

White Paper

Component based Development bei OpenKnowledge

Christian Wachsmann, OpenKnowledge GmbH

Version 1.1, August 2002

Inhalt

1. Einleitung.....	3
2. Component based Development.....	4
2.1. Definition Framework.....	4
2.2. Definition Komponente	5
2.3. Komponententechnologien.....	6
2.4. Component based Development	7
3. OpenKnowledge Enterprise Components.....	9
3.1. CbD bei OpenKnowledge.....	9
3.2. Übersicht OpenKnowledge Enterprise Components	10
3.3. Eigenschaften der OpenKnowledge Components	13
3.4. Einsatzgebiete der Komponenten.....	14

1. Einleitung

Geschäftsziel von OpenKnowledge ist die IT Beratung und das Erstellen von individuellen Enterprise Softwaresystemen. Die Erfahrungen aus den in der Vergangenheit abgeschlossenen und heute noch laufenden Projekte zeigen, dass auch bei sehr individuellen Softwaresystemen ein mehr oder weniger großer Teil der Anforderungen ähnlich sind. Zum Beispiel benötigt fast jedes webbasierte System mit Kundenbindung eine Benutzerverwaltung: Benutzer müssen sich am System registrieren und anmelden können usw. Warum also muss zu Beginn eines Projektes die hundertste Benutzerverwaltung gebaut werden?

Die im Laufe der Zeit angesammelten Projekterfahrungen sind eingeflossen in eine Sammlung von Komponenten, die die grundlegenden Aspekte eines Client-Server Systems im Bereich des Enterprise Computings abdecken. Die Komponenten wurden von OpenKnowledge in den letzten Monaten entwickelt und ständig kommen aus laufenden Projekten neue Ideen für wiederverwendbare Bausteine hinzu. Dieses Dokument stellt die entwickelten Komponenten vor, und zeigt, wie zukünftige Projekte vom Einsatz fertiger Bausteine, der Erfahrung und dem Wissen der OpenKnowledge Berater und Entwickler profitieren können.

In Kapitel 2 sollen zunächst einige Grundlagen vermittelt werden. Dies ist notwendig, da insbesondere der Begriff „Komponente“ in zahlreichen Definitionen existiert und jede ein wenig anders ist. Komponenten und Component based Development (CbD) sind immer noch Gegenstand aktueller Forschungen der Informatik und unterliegen stetigen Änderungen und Weiterentwicklungen.

Kapitel 3 geht ins Detail: Es wird Einblick gegeben in die Herangehensweise, mit der bei OpenKnowledge Component based Development betrieben wird und welche Ergebnisse daraus resultieren, mit denen zukünftige Projekte umgesetzt werden können.

Im letzten Kapitel 4 werden einige Szenarien vorgestellt, mit welcher Auswahl an Komponenten welche Systeme umgesetzt werden können. Gemäß der Kernkompetenz von OpenKnowledge liegen diese Beispiele im Bereich des Enterprise Computings.

Ziel dieses White Papers ist es, einen Überblick der Komponentenlandschaft von OpenKnowledge zu vermitteln. Der Leser sollte über grundlegende Kenntnisse der Software Entwicklung verfügen. Technische Details werden in anderen Dokumenten erläutert, diese sind auf Nachfrage erhältlich (info@openknowledge.de).

2. Component based Development

2.1. Definition Framework

In der Literatur findet sich eine große Anzahl an Definitionen für den Begriff Framework. Mehr oder weniger meinen sie alle das gleiche und drücken es mehr oder weniger akademisch aus. Am besten in den Kontext dieses White Papers passen die folgenden Definitionen:

Ein Framework ist eine Menge von kooperierenden Klassen, die einen wiederverwendbaren Entwurf für einen bestimmten Anwendungskontext vorgeben. Das Framework bestimmt dabei die Architektur der Applikation. [1]

Im unscheinbaren zweiten Satz verbirgt sich dabei eine weitreichende Formulierung: Besonders im Bereich des Enterprise Computing ist eine leistungsfähige Software Architektur entscheidend für den Projekterfolg und den operativen Betrieb. Da die Architektur durch das Framework bestimmt wird, lastet eine entsprechende Verantwortung auf dem Framework.

Eine weitere Definition, die uns in Richtung der Darstellung des Zusammenhangs mit Komponenten bringt, ist die folgende:

Ein Framework bestimmt die Softwarearchitektur der Anwendung: Es definiert die Struktur „im Großen“, seine Unterteilung in Klassen und Objekte, die jeweiligen zentralen Zuständigkeiten, die Zusammenarbeit der Klassen sowie den Kontrollfluß. [2]

Mit einem Framework alleine kann man nun weiter nichts anfangen, es ist eine leere Hülle. Erst durch das Hinzufügen von Business Logik wird das Framework ergänzt zu einer funktionierenden Applikation. Dazu gibt das Framework selbst die Regeln vor: Durch Schnittstellen, abstrakte Klassen und nicht zuletzt Dokumentation wird dem Entwickler vorgegeben, wie das Framework zu einer speziellen Applikation erweitert werden kann.

