines Multimedia-Objektes bearbeitet werden:

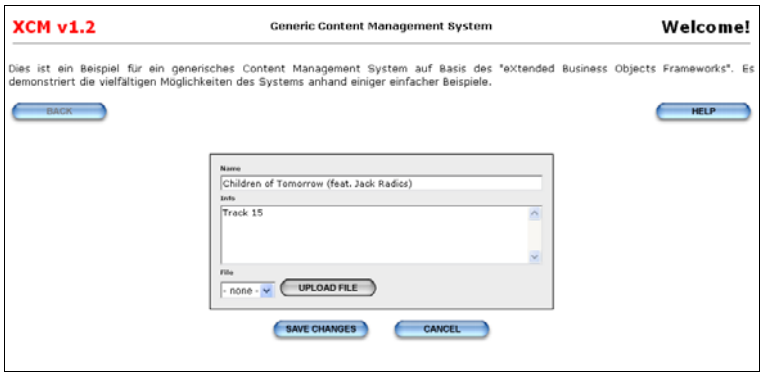

Abb 43: Das Bearbeiten der Multimedia-Daten

Für das Attribut Name (XAttributeText) wurde ein kurzes Eingabefeld generiert während für das Attribut Info (XAttributeTextLong) ein mehrzeiliges Eingabefeld generiert wurde. Zusätzlich wurde für das Attribut File (XAttributeUpload) ein Upload-Element zum Übertragen der eigentlichen Multimedia-Datei generiert.

Das Bearbeiten eines Multimedia-Objektes kann mit den entsprechenden Buttons abgeschlossen oder vorzeitig abgebrochen werden.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde eXtended Attributes, eXBO-Facilities und eXBO-Objekte vorgestellt, die die Basis für das eXBO-Framework für eCommerce-Systeme bilden.

Mit dem Framework sollen Unternehmen zukünftig in die Lage versetzt werden können, deutlich schneller als bisher und deutlich kostengünstiger als bisher eCommerce-Systeme zu erstellen, die elektronische Umsetzungen von Geschäftsmodellen darstellen.

Dies gilt sowohl für neuartige Geschäftsmodelle, bei denen es oft von besonderer Bedeutung ist, sie besonders schnell bzw. als erster vermarkten zu können, als auch für vorhandene Geschäftsmodelle von Unternehmen, die durch elektronische Lösungen optimiert und rationalisiert werden sollen.

Desweiteren wurden in dieser Arbeit Architekturen und Entwurfskonzepte für eCommerce-Systeme vorgestellt, die es ermöglichen ein Maximum an Erweiterbarkeit, Flexibilität und Integrationsfähigkeit in vorhandenen und zukünftigen IT-Strukturen zu gewährleisten.

Die Betrachtung von Geschäftsobjekten, von Objekten also die ihre Entsprechung in der real existierenden Geschäftwelt haben, ermöglicht dabei eine Annäherung zwischen der betriebswirtschaftlich-anwendungsbezogenen Sichtweise von Auftraggebern und Managern und der technisch-implementierungsbezogenen Sicht von Entwicklern. Die nötige enge Zusammenarbeit zwischen Software-Ingenieuren und Experten aus den jeweiligen Geschäftsbereichen, um schnell und korrekt Anforderungen aus der Geschäftswelt in einsatzfähige Systeme umzusetzen, wird durch diese Betrachtungsweise optimal unterstützt.

Die besondere Stärke des eXBO-Frameworks liegt in der Möglichkeit, die Enwicklungszeiten für eCommerce-System drastisch zu verkürzen. Die Verkürzung der Entwicklungszeiten kann dabei als mehrstufiger Prozess angesehen werden:

- Zunächst steigert die konsequente Anwendung von objektorientierter Programmierung die Entwicklungsgeschwindigkeit von Hause aus.
- Durch die Verwendung von fertigen, erprobten, wiederverwendbaren Komponenten verkürzt sich die Fertigungszeit eines eCommerce-Systems ein weiteres mal.
- Der Einsatz von eXBO-Objekten und eXBO-Facilities birgt darüber hinaus das Potential die Entwicklungsgeschwindigkeit noch ein weiteres mal, in bisher unerreichte Grössenordnungen zu steigern.

Die in dieser Arbeit vorgestellten Komponenten bilden lediglich eine Basis des zukünftigen eXBO-Frameworks. Die eXtended Attributes, die eXBO-Facilities sowie die eXBO-Objekte können und sollen durch zukünfitge Arbeiten erweitert, verbessert und ergänzt werden, um ein leistungsfähiges Framework entstehen zu lassen.

