Die Spezifikation – kritischer Erfolgsfaktor der Komponentenorientierung

Sven Overhage[+]

[+] *Technische Universität Darmstadt, Fachgebiet Wirtschaftsinformatik I – Entwicklung von Anwendungssystemen, Institut für Betriebswirtschaftslehre, Hochschulstraße 1, D-64289 Darmstadt, Tel.: +49 (6151) 16-6688, Fax: -4301, E-Mail: overhage@bwl.tu-darmstadt.de, URL: http://www.winf1.bwl.tu-darmstadt.de*

Zusammenfassung. Die Schaffung einer konzeptionellen Terminologie, die von konkreten Technologien der Implementierung abstrahiert, und eines einheitlichen Spezifikationsrahmens zur Dokumentation der Außensicht von Komponenten sind wesentliche Voraussetzungen für den Erfolg der komponentenorientierten Anwendungsentwicklung. In diesem Beitrag wird vor allem die Bedeutung eines einheitlichen Spezifikationsrahmens betont, der als Basis für Vorgehensmodelle, Werkzeuge und die Entstehung von Komponentenmärkten dient und somit als kritischer Erfolgsfaktor zu betrachten ist. Dieser Spezifikationsrahmen basiert auf einer konzeptionellen Terminologie, die sich auf die am Markt vorhandenen Technologien für die Implementierung abbilden lässt. Sein Nutzen wird an verschiedenen Anwendungsbeispielen und einem Marktplatz für den Handel mit Software-Komponenten, der im Rahmen eines Dissertationsprojektes an der TU Darmstadt entwickelt wurde, veranschaulicht.

Schlüsselworte: Komponente; Framework; Spezifikation; Vorgehensmodell; Komponentenmarkt; Komponenten-Repository

1 Einleitung

Die Komponentenorientierung bringt für die Wirtschaftsinformatik und die betriebliche Anwendungsentwicklung eine Reihe von Vorteilen: Durch die Implementierung vieler einzelner Komponenten, die sich jeweils auf die Erfüllung einer zentralen Funktionalität konzentrieren, ist eine bessere Wiederverwendung von Software möglich. Dies führt im Allgemeinen zu einer Verbesserung der Produktivität (Entwicklung konzentriert sich auf Neues) sowie der Qualität (Komponenten sind bereits getestet) und somit zu einer Senkung der Kosten in der Anwendungsentwicklung.

Das Vorhandensein einer Vielzahl von Softwarebausteinen (Komponenten) ermöglicht außerdem (im Idealfall) die individuelle Konfiguration von Anwendungen nach den Vorgaben des Anwenders. So lässt sich beispielsweise eine Anwendung der Finanzbuchführung flexibel den gesetzlichen Erfordernissen anpassen, indem die Komponente „Bilanzierung“ ausgetauscht wird (und so Bilanzen nach verschiedenen Verfahren erstellt werden können: Handelsgesetzbuch – HGB, US Generally Accepted Accounting Principles – US-GAAP, International Accounting Standard – IAS etc.). Neben diesen Vorteilen für das Customizing ergeben sich durch die feinere Granularität von Anwendungen auch Verbesserungen für deren Wartung durch den Entwickler, die sich zu einer Wartung der von einer Änderung betroffenen

Komponenten vereinfacht. Schließlich ist das Vorhandensein eines ausgereiften Konzepts von Bauteilen (Komponenten) immer auch ein (akademisches) Kriterium für die Bewertung der Reife einer Ingenieursdisziplin.

Trotz dieser Vorteile hat sich die Umsetzung der komponentenorientierten Anwendungsentwicklung in der Praxis als schwierig erwiesen. So lassen sich zwar für das Design von grafischen Benutzungsoberflächen und einfache Dienste wie beispielsweise das Umrechnen zwischen verschiedenen Währungen häufig Komponenten finden. Komplexere Dienste mit domänenspezifischer Funktionalität (z.B. Komponenten zur Bilanzierung, Gewinn- und Verlustrechnung etc.) sind bislang jedoch selten als Bausteine implementiert worden, sondern meist Bestandteil komplexer monolithischer Anwendungen.