Weitergedacht bedeuten diese Regeln aber auch, dass das Framework den gesamten Entwicklungsprozess beeinflusst. Durch vorgegebene Interfaces und Design Patterns besitzt der Applikationsentwickler nicht mehr alle Freiheiten. Dies ist aber auch gut so, denn bei guten Frameworks bekommt er wohl durchdachte und bewährte Grundlagen, so dass der Programmierer sich voll auf die Entwicklung der Applikationslogik konzentrieren kann und nicht alle Grundlagen neu erfinden muss („Reinvent the Wheel“).

In der zweiten Definition wird beschrieben, dass das Framework den Kontrollfluss besitzt. Das bedeutet, dass ausschließlich das Framework Komponenten aufruft, niemals rufen Komponenten das Framework auf. (Dies wird von manchen Autoren auch als Hollywood Prinzip bezeichnet: „don't call us, we call you“). Es ist also Zeit zu definieren, was genau eine Komponente ist.

2.2. Definition Komponente

Auch bei Komponenten kann man eine Vielzahl von Definitionen finden. Wir übernehmen die folgende:

„Eine Komponente ist ein Stück Software, das klein genug ist, um es in einem Stück erzeugen und pflegen zu können, groß genug ist, um eine sinnvoll einsetzbare Funktionalität zu bieten und eine individuelle Unterstützung zu rechtfertigen sowie mit standardisierten Schnittstellen ausgestattet ist, um mit anderen Komponenten zusammenzuarbeiten.“ [3]

Diese Definition lässt sich genauer aufschlüsseln, indem die folgenden grundlegenden Eigenschaften von Komponenten erläutert werden (größtenteils aus [4] übernommen):

- ▶ **Schnittstellen-definiert** – Eine Komponente muss eine wohldefinierte Schnittstelle besitzen, über die sie ihre Funktionalität nach außen anbietet. Die Schnittstelle definiert, was die Komponente leistet, nicht wie sie es leistet.
- ▶ **Ersetzbar** – Komponenten müssen durch eine neue Version oder durch eine andere Komponente ersetzbar sein, ohne dass dies Auswirkungen auf den Rest des Systems hat. Anders ausgedrückt: Die Implementierung der Komponenten ist transparent und austauschbar, nur die Schnittstelle und die dahinterliegenden funktionalen Anforderungen müssen gleich bleiben.
- ▶ **Identifizierbar** – Der Funktionsumfang der Komponente muss klar abgrenzbar und identifizierbar sein. Dies schlägt sich in der Praxis u.a. in der Benennung der Komponenten wider: Wenn man Schwierigkeiten hat, die Funktionalität der Komponente in einen kurzen, aussagekräftigen Namen zu verpacken, ist die Komponente vermutlich nicht klar abgegrenzt. Ist eine Komponente nicht klar identifizierbar, gibt es beim Einsatz der Komponente vermutlich auch Schwierigkeiten in Bezug auf Kohärenz und Kopplung.
- ▶ **Abgeschlossen** – Die Verwendbarkeit einer Komponente darf nicht von anderen Komponenten abhängig sein. Ist sie es doch, so kann man durch Zusammenfassung der abhängigen Komponenten eine neue Komponente bilden. Dies sollte allerdings vermieden werden, da es die Wiederverwendbarkeit stark einschränkt. Von ihrem Funktionsumfang kleinere Komponenten können eher in verschiedenen Kontexten eingesetzt werden, als eine funktionsüberladene Komponente.
- ▶ **Unabhängig** – Eine wünschenswerte Eigenschaft einer Komponente ist es, dass sie nicht an eine bestimmte Zielanwendung gebunden ist. Dies würde dem Prinzip der Wiederverwendbarkeit widersprechen. Bei sehr speziellen Anforderungen ist es allerdings trotzdem sinnvoll, eine spezielle – abhängige – Komponente zu entwerfen. Auch wenn sie damit nicht wieder-

verwendbar ist, ist trotzdem der Aspekt der Ersetzbarkeit wichtig für die Wartbarkeit des Gesamtsystems.