8 Abbildungsverzeichnis

9 Literaturverzeichnis

[1] Fox, Oliver; Mendes, Manuel: "e-Business and e-Government. Technological aspects and product evaluation". GMD Fokus Competence for Distributed Object Technology, Plattforms and Services Platin. Berlin, 2000

[2] Morgan Stanley Dean Witter: "The B2B Internet Report. Collaborative Commerce ". Morgan Stanley & Co. Incorporated, USA 2000

[3] Forester Research: "The economy and social impact of electronic commerce". USA, 2000

[4] Korte, Werner; Reinhard, Ulrike (Hrsg.): "Who is who in electronic commerce". Whois Verlags- und Vertriebsgesellschaft, Heidelberg 1998

[5] Gehmeyr, Andreas: "Electonic Commerce - ein †berblick". In: OBJEKTspektrum, H.2, 1999, 56-62

[6] ZDNet News: "CCC warnt: Offene Hinterür in Windows NT. Sichere Verschlüsselung mit Windows in Frage gestellt". http://www.zdnet.de/news/artikel/1999/09/06010-wc.html

[7] NT Security: "Gain local administrator Access on NT", Mai 2000. http//www.ntsecurity.net

[8] NT Security: "ASP Bypass IIS settings", Mai 2000. http//www.ntsecurity.net

[9] NT Security: "IIS may expose ASP code", Mai 2000. http//www.ntsecurity.net

[10] Microsoft: "Microsoft Site Server. Evaluation Guide". USA 1998

[11] Tolksdorf, Robert: "Die Sprache des Web: HTML 3. Informationen aufbereiten und präsentieren im Internet". Dpunkt Verlag, Heidelber,1995

[12] Guther, Apitz; Hoffmann, Glaß: "Wissenschaftliches Arbeiten im World-Wide-Web". TU Berlin, November 1995

[13] OMG: "Business Object DTF Common Business Objects". USA, 1997, Version 1.5

[14] OMG: "Common Business Objects and Business Object Facility". Object Management Group, MA/USA 1996

[15] OMG: "Business Object Domain Task Force RFI ". Object Management Group, MA/USA 1997

[16] IBM: "IBM Common Business Objects. Response to Business Object Task Force RFP". IBM Corporation, MN/USA 1997

[17] IBM; Oracle: "Business Object Facility. Response to Business Object Task Force RFP". IBM Corporation, MN/USA 1997

[18] Data Access; SEMATECH; Prism; IONA: "Business Object Facility. Response to Business Object Task Force RFP". Data Access Technologies, USA 1997

[19] NIIIP: "Task and Session Objects. Response to Business Object Task Force RFP". NIIIP Consortium, USA 1997

[20] Rösch, Martin: "OMG-Standards und ihre Bedeutung für die Praxis". In: Objektspektrum, H.1, 1994, 19-23

[21] Eeles, Peter; Sims, Oliver: "Building Business Objects". John Wiley& Sons Verlag, N.Y/USA 1998

[22] Rösch, Martin: "Sein und Schein von Business-Objekten". In: Objektspektrum, H.6, 1995, 10-14

[23] Rösch, Martin: "Business-Objekte vereinfachen die Struktur von Informationssystemen". In: Objektspektrum, H.4, 1995, 70-75

[24] Fox, Oliver: "IBM's San Francisco Project". GMD Fokus Forschungszentrum Informatiionstechnik GmbH. Berlin, 1998

[25] Apple: "WebObjects Developer's Guide". USA 1998

[26] Apple: "Getting Started with WebObjects". USA 1998

[27] Apple: "Enterprise Objects Framework Developer's Guide". USA 1998

[28] Apple: "Programming WebObjects I". Apple Computer Inc., CA/USA 1998

[29] Apple: "Programming WebObjects II". Apple Computer Inc., CA/USA 1998

[30] Gervae, Nik; Clark, Peter: "Developing Business Applications with Open Step". Springer Verlag, N.Y.1997

[31] Craighill, Nancy: "OpenStep for Enterprises". John Wiley&Sons Verlag, Canada 1997

[32] Lipps, Peter: "Enterprise Objects Framework - Fachspezifische Objekte in OpenStep". In: Objektspektrum, H.5, 1995, 50-59

[33] Braun, Primin: "Business-Software - ein 00-Erfahrungsbericht". In: Objektspektrum,H.6, 1995, 30-35

[34] Booch, Grady: "Entwurfsmuster". In: Objektspektrum, H.1, 1994, 14-18

[35] Sommerlad, Peter: "Entwurfsmuster für Software-Architektur". In: Objektspektrum, H.1, 1991, 16-20

[36] Zimmer,Walter; Neumann, Rainer: "Entwurfskonzepte für die Entwicklung von Frameworks". In: Objektspektrum, H.2, 1996, 22-29

[37] Geppert, Ingo: "ADC- Ein echtes Framework für Benutzungsschnittstellen in VisualWorks". In: Objektspektrum, H.5, 1995, 28-35