Eine Ursache für diese Schwierigkeiten der Komponentenorientierung in der Praxis ist der Mangel an technischen Standards, der die Konfigurierung von Anwendungen aus Softwarekomponenten immer dann erschwert, wenn diese in verschiedenen Technologien realisiert wurden. Hier zeichnet sich vor allem durch die Entwicklung XML basierter Web-Services (vgl. [Bett2001]) ein deutlicher Fortschritt ab. Die Anwendung dieser Technologie ermöglicht es dem Entwickler, aus beinahe jedem Stück (Legacy-) Software eine Komponente zu machen, die ihre Dienste für den Aufruf durch andere Komponenten an einer (technisch) standardisierten Schnittstelle zur Verfügung stellt.

Darüber hinaus fehlen bislang vor allem fachliche Standards in den (betrieblichen) Anwendungsbereichen (Domänen), was die Zusammenarbeit von Komponenten verschiedener Entwickler bzw. Hersteller erschwert (vgl. [Ortn1999a]). So ist eine Komponente für die Bilanzierung des einen Herstellers beispielsweise nur dann sinnvoll an die Anwendung zur Finanzbuchhaltung eines anderen anschließbar, wenn beide unter dem Begriff „Bilanzierung“ das Gleiche verstehen. Beim Austausch von Komponenten ist daher darauf zu achten, dass im Rahmen der Dokumentation die Erfassung von Fachtermini, die bei der Entwicklung einer Komponente implementiert wurden, mitsamt den zugehörigen Definitionen explizit ermöglicht wird. Auf diese Weise wird die Entstehung fachlicher Standards begünstigt und der Entwickler einer Anwendung in die Lage versetzt, Komponenten auch in Bezug auf ihre fachliche Eignung zu begutachten.

Ein weiterer Schlüsselfaktor für die Probleme in der Praxis ist das Fehlen einer (konzeptionellen) Komponenten-Terminologie mit einem einheitlichen Spezifikationsrahmen, ohne den kein gemeinsames Verständnis für die Beschreibung von Komponenten herzustellen ist. Ein solcher Spezifikationsrahmen bildet nicht nur die Grundlage für die Entwicklung von Werkzeugen (CASE Tools) und Vorgehensmodellen. Er ist vor allem die Basis für den Austausch von Komponenten zwischen Projekten (in einem unternehmensinternen Komponentenmarkt) oder gar Unternehmen (in einem offenen Markt) und damit die Entstehung eines Komponentenmarktes. Er ist somit von zentraler Bedeutung für den Erfolg der komponentenorientierten Anwendungsentwicklung (vgl. [Grif1998]).

In diesem Beitrag wird ein einheitlicher Spezifikationsrahmen für Komponenten vorgestellt, der auf einem Standardisierungsprozess aufsetzt, an dem Universitäten und Industrie gleichermaßen beteiligt waren (vgl. [Turo2002]). Er besitzt dadurch die Chance, in der komponentenorientierten (betrieblichen) Anwendungsentwicklung eine herausragende Geltung zu erlangen und von Werkzeugen auf breiter Ebene unterstützt zu werden.

Auf Basis dieses Spezifikationsrahmens werden anschließend ein einfaches Vorgehensmodell der komponentenorientierten Anwendungsentwicklung und mögliche Werkzeuge für dessen

Unterstützung beschrieben. Der Beitrag endet mit der Vorstellung eines elektronischen Marktplatzes für den Handel mit Software-Komponenten, der am Fachgebiet Wirtschaftsinformatik I der TU Darmstadt entwickelt wurde. Dieser Marktplatz implementiert den vorgeschlagenen Spezifikationsrahmen und integriert sich somit nahtlos in den Entwicklungsprozess, wie er hier beschrieben wird.

2 Eine konzeptionelle Komponenten-Terminologie

Im Zentrum der komponentenorientierten Anwendungsentwicklung steht der Begriff der „Komponente“ (Software-Komponente, Baustein), der in der Literatur vielfach (und nicht immer übereinstimmend) definiert wurde.