- ▶ Selbstbeschreibend – Neben der syntaktischen Schnittstelle, also dem eigentlichen Interface in der Programmiersprache, sollte eine Komponente über eine semantische Schnittstelle verfügen. Im allgemeinen bedeutet dies, dass eine ausführliche Dokumentation die Funktionen der Komponente erläutert. Wünschenswert sind formelle Methoden, die das Zusammenschalten von Komponenten vereinfachen. Diese sind aber noch Gegenstand aktueller Forschungen.
- ▶ Generisch – Eine weitere wünschenswerte Eigenschaft ist der generische Einsatz einer Komponente. Über Parametrisierung oder deklarative Programmierung sollte die Komponente individuell auf einen Anwendungskontext angepasst werden können. Dabei dürfen natürlich die Binärdaten der Komponente selbst nicht angepasst werden. Diese Eigenschaft erhöht die Wiederverwendbarkeit enorm, da die Komponente ihren statischen Charakter verliert.

2.3. Komponententechnologien

Unabhängig von den Eigenschaften der Komponenten, von den Komponententypen und der Analyse und dem Design von komponentenbasierter Software ist die Technologie, mit der die Komponenten schließlich realisiert werden. Momentan sind am Markt die folgenden Technologien vorherrschend:

- ▶ *.NET*. Die Microsoft Komponententechnologie wurde im Laufe der Zeit öfters umbenannt. Aus OLE, COM, DCOM und der Internet Zwischenstation ActiveX wurde schließlich die *.NET* Technologie. Diese Technologie ist vorwiegend für die Microsoft Welt geeignet. Details siehe [5].
- ▶ *CORBA Component Model*. Die Komponententechnologie der CORBA Spezifikation ist nicht zu verwechseln mit der Architektur einer verteilten Middleware, mit der das Akronym CORBA normalerweise assoziiert wird. Die CORBA Component Model Spezifikation beschreibt eine Architektur zum Erstellen von skalierbaren, sprach-unabhängigen, transaktionssicheren Enterprise Applikationen. Damit ähnelt es der EJB Spezifikation. Details siehe [6].
- ▶ *Enterprise Java Beans*. Die EJB Spezifikation ist Bestandteil der Java 2 Enterprise Edition und wird in Kapitel 3 genauer beschrieben. Die vollständige Referenz befindet sich unter [7].

In der Literatur finden sich folgende Anforderungen an Komponententechnologien [4]. Diese Anforderungen sollen kurz auf die oben aufgeführten Technologien untersucht werden. Eine ausführliche Untersuchung würde den Rahmen dieses White Papers sprengen, so dass hier auf andere Untersuchungen verwiesen wird [8].

- ▶ Sprach- und Plattformunabhängigkeit. Die Kommunikation von Komponenten sollte sprach- und plattformübergreifend funktionieren. Am deutlichsten ist dies beim CORBA Component Model ausgeprägt, die Microsoft Technologie .NET ist dagegen auf die Microsoft Welt zugeschnitten. Die Kommunikation mit Komponenten anderer Technologien erfordert in diesem Fall dritte Technologien, z.B. SOAP. EJB ist hingegen die einzige Technologie, die nur mit der Java Programmiersprache arbeitet. Bei .NET und CORBA und können mehrere bzw. beliebige Sprachen eingesetzt werden.
- ▶ Dynamische Wiederverwendbarkeit der Dienste. Im Idealfall sollten Komponenten mit einer Art Plug & Play Technik sofort einsatzbereit sein. In der Praxis ist in allen Komponententechnologien ein nicht zu unterschätzender Deployment Aufwand notwendig.
- ▶ Ortstransparenz. Für den Anwender einer Komponente soll verborgen bleiben, ob die Komponente auf einem lokalen oder einem entfernten Rechner läuft. Diese Forderung macht die eigentliche Komplexität der oben aufgeführten Technologien aus, denn transparente Verteilung ist mit hohem Aufwand verbunden. Alle obigen Technologien erfüllen diese Anforderung.
- ▶ Selbstbeschreibung. Im Gegensatz zu den unter 2.2 aufgeführten Eigenschaften einer Komponente ist hier gemeint, dass Komponenten zur Laufzeit Auskunft über ihre Funktionalität geben können. Metadaten lassen sich bei allen drei Technologien abfragen, allerdings nur die syntaktischen Informationen. Aufgrund fehlender formeller Methoden können Metadaten über die Semantik der Komponente nicht zur Laufzeit abgefragt werden. Dies verhindert auch eine echte dynamische Integration von Komponenten zur Laufzeit.
- ▶ Trennung zwischen Interface und Implementierung. Diese nach Meinung des Autors wichtigste Eigenschaft einer Komponente ist bei allen oben aufgeführten Technologien vorhanden.

2.4. Component based Development

Component based Development (CbD) ist nach Meinung einiger Autoren als neues Paradigma zu sehen, das den Ansprüchen einer industriellen Fertigung von Software Bausteinen näher kommt. Obwohl Komponenten meist in einer objektorientierten Programmiersprache umgesetzt werden, ist CbD keinesfalls gleichzusetzen mit objektorientierter Entwicklung. CbD ist vielmehr der abstraktere Blick auf ein Softwaresystem.