[38] Eckstein, Jutta: "Wiew lernt man ein fremdes Framework am besten kennen?". In: Objektspektrum, H.6, 1998, 80-83

[39] Paulisch, Dr. Frances: "Framework - Ein heißes Thema". In: Objektspektrum, H.5, 1995, 6

[40] Weyerhäuser, Markus: "Taligents CommonPoint-Architektur: Frameworks - mehr als nur Objekte". In: Objektspektrum, H.5, 1995, 60-67

[41] Birrer, Andi; Bischofberger, W.R.; Eggenschwiler, Th.: "Wiederverwendung durch Framework Technik - vom Mythos zur Realität". In: Objektspektrum, H.5, 1995, 18-26

[42] Auszug aus White Paper "Leveraging Object-Oriented Frameworks": "Vorteile objektorientierter Frameworks". In: Objektspektrum, H.5, 1995, 10-16

[43] Bäumer, Dirk; Knoll, Rolf; Gryczan, Guido; et al: "Objektorientierte Entwicklung anwendungsspezifischer Rahmenwerke". In: Objektspektrum, H.6, 1995, 48-59

[44] Wegener, Hans: "Für und Wider des Extreme Programming. Extreme Ansichten". In: iX, H.12, 1999

[45] Buschmann, Frank; Meunier, Regine: "Software-Konstruktion mit Entwurfsmustern". In: Objektspektrum, H.5, 1994, 48-57

[46] Tambouris, Efthimios; Fox, Oliver, et al: "An Integrated platform for realising online one-stop government - Proposal from IST Project IST-00-4-1A". Archetypon S.A., Greek 2001

[47] Seidel, Malte; et al: "VeZuDa Leitfaden - Vereinheitlichung bzw. Zusammenführung der verschiedenen Datenstrukturen in der Berliner Verwaltung". Senatsverwaltung für Inneres, Berlin 2000

[48] Fox, Oliver; et al: "Der NeXT Computer - Der Rechner mit dem das World Wide Web erfunden wurde". Multimedia- Anwendung, GMD Fokus Forschungszentrum Informationstechnik GmbH, Musum für Verkehr und Technik, Berlin, 1998

[49] Alber, Thomas: "Servlet Session-Tracking (2)". In: Internet world, November 1999, 84-88

[50] Schmidt, Jürgen: "Dasein oder Nicht-Dasein. Analyse von Ausfallzeiten von Web-Servern". In: C't, H.8, 2000, 174-179

[51] SQLI: "Application Server Study". 1999

[52] Apple: "WebObjects Customers", http://www.apple.com/WebObjects/, 2003

10 Listings

XAttribute.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import <XCollection/XObject.h>

@interface XAttribute : XObject <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    NSString *varName;           // defines the name of the variable which references this object
    NSString *label;             // defines the label which gets displayed in the interface
    NSString *help;              // defines an optional help-text for the attribute
    int       isVisible;         // defines whether the attribute is currently visible or not
    int       isListAttribute;   // defines whether the attribute is visible in the list page
    int       isEditAttribute;   // defines whether the attribute is visible in the edit page
    int       isSearchAttribute; // defines whether the attribute is visible in the search page
}

ciAccessor_h( varName, setVarName )
ciAccessor_h( label, setLabel )
ciAccessor_h( help, setHelp )
intLimitAccessor_h( isVisible, setIsVisible, 0, 1 )
intLimitAccessor_h( isListAttribute, setIsListAttribute, 0, 1 )
intLimitAccessor_h( isEditAttribute, setIsEditAttribute, 0, 1 )
intLimitAccessor_h( isSearchAttribute, setIsSearchAttribute, 0, 1 )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;
- initWithVarName: (NSString *) vn label: (NSString *) l;
- initWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h isVisible: (int) iv
isListAttribute: (int) il isEditAttribute: (int) ie;

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l;
+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h
isVisible: (int) iv isListAttribute: (int) il isEditAttribute: (int) ie;

@end
```

XAttribute.m

```
#import "XAttribute.h"
#import <XCollection/NSString.h>

@implementation XAttribute

ciAccessor( varName, setVarName )
ciAccessor( label, setLabel )
ciAccessor( help, setHelp )
intLimitAccessor( isVisible, setIsVisible, 0, 1 )
intLimitAccessor( isListAttribute, setIsListAttribute, 0, 1 )
intLimitAccessor( isEditAttribute, setIsEditAttribute, 0, 1 )
intLimitAccessor( isSearchAttribute, setIsSearchAttribute, 0, 1 )

- (void) dealloc
{
    if (varName) { [varName release]; }
    if (label) { [label release]; }
    if (help) { [help release]; }
    [super dealloc];
}