Einige Autoren (vgl. [Orfa1996]) definieren sie als objektorientiertes Konstrukt und verstehen Komponenten als spezielle Objekte, die neben Kapselung, Vererbung und Polymorphismus vor allem die Eigenschaft der Unabhängigkeit (von einem Programm, einer Programmiersprache bzw. einer Implementierung) besitzen. Für die Praxis ist diese Definition einer Komponente gleich aus mehreren Gründen zu eng gefasst. Zum einen erzwingt sie bei der Entwicklung von Komponenten den Einsatz objektorientierter Technologien (Vererbung, Polymorphismus) und lässt Komponenten, die auf nicht objektorientierten Technologien basieren (z.B. in Cobol implementiert wurden), nicht zu. Zum anderen erzwingt sie Komponenten mit einer sehr feinen Granularität (da Komponenten spezielle Objekte sind und diese häufig sehr feingranular implementiert werden). So sind insbesondere fertige Anwendungen als Komponenten (z.B. für integrierte Systeme) nicht zugelassen.

Allgemeinere Definitionen (vgl. [Souz1998] und [Szyp1998]) lassen in Bezug auf die einzusetzende Technologie mehr Spielraum. Obwohl es auch hier zahlreiche verschiedene Begriffsbildungen gibt, lässt sich doch ein gemeinsamer Kern an wichtigen Eigenschaften finden, der auf folgende Definition führt (in Anlehnung an [Turo2001]):

Eine Komponente besteht aus verschiedenartigen (Software-) Artefakten. Sie ist wieder verwendbar, abgeschlossen und vermarktbar, stellt Dienste über wohl definierte Schnittstellen zur Verfügung, verbirgt ihre Realisierung und kann in Kombination mit anderen Komponenten eingesetzt werden. Zur Zeit der Entwicklung einer Komponente sind diese Kombinationen noch nicht unbedingt vorhersehbar.

Zu den (Software-) Artefakten sind insbesondere ausführbarer (compilierter) Code oder Quellcode zu zählen, sowie darüber hinaus die Spezifikation, (Anwender-) Dokumentation und (automatisierte) Tests. Als abgeschlossen bezeichnet man eine Komponente, falls sie in der Lage ist, die von ihr erfüllte Aufgabe zu erfüllen, ohne auf andere Komponenten angewiesen zu sein (in der Regel lässt man dabei allerdings Abhängigkeiten in Bezug auf das Vorhandensein einer bestimmten Middleware zu). Setzt man die Abgeschlossenheit voraus, kann man eine Komponente als ein für sich selbst stehendes Gut verstehen, dass auf einem (offenen oder unternehmensinternen) Komponentenmarkt gehandelt werden kann (also vermarktbar ist). Nach dieser Definition spielt die Granularität für den Komponentenbegriff zunächst keine Rolle. Es kann somit Komponenten unterschiedlichster Granularität geben, von elementaren Komponenten bis hin zu Anwendungen, die (beispielsweise im Enterprise Application Integration – EAI) ebenfalls als Komponenten aufgefasst werden können.

2.1 Unterschiedliche Formen der Wiederverwendung

Eine besondere Aufmerksamkeit ist der Wiederverwendung von Komponenten während der Anwendungsentwicklung zu schenken, da es hierbei mehrere Arten mit unterschiedlichen Vor- und Nachteilen für den Anwender bzw. Hersteller gibt. Abbildung 1 zeigt diese verschiedenen Arten der Wiederverwendung schematisch.

	logisch	**physisch**
White-Box	Opensource	—
Black-Box	Enterprise Java Beans, Microsoft DCOM	XML Web-Services, CORBA