Doch Component based Development ist noch mehr: Es beinhaltet den Entwicklungs- und Wartungsprozess während der gesamten Lebensdauer einer Komponente. Während der Entwurfsphase eines großen Softwaresystems bedeutet CbD vor allem das Denken in Komponenten: Das Suchen nach in

sich abgeschlossenen Bausteinen, die wiederverwendet werden können. Eine reine objekt-orientierte Herangehensweise würde eventuell zu einem monolithischen System führen: Eine aggregierte und komponierte Sammlung von Objekten mit geringer Kohäsion und hoher Kopplung, ohne größere Strukturierung. CbD hilft, große Softwaresysteme auf einer hohen Abstraktionsebene zu strukturieren.

CbD bedeutet dagegen zunächst nicht, dass gezwungenermaßen fertige, am Markt erhältliche, Komponenten eingesetzt werden. Es ist eher ein angenehmer Nebeneffekt, dass während des Entwurfs solche Aspekte beachtet werden. Sobald ein wirklicher Markt für industriell gefertigte Komponenten praxisreif ist, werden sich solche Aspekte in Softwareprojekten noch weiter durchsetzen.

Ein anderer Blickwinkel auf CbD soll den Zusammenhang zwischen Framework und Komponenten verdeutlichen. Wie in Abschnitt 2.1 erläutert, gibt das Framework die Architektur und den Kontrollfluß vor. Die Zielapplikation, basierend auf dem Framework, existiert am Anfang des Entwicklungsprozesses als Gerüst mit weißen Stellen. Genau an diese freien Stellen werden Komponenten eingehängt. Nach dem Hollywood Paradigma ruft das den Kontrollfluß innehabende Framework die Komponenten auf, um die funktionalen Anforderungen an das Gesamtsystem zu erfüllen.

Soweit die Theorie. In der Praxis besteht die eigentliche Aufgabe der Entwickler darin, das Framework und die Komponenten zu einem Ganzen zu verbinden. Je nach Granularität der Komponenten ist dies mit mehr oder weniger Aufwand verbunden. Von einem echten Plug & Play von Framework und Komponenten ist die Softwareindustrie jedoch noch weit entfernt. Dies ist jedoch keineswegs nur negativ einzuschätzen: Gerade in der Kombination von Fertigbausteinen und individuellen Verbindungsteilen lässt sich eine individuelle und spezialisierte Software entwickeln, die genau auf die Erfordernisse der Endanwender abgestimmt ist.

Damit werden die Vorteile von Komponenten (fertig, getestet) und Individualsoftware (speziell, angepasst) verbunden und ergeben in ihrer Synergie den gewünschten Mehrwert für den Auftraggeber und Endanwender. Bei Einführung eines neuen Softwaresystems stellt sich nicht mehr die Frage „make or buy?“, sondern die Antwort sollte lauten: „make *and* buy!“.

3. OpenKnowledge Enterprise Components

3.1. CbD bei OpenKnowledge

Die eher theoretischen Abhandlungen aus Kapitel 2 sollen hier nun konkretisiert werden. Was bedeutet Component based Development bei OpenKnowledge, und welche Vorteile ergeben sich hieraus für den Kunden?

Die Kernkompetenz von OpenKnowledge liegt in der Entwicklung von individuellen Enterprise Systemen. Enterprise Computing bedeutet in diesem Zusammenhang, verschiedenste Softwaresysteme zu integrieren und als ein großes System nach außen wirken zu lassen. Neben der Zusammenführung heterogener Systeme spielt vor allem der Aspekt der physikalischen Verteilung und der Skalierung eine große Rolle. Oftmals werden diese integrierten Systeme über ein Web Frontend angesprochen.

Zur Umsetzung dieser Anforderungen hat OpenKnowledge sich für die Java Technologie entschieden. Java bietet für alle Problemfelder im Enterprise Computing entsprechende Lösungen. Insbesondere die Enterprise Java Beans Komponententechnologie erfüllt alle in Kapitel 2 aufgestellten Anforderungen und ermöglicht die Entwicklung komponentenbasierter Software für den Geschäftsbereich von OpenKnowledge.

Component based Development bedeutet für OpenKnowledge die Entwicklung *von* und *mit* Komponenten. Die Entwicklung *von* Komponenten bedeutet Lösungen für Standardprobleme zu schaffen. Entwicklung *mit* Komponenten bedeutet, im Rahmen eines Projektes Individual-Software auf Basis eigener oder auch dritter Komponenten zu erstellen.