- (NSString *) iInfo
{
      return [NSString stringWithFormat:@"%@ with varName = '%@' and label = '%@' and help = '%@' and
isVisible = %i and isListAttribute = %i and isEditAttribute = %i and isSearchAttribute = %i",
        [super iInfo],varName,label,help,isVisible,isListAttribute,isEditAttribute,isSearchAttribute];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttribute setVersion:2];
    [super encodeWithCoder:coder];
    [coder encodeObject:varName];
    [coder encodeObject:label];
    [coder encodeObject:help];
    [coder encodeValueOfObjCType:"i" at:&isVisible];
    [coder encodeValueOfObjCType:"i" at:&isListAttribute];
    [coder encodeValueOfObjCType:"i" at:&isEditAttribute];
    [coder encodeValueOfObjCType:"i" at:&isSearchAttribute];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttribute"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [self setVarName:[coder decodeObject]];
        [self setLabel:[coder decodeObject]];
        [self setHelp:[coder decodeObject]];
        [coder decodeValueOfObjCType:"i" at:&isVisible];
        [coder decodeValueOfObjCType:"i" at:&isListAttribute];
        [coder decodeValueOfObjCType:"i" at:&isEditAttribute];
    }
    if (v == 2) {
        [self setVarName:[coder decodeObject]];
        [self setLabel:[coder decodeObject]];
        [self setHelp:[coder decodeObject]];
        [coder decodeValueOfObjCType:"i" at:&isVisible];
        [coder decodeValueOfObjCType:"i" at:&isListAttribute];
        [coder decodeValueOfObjCType:"i" at:&isEditAttribute];
        [coder decodeValueOfObjCType:"i" at:&isSearchAttribute];
    }
    return self;
}
```

```
- copyWithZone: (NSZone *) zone
{
    XAttribute *copy = [super copyWithZone:zone];
    [copy setVarName:[varName copyWithZone:zone]];
    [copy setLabel:[label copyWithZone:zone]];
    [copy setHelp:[help copyWithZone:zone]];
    [copy setIsVisible:isVisible];
    [copy setIsListAttribute:isListAttribute];
    [copy setIsEditAttribute:isEditAttribute];
    [copy setIsSearchAttribute:isSearchAttribute];
    return copy;
}

- init
{
    self = [super init];
    [self setVarName:nil];
    [self setLabel:@"unnamed"];
    [self setHelp:@"No help available!"];
    [self setIsVisible:1];
    [self setIsListAttribute:1]; // TODO: default could be 0
    [self setIsEditAttribute:1];
    [self setIsSearchAttribute:1];
    return self;
}

- initWithVarName: (NSString *) vn label: (NSString *) l
{
    self = [self init];
    [self setVarName:vn];
    [self setLabel:l];
    return self;
}

- initWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h isVisible: (int) iv
isListAttribute: (int) il isEditAttribute: (int) ie
{
    self = [self init];
    [self setVarName:vn];
    [self setLabel:l];
    [self setHelp:h];
    [self setIsVisible:iv];
    [self setIsListAttribute:il];
    [self setIsEditAttribute:ie];
    return self;
}

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l
{
    return [[[[self class] alloc] initWithVarName:vn label:l] autorelease];
}

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h
isVisible: (int) iv isListAttribute: (int) il isEditAttribute: (int) ie
{
       return [[[[self class] alloc] initWithVarName:vn label:l help:h isVisible:iv isListAttribute:il
isEditAttribute:ie] autorelease];
}

@end
```

XAttributeCheckBox.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeCheckBox : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    int value; // holds the boolean value 1 or 0
}

intLimitAccessor_h( value, setValue, 0, 1 )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;

- (NSString *) stringValue;

@end
```

XAttributeCheckBox.m

```
#import "XAttributeCheckBox.h"

@implementation XAttributeCheckBox

intLimitAccessor( value, setValue, 0, 1 )

- (void) dealloc
{
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = %i",
        [super iInfo],value];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeCheckBox setVersion:1];
    [super encodeWithCoder:coder];
    [coder encodeValueOfObjCType:"i" at:&value];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeCheckBox"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [coder decodeValueOfObjCType:"i" at:&value];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeCheckBox *copy = [super copyWithZone:zone];
    [copy setValue:value];
    return copy;
}

- init
{
    self = [super init];
    [self setValue:0];
    return self;
}

- (NSString *) stringValue
{
    if (value) { return @"YES"; }
    return @"NO";
}

@end
```

XAttributeChoice.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeChoice : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    NSObject *value;     // holds the selected value
    NSArray  *items;     // holds all possible items of the choice
    int       allowNone; // defines whether it is allowed to select none of the items
}

ciAccessor_h( value, setValue )
ciAccessor_h( items, setItems )
intLimitAccessor_h( allowNone, setAllowNone, 0, 1 )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;
- initWithVarName: (NSString *) vn label: (NSString *) l items: (NSArray *) i allowNone: (int) an;