Abbildung 1 Wiederverwendung von Komponenten

Zunächst ist zu unterscheiden, ob für die Wiederverwendung einer Komponente der Quellcode zur Verfügung gestellt wird, oder ob lediglich kompilierte (also direkt ausführbare) Artefakte zur Verfügung stehen. Im ersten Fall lassen sich Komponenten wieder verwenden, indem ihr Quellcode auf den Anwendungsfall angepasst wird. Man nennt diese Wiederverwendung durch Anpassung auch White-Box-Wiederverwendung, da die Innensicht der Komponente dem Anwendungsentwickler bekannt und veränderbar ist. So ist es beispielsweise möglich, eine vorhandene Komponente zur Bilanzierung nach einem Verfahren (z.B. HGB) durch Veränderung des Quellcodes zu einer Komponente anzupassen, die nach den Vorschriften eines anderen Verfahrens (z.B. IAS) bilanziert. White-Box-Wiederverwendung tritt typischerweise bei der Verwendung von Codebibliotheken oder Opensource Software auf.

Im anderen Fall lassen sich Komponenten nur durch Auswahl einer für den Anwendungszweck geeigneten Variante wieder verwenden. Dem Anwendungsentwickler ist lediglich bekannt, was die Komponente tut, aber nicht wie sie es tut. Daher spricht man im Falle der Wiederverwendung durch Auswahl auch von Black-Box-Wiederverwendung. Wird also eine Komponente zur Bilanzierung nach einer bestimmten Vorschrift gesucht, braucht man eine geeignete Variante der Komponente „Bilanzierung“, die genau dieses leistet. Diese Variante muss von einem Hersteller am Markt zur Verfügung gestellt werden. Black-Box-Wiederverwendung sollte den Regelfall am Softwaremarkt darstellen.

In der Praxis entpuppen sich die vermeintlichen Vorteile der White-Box-Wiederverwendung, die meist höhere Flexibilität durch bessere Anpassung verheißen, schnell als Nachteil. Zur Anpassung wird einerseits komplexes Fach- und Implementierungswissen benötigt. Andererseits kann von neuen, vom Hersteller verbesserten Versionen der Komponente nicht mehr automatisch profitiert werden. Die Black-Box-Wiederverwendung entlastet den Anwendungsentwickler von diesen Nachteilen, setzt allerdings eine entsprechende Variantenzahl jeder Komponente voraus. Dies kann in einem funktionierenden Komponentenmarkt jedoch sicher erwartet werden, zumal es dem Hersteller von Komponenten gelingt, sein (im Quellcode verborgenes) Implementierungswissen durch den Vertrieb von Black-Box-Komponenten

besser zu schützen. Durch das Konzept der Black-Box-Wiederverwendung wird der Anwendungsentwickler im Idealfall zum Konfigurator, der Anwendungen aus Komponenten zusammensetzt.

Über die obige Diskussion hinausgehend lassen sich (zumindest für Black-Box-Komponenten) logische und physische Wiederverwendung unterscheiden. Logische Wiederverwendung liegt immer dann vor, wenn eine Komponente repliziert und vom Hersteller an den Anwendungsentwickler ausgeliefert wird, was derzeit dem Regelfall am Softwaremarkt entspricht. Von physischer Wiederverwendung wird hingegen gesprochen, wenn ein entfernter Aufruf der Komponente (die beim Hersteller oder Service Provider gespeichert ist) erfolgt. Diese Art der Wiederverwendung wird vor allem durch XML Web-Services und das Application Service Providing (ASP) weitere Verbreitung finden.

Für die physische Wiederverwendung spricht hauptsächlich die weniger kostenintensive Nutzbarkeit von (Software-) Diensten durch den Anwender (der weder Hardwareressourcen bereitstellen noch sich um die Installation kümmern muss) sowie die effizientere Wartbarkeit der Komponenten durch den Hersteller, da nur wenige Installationsbasen zu ändern sind. Diesen Vorteilen steht jedoch eine Reihe von Nachteilen gegenüber. So sind seitens des Herstellers (bzw. Anbieters) solcher Dienste zunächst Fragen der Skalierbarkeit und Verfügbarkeit zu klären, für die er eine gewisse Garantie geben muss (Quality of Service). Darüber hinaus besteht bei Anwendern immer dann eine starke Zurückhaltung, wenn sie für den Aufruf einer entfernten Komponente kritische Daten über eine potenziell unsichere Leitung übermitteln sollen. So ist es beispielsweise für die Nutzung einer Komponente zur Bilanzierung notwendig, kritische Zahlen aus der Gewinn- und Verlustrechnung als Daten zu übergeben. Nicht zuletzt diese Zurückhaltung bei den Anwendern ist einer der Gründe, warum die Entwicklung des ASP die Erwartungen bislang nicht erfüllen konnte.