Im Schnelldurchlauf lässt sich ein typischer Projektablauf auf Basis von Component based Development in folgende Schritte aufteilen (der eigentliche Entwicklungsprozess ist hiervon unabhängig und wird in [10] detailliert beschrieben).

1. Anforderungsdefinition: Grundlage ist wie bei jedem Software Entwicklungsprozess eine zumindest grobe Anforderungsdefinition.
2. Fertigkomponenten auswählen: Das Entwicklungsteam analysiert die Anforderungsdefinition und ermittelt daraus die notwendigen Komponenten
3. Customizing der Komponenten: Die Komponenten werden auf die speziellen Anforderungen angepasst.
4. Business Logik erstellen: Das Framework und die Komponenten werden verbunden, indem die Kommunikation zwischen den Komponenten entwickelt wird.

Dieses Vorgehen entspricht der „make and buy“ Philosophie wie in Abschnitt 2.4 beschrieben.

Welche Vorteile erhält man durch Cbd?

- ▶ **Schnellerer Projektstart.** Durch den Einsatz eines Frameworks und darin integrierten fertigen Bauteilen erhält man in kürzester Zeit bereits eine lauffähige Version. Das Projektteam kann sich dadurch voll auf die funktionalen Anforderungen der Applikation konzentrieren.
- ▶ **Minimierung von technischen Risiken.** Viele Projekte scheitern heutzutage, weil das Entwicklungsteam die technischen Risiken unterschätzt hat. Die schnelle Weiterentwicklung und der Trieb, die modernsten Technologien einzusetzen, kann gerade bei mangelnder Erfahrung auf dem Gebiet der New Technologies zu erheblichen Projektverzögerungen, wenn nicht gar zum kompletten Scheitern des Projektes führen. Die Erfahrung von OpenKnowledge mit neuesten Software Technologien und der Einsatz von modernen, aber bereits bewährten Komponenten, minimieren das Risiko von technischen Problemen.
- ▶ **Höhere Software Qualität.** Das Problem der 1.0 Software ist allgemein bekannt: Die ersten Versionen einer neuen Software beinhalten Fehler, die sich im allgemeinen auf die Gebrauchstauglichkeit der Software auswirken. Mehrfach eingesetzte und intensiv getestete Komponenten sind von diesen Kinderkrankheiten befreit.
- ▶ **Kürzere Entwicklungszeiten (Time-to-Market).** Durch Verwendung von fertigen Bausteinen, Verringerung des technischen Risikos und gesteigerter Qualität ergibt sich zwangsläufig eine schnellere Entwicklung.
- ▶ **Investitionsschutz.** Durch die Ersetzbarkeit von Komponenten ist der Auftraggeber nicht zwangsläufig an ein integriertes Produkt bzw. eine integrierte Komponente gebunden. Komponenten, die nicht mehr den Ansprüchen genügen, können ausgetauscht werden, ohne dass Änderungen sich durch das gesamte System ziehen. Generische Komponenten lassen sich weiterhin schneller an ein verändertes Geschäftsmodell anpassen als statische Komponenten.
- ▶ **Integration von Drittsystemen.** Die Komponententechnologie setzt sich auch bei etablierten und weit verbreiteten Standardsystemen immer mehr durch. Besonders im Bereich der CRM- und ERP-Systeme (z.B. Siebel, SAP) können funktionale Teilbereiche als Komponenten gekauft werden. Diese können natürlich leichter integriert werden als ein monolithischer Block.

3.2. Übersicht OpenKnowledge Enterprise Components

Die Erfahrungen von OpenKnowledge in vielen breitgefächerten Enterprise Projekten sind in die Entwicklung der OpenKnowledge Enterprise Components eingeflossen. Die Komponenten können in Projekten eingesetzt werden, um die oben beschriebenen Vorteile für Projekte auszunutzen.

Alle Komponenten zielen auf den Bereich Enterprise Computing ab und sind für den serverseitigen Einsatz konzipiert. Die Komponenten sind branchenunabhängig und bieten Standardlösungen für immer wiederkehrende Teilprobleme im Enterprise Computing. Die Komponenten enthalten keine Businesslogik und sind für sich alleine genommen nicht sinnvoll lauffähig.

Als „Lebensraum“ der Komponenten dient das OpenKnowledge Enterprise Integration Framework, welches für die Kommunikation der Komponenten untereinander und nach außen verantwortlich ist. Die Anpassung der Komponenten zu einer Individuallösung kann von OpenKnowledge Spezialisten oder von Dritten erfolgen.

Die Komponenten sind zusammengefasst zu Softwarepaketen, die im folgenden kurz beschrieben werden. Details zu den einzelnen Paketen werden in gesonderten Dokumenten erläutert.