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l items: (NSArray *) i
allowNone: (int) an;

@end
```

XAttributeChoice.m

```
#import "XAttributeChoice.h"
#import <XCollection/NSArray.h>

@implementation XAttributeChoice

ciAccessor( value, setValue )
ciAccessor( items, setItems )
intLimitAccessor( allowNone, setAllowNone, 0, 1 )

- (void) dealloc
{
    if (value) { [value release]; }
    if (items) { [items release]; }
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = %@ and items = %@ and allowNone = %i",
        [super iInfo],value,[items oInfo],allowNone];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeChoice setVersion:1];
    [super encodeWithCoder:coder];
    [coder encodeObject:value];
    [coder encodeObject:items];
    [coder encodeValueOfObjCType:"i" at:&allowNone];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeChoice"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [self setValue:[coder decodeObject]];
        [self setItems:[coder decodeObject]];
        [coder decodeValueOfObjCType:"i" at:&allowNone];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeChoice *copy = [super copyWithZone:zone];
    [copy setValue:value]; // NOTE: copy or not copy
    [copy setItems:items]; // NOTE: copy or not copy
    [copy setAllowNone:allowNone];
    return copy;
}

- init
{
    self = [super init];
    [self setValue:nil];
    [self setItems:nil];
    [self setAllowNone:0];
    return self;
}

- initWithVarName: (NSString *) vn label: (NSString *) l items: (NSArray *) i allowNone: (int) an
{
    self = [super initWithVarName:vn label:l];
```

```
    [self setItems:i];
    [self setAllowNone:an];
    if (!an && ([items count] > 0)) {
        [self setValue:[items objectAtIndex:0]];
    }
    return self;
}

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l items: (NSArray *) i
allowNone: (int) an
{
    return [[[[self class] alloc] initWithVarName:vn label:l items:i allowNone:an] autorelease];
}

@end
```

XAttributeDate.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeDate : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    NSDate   *value;      // holds the value for the attribute
    NSString *dateFormat; // defines the display of the date
}

ciAccessor_h( value, setValue )
ciAccessor_h( dateFormat, setDateFormat )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;

- (NSString *) stringValue;

@end
```

XAttributeDate.m

```
#import "XAttributeDate.h"

@implementation XAttributeDate

ciAccessor( value, setValue )
ciAccessor( dateFormat, setDateFormat )

- (void) dealloc
{
    if (value)      { [value release]; }
    if (dateFormat) { [dateFormat release]; }
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = %@ and dateFormat = %@",
        [super iInfo],value,dateFormat];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeDate setVersion:2];
    [super encodeWithCoder:coder];
    [coder encodeObject:value];
    [coder encodeObject:dateFormat];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeDate"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [self setValue:[coder decodeObject]];
    }
    if (v == 2) {
        [self setValue:[coder decodeObject]];
        [self setDateFormat:[coder decodeObject]];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeDate *copy = [super copyWithZone:zone];
    [copy setValue:[value copyWithZone:zone]];
    [copy setDateFormat:[dateFormat copyWithZone:zone]];
    return copy;
}

- init
{
    self = [super init];
    [self setValue:[NSDate date]];
    [self setDateFormat:@"%d.%m.%Y %H:%M"];
    return self;
}

- (NSString *) stringValue
{
    return [value descriptionWithCalendarFormat:dateFormat timeZone:nil locale:nil];
}
```

```
@end
```

XAttributeFloat.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeFloat : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    float value; // holds the value of the attribute
    int   size;  // defines the size for the interface element
}

floatAccessor_h( value, setValue )
intAccessor_h( size, setSize )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;

@end
```

XAttributeFloat.m

```
#import "XAttributeFloat.h"

@implementation XAttributeFloat

floatAccessor( value, setValue )
intAccessor( size, setSize )

- (void) dealloc
{
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = %f and size = %i",
        [super iInfo],value,size];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeFloat setVersion:1];
    [super encodeWithCoder:coder];
    [coder encodeValueOfObjCType:"f" at:&value];
    [coder encodeValueOfObjCType:"i" at:&size];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeFloat"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [coder decodeValueOfObjCType:"f" at:&value];
        [coder decodeValueOfObjCType:"i" at:&size];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeFloat *copy = [super copyWithZone:zone];
    [copy setValue:value];
    [copy setSize:size];
    return copy;
}

- init
{
    self = [super init];
    [self setValue:0.0];
    [self setSize:10];
    return self;
}

@end
```

XAttributeInteger.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeInteger : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    int value; // holds the value of the attribute
    int size;  // defines the size for the interface element
}

intAccessor_h( value, setValue )
intAccessor_h( size, setSize )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;

@end
```