Es steht zu erwarten, dass sich (am offenen Markt) für die logische und physische Wiederverwendung von Softwarekomponenten voraussichtlich unterschiedliche Geschäftsmodelle entwickeln werden (vgl. [McKi2001]). Die logische Wiederverwendung ist in der Regel durch das Vorhandensein von Pauschalpreisen (für Überlassung, Verkauf oder Vermietung) einer uneingeschränkten Nutzungserlaubnis gekennzeichnet, während sich für die physische Wiederverwendung eher nutzungsabhängige Entgelte (Pay per Use, Flat Fee Modelle mit Maximallast etc.) zu etablieren scheinen. Beim Konfigurieren einer Anwendung aus Bausteinen bzw. bei der Herstellung von Komponenten ist daher die Art der Wiederverwendung (und damit die Art und Weise der Einbindung von Komponenten in das System) stets mit Bedacht zu berücksichtigen bzw. zu planen.

2.2 Komponentenarten für die Anwendungsentwicklung

Die gegebene Definition des Begriffs „Komponente“ ist bewusst recht allgemein gehalten und umfasst viele verschiedene Arten von Komponenten. Soweit es für die Architektur einer (komponentenorientierten) Anwendung nützlich sein kann, sollen einige Arten näher erwähnt werden (vgl. Abbildung 2). Zunächst lässt sich unterscheiden, ob eine Komponente anwendungsinvariante (generische) oder anwendungsspezifische (domänenspezifische) Dienste erbringt. Diese Unterscheidung ist sinnvoll, da generische Dienste domänenübergreifend für die verschiedensten Anwendungen wieder verwendet werden können und somit zur Basissoftware einer Anwendung gehören. Ein Beispiel für eine solche (komplexe) generische Komponente ist ein Datenbank-Managementsystem, das als Basissoftware für eine Datenbankanwendung eingesetzt wird. Als domänenspezifische Komponente ist beispielsweise die schon

erwähnte Komponente „Bilanzierung“ im Rahmen einer Finanzbuchführung anzusehen. Im Folgenden werden generische Komponenten als „Systemkomponenten“ (System Components) bezeichnet, während domänenspezifische mit dem Begriff „Fachkomponenten“ (Expert Components) umschrieben werden.