Enterprise Integration Framework

Kernstück der OpenKnowledge Komponenten ist das Enterprise Integration Framework (Arbeitsname *Beckström*). Der Anwendungskontext des Frameworks (siehe Definition in Abschnitt 2.1) ist das serverseitige Enterprise Computing.

Das Framework definiert eine moderne 4-Schicht Architektur und basiert auf dem J2EE Blueprint von Sun Microsystems [9]. Damit ist das Framework keine proprietäre Software, sondern fasst die bewährtesten Softwaretechnologien in einem praxistauglichen Paket zusammen. Als Whitebox Framework ist es dazu gedacht, die Grundlage für individualisierte Anwendungen zu bilden. Komponenten für Standardprobleme werden integriert und Business Logik zur Individualisierung wird hinzugefügt, basierend auf den Regeln, die durch das Framework vorgegeben werden.

Das Framework macht intensiven Gebrauch von offenen und bewährten Standards. Hier nur einige Schlagwörter:

- ▶ J2EE. Für den Enterprise Einsatz empfiehlt sich der Gebrauch der EJB Technologie, obwohl das Framework prinzipiell auch ohne Application Server läuft. Durch die EJB Technologie werden Aspekte wie Skalierbarkeit, Transaktionssicherheit und Stabilität per se unterstützt.
- ▶ JSP und XSLT. Als Ausgabetechniken werden insbesondere die Java Server Pages sowie XSL Transformationen unterstützt. Das Framework trennt explizit die eigentlichen Daten von deren Darstellung und ermöglicht damit SSMP (Single Source Multiple Platforms).
- ▶ JMX und SNMP. Im Enterprise Bereich von besonderer Bedeutung ist das Management der Applikation. Hierunter wird die Überwachung und Steuerung des Systems in einem Rechenzentrum verstanden, um die Hochverfügbarkeit der Applikation sicherzustellen. Zur Realisierung werden die Java Management Extensions und das SNMP Protokoll eingesetzt, Details siehe in einem separaten White Paper [11].

- ▶ SOAP. Zur Kommunikation insbesondere mit .NET Technologie ist das Framework in der Lage, über das SOAP Protokoll mit externen Systemen zu kommunizieren.

Das Framework bietet insgesamt wesentlich mehr, als nur das Zusammenhalten der Komponenten. Durch die Unterstützung verschiedenster Protokolle, Komponenten- und Visualisierungstechnologien ist das Framework als unternehmensweite Kommunikationszentrale geeignet. Details über das Framework siehe [12].

Enterprise Service Components

Die Enterprise Service Components sind eine Sammlung von sogenannten Services, die applikationsunabhängige Dienste zur Verfügung stellen. Diese können ohne weiteren Aufwand direkt in das Framework integriert werden, sie können aber auch gemäß der Eigenschaften von Komponenten separat in anderen Umgebungen eingesetzt werden. Folgende Service Komponenten sind derzeit verfügbar:

- ▶ Authentication. Dieser Service bietet die Funktionalität für Authentifizierungsprozesse. Durch eine Plug-In Technologie können verschiedenste Mechanismen unterstützt werden. Der einfachste und meistgenutzte Mechanismus ist die Login/Passwort Authentifizierung.
- ▶ Authorization: Der Service bietet die Funktionalität für ein generisches Rechtekonzept und ermöglicht damit Anfragen der Form: Hat eine bestimmte Ressource ein bestimmtes Recht?
- ▶ User Management. Dieser Service ermöglicht die Speicherung und Abruf von Benutzerstammdaten. Durch die Mischung von speziellen und generischen Mechanismen ist der Service für häufig verwendete Daten sehr schnell, ermöglicht über entsprechendes Customizing aber auch das Speichern beliebiger Benutzerdaten.
- ▶ Tracking. Der Tracking Service bietet eine asynchrone Protokollierung von Systemereignissen. Diese Ereignisse dienen normalerweise zum Erfassen von Nutzerverhalten oder als Grundlage für ein Datawarehouse.
- ▶ Unified Messaging (UMS). Der Service bietet die Möglichkeit zum Versand von E-Mail, Fax und SMS/MMS. Die Komponente ist für sich alleine nicht lauffähig, sie benötigt zusätzlich Mail-Server, Fax bzw. SMS/MMS Gateways.

Alle Services bis auf den UMS Dienst erfordern eine relationale Datenbank und einen J2EE kompatiblen Application Server, falls die EJB Vorteile wie Skalierbarkeit und Transaktionssicherheit benötigt werden. In Planung für die nähere Zukunft sind die Service Komponenten Shopping Cart, Web Miles und E-Payment.