XAttributeInteger.m

```
#import "XAttributeInteger.h"

@implementation XAttributeInteger

intAccessor( value, setValue )
intAccessor( size, setSize )

- (void) dealloc
{
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = %i and size = %i",
        [super iInfo],value,size];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeInteger setVersion:1];
    [super encodeWithCoder:coder];
    [coder encodeValueOfObjCType:"i" at:&value];
    [coder encodeValueOfObjCType:"i" at:&size];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeInteger"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [coder decodeValueOfObjCType:"i" at:&value];
        [coder decodeValueOfObjCType:"i" at:&size];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeInteger *copy = [super copyWithZone:zone];
    [copy setValue:value];
    [copy setSize:size];
    return copy;
}

- init
{
    self = [super init];
    [self setValue:0];
    [self setSize:10];
    return self;
}

@end
```

XAttributeMultiChoice.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeMultiChoice : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    NSMutableArray *value;     // holds the selected value as an array of objects
    NSArray        *items;     // holds all possible items of the choice
}

ciAccessor_h( value, setValue )
ciAccessor_h( items, setItems )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;
- initWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h isVisible: (int) iv
isListAttribute: (int) il isEditAttribute: (int) ie items: (NSArray *) i;

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h
isVisible: (int) iv isListAttribute: (int) il isEditAttribute: (int) ie items: (NSArray *) i;

- (NSString *) stringForSelection;

@end
```

XAttributeMultiChoice.m

```
#import "XAttributeMultiChoice.h"
#import <XCollection/NSArray.h>

@implementation XAttributeMultiChoice

ciAccessor( value, setValue )
ciAccessor( items, setItems )

- (void) dealloc
{
    if (value) { [value release]; }
    if (items) { [items release]; }
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = %@ and items = %@",
        [super iInfo],[value oInfo],[items oInfo]];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeMultiChoice setVersion:2];
    [super encodeWithCoder:coder];
    [coder encodeObject:value];
    [coder encodeObject:items];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeMultiChoice"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        int dummy;
        [self setValue:[coder decodeObject]];
        [self setItems:[coder decodeObject]];
        [coder decodeValueOfObjCType:"i" at:&dummy];
    }
    if (v == 2) {
        [self setValue:[coder decodeObject]];
        [self setItems:[coder decodeObject]];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeMultiChoice *copy = [super copyWithZone:zone];
    [copy setValue:[value realCopyWithZone:zone]]; // NOTE: deep copy or real copy
    [copy setItems:items];                         // NOTE: copy or not copy
    return copy;
}

- init
{
    self = [super init];
    [self setValue:nil];
    [self setItems:nil];
    return self;
}

- initWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h isVisible: (int) iv
isListAttribute: (int) il isEditAttribute: (int) ie items: (NSArray *) i
```

```
{
    self = [super initWithVarName:vn label:l help:h isVisible:iv isListAttribute:il isEditAttribute:ie];
    [self setItems:i];
    return self;
}

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h
isVisible: (int) iv isListAttribute: (int) il isEditAttribute: (int) ie items: (NSArray *) i
{
       return [[[[self class] alloc] initWithVarName:vn label:l help:h isVisible:iv isListAttribute:il
isEditAttribute:ie items:i] autorelease];
}

- (NSString *) stringForSelection
{
    int i;
    NSMutableString *str = [NSMutableString string];
    for (i=0; i<[value count]; i++) {
        [str appendString:@"\t"];
        [str appendString:[[value objectAtIndex:i] stringValue]];
    }
    [str appendString:@"\t"];
    return str;
}

@end
```

XAttributeSubList.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeSubList : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    NSMutableArray *value;     // holds all objects for that attribute
    NSString       *itemClass; // defines the class off all possible value objects
}

ciAccessor_h( value, setValue )
ciAccessor_h( itemClass, setItemClass )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;
- initWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h isVisible: (int) iv
isListAttribute: (int) il isEditAttribute: (int) ie itemClass: (NSString *) ic;

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h
isVisible: (int) iv isListAttribute: (int) il isEditAttribute: (int) ie itemClass: (NSString *) ic;

@end
```