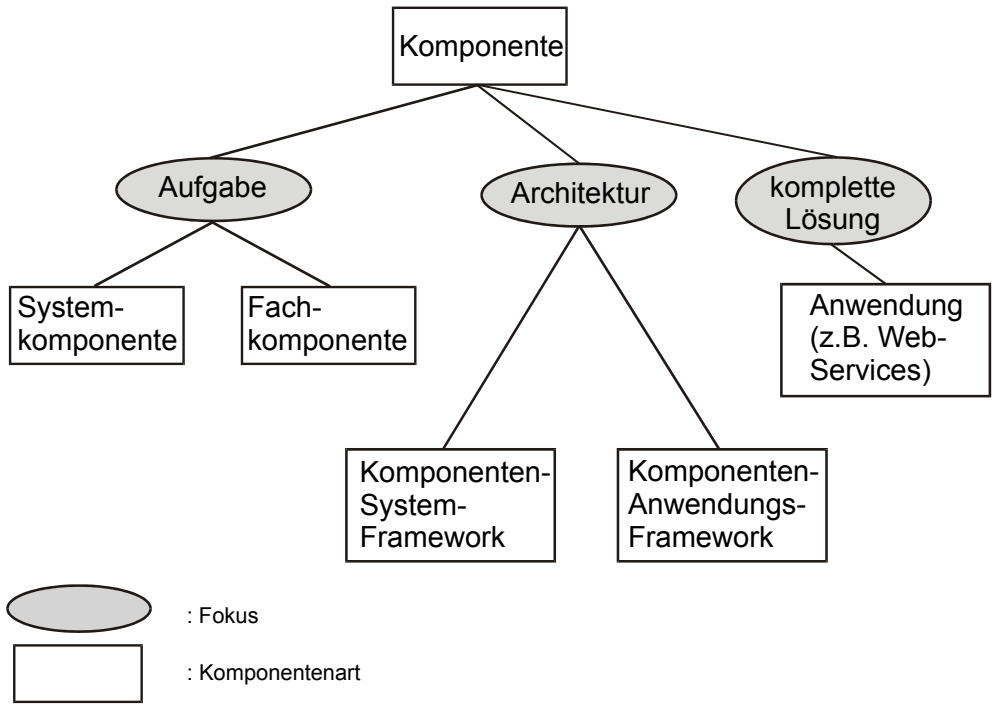

Abbildung 2 Komponentenarten

Unabhängig von diesem Kriterium gibt es noch eine zweite Unterscheidung im Hinblick auf Komponenten, die sich auf die Erfüllung einer Aufgabe konzentrieren, und solche, die eher ein Gerüst (also eine Architektur) für eine Anwendung zur Verfügung stellen. Im letzteren Falle spricht man statt von Komponenten auch von Frameworks. Ein solches Framework gibt also allgemeine Dienste und Regeln für das Zusammenwirken von Komponenten vor. Auch hier lassen sich generische (Komponenten-Systemframeworks bzw. System Frameworks) und domänenspezifische Frameworks (Komponenten-Anwendungsframeworks bzw. Application Frameworks) unterscheiden. Ein generisches Framework ist z.B. ein Betriebssystem oder eine Middleware wie die Technologie der Web-Services bzw. die eines EJB-Applikationsservers. Beispiele für ein anwendungsspezifisches Framework (aus der Domäne der betrieblichen Anwendungen) sind das San Francisco Framework von IBM (vgl. [IBM2000]) bzw. die entsprechenden CORBA Domain Interfaces.

Somit besteht eine komponentenorientierte Anwendung also ggf. aus folgenden Komponentenarten (in Anlehnung an [Raut2001]): einem Komponenten-Systemframework (das die Middleware-Schicht bildet), Systemkomponenten (die einzelne generische Dienste bereitstellen), einem Komponenten-Anwendungsframework (das allgemeine Dienste für eine Anwendungsdomäne bereitstellt und somit die Anwendungsplattform bildet) sowie aus Fachkomponenten (die spezialisierte Dienste der Anwendungsdomäne bereitstellen).

3 Ein einheitlicher Spezifikationsrahmen

Der Erfolg der komponentenorientierten Anwendungsentwicklung hängt entscheidend von einer einheitlichen, herstellerübergreifenden Spezifikation der Komponenten ab. Diese Spezifikation ist die Basis, auf der geeignete Komponenten ausgewählt und in die Anwendungsentwicklung integriert werden können. Im Rahmen eines Arbeitskreises der Gesellschaft für Informatik, in dem Vertreter aus der Industrie und der Universitäten zusammenkommen, wurde ein Vorschlag für die Vereinheitlichung der Spezifikation von Fachkomponenten entwickelt (vgl. [Turo2002]), der sich leicht zu einem Vorschlag für die einheitliche Spezifikation aller hier genannten Komponentenarten verallgemeinern lässt.

Kern dieses Vorschlags ist ein Spezifikationsrahmen, der neben wichtigen Aspekten einer Komponente auch Empfehlungen für den Einsatz konkreter Beschreibungssprachen (die am Markt bereits etabliert sind) vorgibt. Er dürfte daher (neben Universitäten) vor allem für die Abteilungen der betrieblichen Anwendungsentwicklung, die Standards und Methoden für die Anwendungsentwicklung vorgeben, sowie für Werkzeughersteller von Interesse sein.