Content Syndication Server

Der Content Syndication Server ist ein System für Datenkonvertierungs-, Migrations- und Syndicationprozesse mit folgenden Anwendungsschwerpunkten:

- ▶ *Datenkonvertierung.* Unter Datenkonvertierung wird die Umwandlung von Daten von einem Format in ein anderes Format verstanden.
- ▶ *Import/Export Prozesse.* Import/Export Prozesse gehen über die reine Konvertierung hinaus, da eine Reihe von Konvertierungsprozessen nacheinander ablaufen und somit komplexere Prozesse abbildbar sind.
- ▶ *Syndication Prozesse.* Content Syndication stellt die höchste Stufe der Konvertierung dar. Die Import/Export Prozesse werden in eine Laufzeitumgebung eingebunden und werden vollautomatisiert von einem Scheduler gesteuert. Zusätzlich werden alle Systemaktionen protokolliert und können in Datenbanken gespeichert und ausgewertet werden.

Die Zielgruppe für den Einsatz dieses Servers sind Unternehmen, die wiederkehrend eine Vielzahl von unterschiedlichen Datenformaten verarbeiten müssen. Besondere Eigenschaft des Syndication Servers ist die Konvertierung von relationalen zu hierarchischen Daten und umgekehrt. Details zum Syndication Server siehe im gesonderten White Paper [13].

3.3. Eigenschaften der OpenKnowledge Components

Die besonderen Eigenschaften der OpenKnowledge Components sollen in diesem Abschnitt beschrieben werden. Der Unique Selling Point der Komponenten besteht in der konsequenten Ausrichtung auf Enterprise Computing, welches aus den folgenden Aufführungen hervorgeht.

- ▶ *Reusable.* Haupteigenschaft ist die Wiederverwendung der Komponenten. Ihre volle Stärke erzielen die Komponenten im Zusammenspiel, sie sind aber auch separat einsetzbar. Alle Komponenten sind auf den Enterprise Bereich ausgerichtet, sind dabei aber branchen- und anforderungsunabhängig.
- ▶ *Customizable.* Die hohe Wiederverwendbarkeit wird insbesondere durch den generischen Ansatz der Komponenten ermöglicht. Über deklarative Programmierung oder Parametrisierung können die Komponenten an die individuellen Bedürfnisse angepasst werden. Die Anpassung erfolgt, wie bei Komponenten üblich, ohne Änderungen im Source Code.
- ▶ *Expandable.* Der Einsatz von offenen Standards, die offene Architektur und die dokumentierten Schnittstellen erlauben die Erweiterung bestehender Komponenten. Beispielsweise kön-

nen weitere Services in der Beckström Komponente für spezielle Kundenanforderungen hinzugefügt werden.

- ▶ *Scalable*. Alle Komponenten sind für den Einsatz in J2EE Application Servern konzipiert. Durch Verwendung der EJB Technologie werden insbesondere Eigenschaften wie Skalierbarkeit und Transaktionssicherheit übernommen. Die Komponenten sind explizit für Clusterbetrieb ausgelegt.
- ▶ *Portable*. Alle Komponenten sind in Java J2EE Technologie entwickelt. Sie sind damit prinzipiell plattformunabhängig und laufen auf jedem System, für das ein J2EE Application Server zur Verfügung steht. Es werden keine proprietären Eigenschaften von Application Servern oder Datenbanken verwendet. In Ausnahmefällen wird dieser Gebrauch gekapselt.
- ▶ *Managable*. Alle Komponenten können über standardisierte Verwaltungsschnittstellen überwacht und gesteuert werden. Über existierende Managementsysteme (IBM Tivoli, HP OpenView, CA Unicenter) können die Komponenten in den Überwachungsbetrieb eines Rechenzentrums integriert werden. In alle Komponenten integriert ist eine Messpunktanalyse für Performance Messungen und Optimierungen.

3.4. Einsatzgebiete der Komponenten

Als Anwendungsgebiet des Framework wurde bislang der Begriff Enterprise Computing genannt. In diesem Abschnitt sollen einige reale Anwendungsszenarien dargestellt werden, in denen das Framework und die Komponenten zum Einsatz kommen können. Success Stories der OpenKnowledge Projekte sind auf Anfrage erhältlich.

Szenario 1: Web Applikation

Eine Web Applikation ist die klassische Anwendungsdomäne der vorgestellten Komponenten. Ein kleine Applikation mit wenig, einfacher oder Standard-Funktionalität lässt eventuell mit einem Content Management System genauso gut lösen. Bei sehr speziellen Anforderungen, die mit einer gemeinhin als Portal bezeichneten Anwendung nicht mehr viel zu tun haben, muss jedoch eine Individuallösung eingesetzt werden.