XAttributeSubList.m

```
#import "XAttributeSubList.h"
#import <XCollection/NSArray.h>

@implementation XAttributeSubList

ciAccessor( value, setValue )
ciAccessor( itemClass, setItemClass )

- (void) dealloc
{
    if (value) { [value release]; }
    if (itemClass) { [itemClass release]; }
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = %@ and itemClass = %@",
        [super iInfo],[value oInfo],itemClass];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeSubList setVersion:1];
    [super encodeWithCoder:coder];
    [coder encodeObject:value];
    [coder encodeObject:itemClass];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeSubList"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [self setValue:[coder decodeObject]];
        [self setItemClass:[coder decodeObject]];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeSubList *copy = [super copyWithZone:zone];
    [copy setValue:[value deepCopyWithZone:zone]];
    [copy setItemClass:[itemClass copyWithZone:zone]];
    return copy;
}

- init
{
    self = [super init];
    [self setValue:[NSMutableArray array]];
    [self setItemClass:@"NSString"];
    return self;
}

- initWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h isVisible: (int) iv
isListAttribute: (int) il isEditAttribute: (int) ie itemClass: (NSString *) ic
{
    self = [super initWithVarName:vn label:l help:h isVisible:iv isListAttribute:il isEditAttribute:ie];
    [self setItemClass:ic];
    return self;
}
```

```
+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h
isVisible: (int) iv isListAttribute: (int) il isEditAttribute: (int) ie itemClass: (NSString *) ic
{
       return [[[[self class] alloc] initWithVarName:vn label:l help:h isVisible:iv isListAttribute:il
isEditAttribute:ie itemClass:ic] autorelease];
}

@end
```

XAttributeText.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeText : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    NSString *value;     // holds the string value of the text
    int       size;      // defines the number of characters for the interface element
}

ciAccessor_h( value, setValue )
intAccessor_h( size, setSize )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;

@end
```

XAttributeText.m

```
#import "XAttributeText.h"

@implementation XAttributeText

ciAccessor( value, setValue )
intAccessor( size, setSize )

- (void) dealloc
{
    if (value) { [value release]; }
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = '%@' and size = %i",
        [super iInfo],value,size];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeText setVersion:1];
    [super encodeWithCoder:coder];
    [coder encodeObject:value];
    [coder encodeValueOfObjCType:"i" at:&size];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeText"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [self setValue:[coder decodeObject]];
        [coder decodeValueOfObjCType:"i" at:&size];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeText *copy = [super copyWithZone:zone];
    [copy setValue:[value copyWithZone:zone]];
    [copy setSize:size];
    return copy;
}

- init
{
    self = [super init];
    [self setValue:@""];
    [self setSize:40];
    return self;
}

@end
```

XAttributeTextLong.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeTextLong : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    NSString *value;     // holds the string value of the text
    int       cols;      // defines the number of cols for the interface element
    int       rows;      // defines the number of rows for the interface element
}

ciAccessor_h( value, setValue )
intAccessor_h( cols, setCols )
intAccessor_h( rows, setRows )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;

- (NSString *) shortStringValue;

@end
```

XAttributeTextLong.m

```
#import "XAttributeTextLong.h"

@implementation XAttributeTextLong

ciAccessor( value, setValue )
intAccessor( cols, setCols )
intAccessor( rows, setRows )

- (void) dealloc
{
    if (value) { [value release]; }
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = '%@' and cols = %i and rows = %i",
        [super iInfo],value,cols,rows];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeTextLong setVersion:1];
    [super encodeWithCoder:coder];
    [coder encodeObject:value];
    [coder encodeValueOfObjCType:"i" at:&cols];
    [coder encodeValueOfObjCType:"i" at:&rows];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeTextLong"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [self setValue:[coder decodeObject]];
        [coder decodeValueOfObjCType:"i" at:&cols];
        [coder decodeValueOfObjCType:"i" at:&rows];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeTextLong *copy = [super copyWithZone:zone];
    [copy setValue:[value copyWithZone:zone]];
    [copy setCols:cols];
    [copy setRows:rows];
    return copy;
}

- init
{
    self = [super init];
    [self setValue:@""];
    [self setCols:40];
    [self setRows:5];
    return self;
}

- (NSString *) shortStringValue
{
    if ([value length] <= 40) {
        return [NSString stringWithString:value];
    }
```

```
    return [NSString stringWithFormat:@"%@...",[value substringToIndex:40]];
}

@end
```

XAttributeUpload.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import "XAttribute.h"

@interface XAttributeUpload : XAttribute <XPAllocation, XPInfo, NSCoding, NSCopying>
{
    NSString *value;     // defines the path to the actual used file
    NSString *folder;    // defines the path to the folder where all uploads are stored
    int       allowNone; // defines whether it is possible to have an empty selection
      NSString *subFolder; // defines a unique folder which is used to separate files from different
attributes
}

ciAccessor_h( value, setValue )
ciAccessor_h( folder, setFolder )
intLimitAccessor_h( allowNone, setAllowNone, 0, 1 )
ciAccessor_h( subFolder, setSubFolder )

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;
- initWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h isVisible: (int) iv
isListAttribute: (int) il isEditAttribute: (int) ie folder: (NSString *) f allowNone: (int) an;

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h
isVisible: (int) iv isListAttribute: (int) il isEditAttribute: (int) ie folder: (NSString *) f allowNone:
(int) an;

- (NSArray *) allFiles;

- (NSString *) fullFolder;

@end
```

XAttributeUpload.m

```
#import "XAttributeUpload.h"
#import <XCollection/NSString.h>
#import <XCollection/NSFileManager.h>

@implementation XAttributeUpload

ciAccessor( value, setValue )
ciAccessor( folder, setFolder )
intLimitAccessor( allowNone, setAllowNone, 0, 1 )
ciAccessor( subFolder, setSubFolder )

- (void) dealloc
{
    if (value) { [value release]; }
    if (folder) { [folder release]; }
    if (subFolder) { [subFolder release]; }
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@ with value = %@ and folder = %@ and allowNone = %i and subFolder
= %@",
        [super iInfo],value,folder,allowNone,subFolder];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XAttributeUpload setVersion:2];
    [super encodeWithCoder:coder];
    [coder encodeObject:value];
    [coder encodeObject:folder];
    [coder encodeValueOfObjCType:"i" at:&allowNone];
    [coder encodeObject:subFolder];
}

- initWithCoder: (NSCoder *) coder
{
    int v = [coder versionForClassName:@"XAttributeUpload"];
    self = [super initWithCoder:coder];
    if (v == 1) {
        [self setValue:[coder decodeObject]];
        [self setFolder:[coder decodeObject]];
        [coder decodeValueOfObjCType:"i" at:&allowNone];
    }
    if (v == 2) {
        [self setValue:[coder decodeObject]];
        [self setFolder:[coder decodeObject]];
        [coder decodeValueOfObjCType:"i" at:&allowNone];
        [self setSubFolder:[coder decodeObject]];
    }
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XAttributeUpload *copy = [super copyWithZone:zone];
    //[copy setValue:[value copyWithZone:zone]]; // NOTE: See subFolder...
    [copy setFolder:[folder copyWithZone:zone]];
    [copy setAllowNone:allowNone];
     //[copy setSubFolder:[subFolder copyWithZone:zone]]; // NOTE: Copy or not copy... should be a unique
id...
    return copy;
}
```

```
- init
{
    self = [super init];
    [self setValue:nil];
    [self setFolder:@"/tmp"];
    [self setAllowNone:1];
    [self setSubFolder:[NSString uniqueString]];
    return self;
}

- initWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h isVisible: (int) iv
isListAttribute: (int) il isEditAttribute: (int) ie folder: (NSString *) f allowNone: (int) an
{
    self = [super initWithVarName:vn label:l help:h isVisible:iv isListAttribute:il isEditAttribute:ie];
    [self setFolder:f];
    [self setAllowNone:an];
    return self;
}

+ (XAttribute *) attributeWithVarName: (NSString *) vn label: (NSString *) l help: (NSString *) h
isVisible: (int) iv isListAttribute: (int) il isEditAttribute: (int) ie folder: (NSString *) f allowNone:
(int) an
{
       return [[[[self class] alloc] initWithVarName:vn label:l help:h isVisible:iv isListAttribute:il
isEditAttribute:ie folder:f allowNone:an] autorelease];
}

- (NSArray *) allFiles
{
                  return     [[NSFileManager     defaultManager]     recursiveDirectoryContentsAtPath:[folder
stringByAppendingPathComponent:subFolder]];
}

- (NSString *) fullFolder
{
    return [folder stringByAppendingPathComponent:subFolder];
}

@end
```

XBusinessObject.h

```
#import <Foundation/Foundation.h>
#import <XCollection/AccessorMacros.h>
#import <XCollection/XObject.h>

@interface XBusinessObject : XObject <XPAllocation, XPInfo, NSCoding, NSCopying>
{

}

// @protocol XPAllocation
- (void) dealloc;
// @end

// @protocol XPInfo
- (NSString *) iInfo;
// @end

// @protocol NSCoding
- (void) encodeWithCoder: (NSCoder *) coder;
- initWithCoder: (NSCoder *) coder;
// @end

// @protocol NSCopying
- copyWithZone: (NSZone *) zone;
// @end

- init;

- (NSArray *) attributes;

- (NSString *) stringValue;

@end
```

XBusinessObject.m

```
#import "XBusinessObject.h"

@implementation XBusinessObject

- (void) dealloc
{
    [super dealloc];
}

- (NSString *) iInfo
{
    return [NSString stringWithFormat:@"%@",[super iInfo]];
}

- (void) encodeWithCoder: (NSCoder *) coder
{
    [XBusinessObject setVersion:1];
    [super encodeWithCoder:coder];
}

- initWithCoder: (NSCoder *) coder
{
    self = [super initWithCoder:coder];
    return self;
}

- copyWithZone: (NSZone *) zone
{
    XBusinessObject *copy = [super copyWithZone:zone];
    return copy;
}

- init
{
    self = [super init];
    return self;
}

- (NSArray *) attributes
{
    return [NSArray array];
}

- (NSString *) stringValue
{
    return @"unnamed";
}

@end
```