Die dabei zum Einsatz kommenden Technologien sind typischerweise JSPs, die HTML für den Browser oder WML für mobile Geräte ausgeben. Die dafür notwendige Infrastruktur ist durch das Framework vorgegeben. Die anwendungsunabhängige Business Logik der Applikation hingegen kann durch den Einsatz der oben aufgeführten Service Komponenten abgedeckt werden. Die applikationsabhängigen Teile werden selbst implementiert, so dass eine auf die speziellen Anforderungen zugeschnittene Web Applikation entsteht.

Szenario 2: Web Services

Web Services kann man vereinfacht darstellen als Komponenten, die über das Internet ihre Dienste anbieten. Web Services sind damit ein idealer Kandidat für den Einsatz der OpenKnowledge Komponenten: Die in das Framework integrierten Komponenten und integrierten Drittsysteme sind über die SOAP Fähigkeiten des Frameworks als Web-Service erreichbar.

Szenario 3: Content Syndication

Ein immer wiederkehrendes Problem beim Betrieb von Softwaresystemen, die ständig aktuelle Informationen oder Content benötigen, stellt die Beschaffung und Integration des Contents dar. Der in Abschnitt 3.2 erläuterte Syndication Server kann als Ergänzung zum Enterprise Framework verwendet werden, um vollautomatisch und regelbasiert Inhalte in Empfang zu nehmen, zu transformieren und zur Verfügung zu stellen. Über den gleichen abstrakten Mechanismus lassen sich auch Inhalte exportieren.

Szenario 4: Legacy Integration

Die große Herausforderung beim Enterprise Computing besteht in der Zusammenführung von sogenannten Legacy Systemen. Dies sind meist ältere, monolithisch aufgebaute Applikationen ohne frei zugängliche Schnittstellen, die allerdings in ihrer Funktionalität im allgemeinen unersetzbar sind.

Mit einer Integration mehrerer dieser Systeme oder mit dem Vorschalten eines Inter-/Intranet basierten Frontends verspricht sich der Betreiber einen Mehrwert. Das Enterprise Integration Framework stellt für genau diese Problematik eine Lösung bereit, indem die Integration von Legacy Systemen in der Architektur per se vorgesehen ist. Die Legacy Systeme werden darin gekapselt und können mit eigener Business Logik erweitert werden, um den gewünschten Mehrwert zu schaffen.

Literatur

- [1] Richard Helm, Erich Gamma et al: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995
- [2] H.-J. Applerath: *Implementierung von Informationssystemen*, Universität Oldenburg, 1998
- [3] Frank Griffel: *Componentware*, dpunkt Verlag, 1998
- [4] David Sprott, Lawrence Wilkes: *Using Componentised Software*, CBDi Forum, 1999
- [5] Microsoft: *.NET Homepage*, <http://www.microsoft.com/net>, 03.06.2002
- [6] Object Management Group: *CORBA Homepage*, <http://www.corba.org>, 03.06.2002
- [7] Sun Microsystems: *Java 2 Platform Enterprise Edition Homepage*, <http://java.sun.com/j2ee>, 03.06.2002
- [8] Google Web Directory: *Component Frameworks – Comparison and Review*, http://directory.google.com/Top/Computers/Programming/Component_Frameworks/Comparison_and_Review, 02.06.2002
- [9] Nicholas Kasseem, et al: *Designing Enterprise Applications with the Java 2 Plattform (Enterprise Edition)*, Sun Press, 2000
- [10] Frank Müller: *OpenKnowledge Agile Process*, OpenKnowledge GmbH, 2002
- [11] Carsten Diekmann, Christian Wachsmann: *Management Technologie des Beckström Integrationframeworks*, OpenKnowledge GmbH, 2002
- [12] Christian Wachsmann: *Enterprise Integration Framework*, OpenKnowledge GmbH, 2002
- [13] Christian Wachsmann: *Content Syndication Server*, OpenKnowledge GmbH, 2002.

Copyright

Dieses Dokument ist urheberrechtlich geschützt. Sämtliche Rechte vorbehalten. Jegliche Vervielfältigung des Inhalts, gleich welcher Art und auf welchem Medium, zu gewerblichen Zwecken ist nicht gestattet.

Haftungsausschluss und Rechte Dritter

Die Informationen im vorliegenden Dokument werden vorbehaltlich etwaig bestehender Patente, Marken, Gebrauchsmuster oder sonstiger absoluter Schutzrechte veröffentlicht.

Vorliegende Texte und Abbildungen wurden nach bestem Wissen und Gewissen erstellt. Trotz der höchstmöglichen Sorgfalt können jedoch Fehler nicht vollständig ausgeschlossen werden. Für derartige Fehler und deren Folgen wird keine Haftung oder sonstige juristische Verantwortlichkeit übernommen.