Das Ziel eines Spezifikationsrahmens ist die möglichst vollständige Beschreibung der Außensicht einer Komponente, so dass ihre Eignung für einen Einsatz in der Anwendungsentwicklung möglichst allein auf Grund dieser Dokumentation (und ohne einen Testlauf) beurteilt werden kann. Dazu werden in der Praxis häufig Muster verwendet, welche entweder die Struktur oder den (fachlichen) Inhalt einer Komponente beschreiben. Eine verbreitete Darstellung von Mustern findet sich in abstrakten Datentypen, die sich auf die Spezifikation der Schnittstelle, der Vor- und Nachbedingungen für einzelne Dienste sowie die Angabe von Invarianten konzentrieren (vgl. [Grif1998]). Dort werden verschiedene Aspekte einer Komponente zusammen (und dadurch recht unübersichtlich) spezifiziert. Darüber hinaus reichen die gemachten Angaben für eine vollständige Beschreibung der Außensicht meist nicht aus. Der durch den Arbeitskreis standardisierte Spezifikationsrahmen greift diese Kritikpunkte auf und identifiziert zunächst verschiedene Aspekte einer Komponente, die sich den thematischen Bereichen „Administration“ (White bzw. Yellow Pages), „Konzeption (fachliche Semantik)“ (Blue Pages) und „Technik“ (Green Pages) zuordnen lassen. Durch die Gliederung in Aspekte wird zum einen eine Komplexitätsreduktion erreicht (da Spezifikationen jeweils thematisch zugeordnet werden können). Darüber hinaus ist dieser Spezifikationsrahmen ggf. um zusätzliche Aspekte erweiterbar. Auf diese Weise können zukünftige Versionen kompatibel gehalten werden. Abbildung 3 zeigt eine Übersicht, in der die einzelnen Aspekte zusammengefasst sind.

Die Administration umfasst zwei Aspekte, den Vermarktungs- und den Qualitätsaspekt. Der Vermarktungsaspekt spezifiziert zunächst allgemeine Informationen über eine Komponente. Dort werden z.B. der Hersteller, der Lieferumfang, die Version, Art der Wiederverwendung, die Komponentenart, die Systemanforderungen einer Komponente sowie die Plattform, für die sie entwickelt wurde, beschrieben. Darüber hinaus werden an dieser Stelle die Dienste einer Komponente klassifiziert, indem ihnen eine Anwendungsdomäne (Finanzbuchführung) bzw. eine generische Funktion (Datenverwaltung) zugeordnet wird, je nachdem, ob es sich um eine Fach- oder Systemkomponente handelt.

Der Qualitätsaspekt umfasst Aussagen über das Leistungsverhalten einer Komponente. Hierzu gehören Aussagen zur Antwortzeit, Lastverhalten, Quality of Service etc. Die Aspekte der Administration ermöglichen also zunächst einen Überblick über die Komponente. Sie dienen dem Entwickler, der nach einer passenden Komponente sucht, als ersten Anlaufpunkt. Von

dort ausgehend können weitergehende Aspekte für die Auswahl in Betracht gezogen werden. Dabei geht es zunächst um die Klärung fachlicher Aspekte, bevor schließlich technische Aspekte im Mittelpunkt seines Interesses stehen.

Hinter dem Schlagwort „Konzeption“ verbirgt sich dann auch die Beschreibung dieser fachlichen Semantik, also der thematischen Inhalte einer Komponente. Diese Inhalte sind im Falle einer Fachkomponente bzw. eines Komponenten-Anwendungsframeworks objektsprachliche Fachbegriffe (also Fachbegriffe eines Anwendungsbereichs), im Falle einer Systemkomponente bzw. eines Komponenten-Systemframeworks metasprachliche (generische) Termini. Die Beschreibung umfasst wiederum zwei Aspekte, den Aufgaben- und den Terminologieaspekt. Im Aufgabenaspekt wird vor allem die von der Komponente unterstützte Aufgabe definiert und ggf. in Teilaufgaben zerlegt. Bei Fachkomponenten steht an dieser Stelle eine Aufgabe aus der Anwendungsdomäne (z.B. Bilanzieren), bei Systemkomponenten eine generische Aufgabe (z.B. Daten persistent speichern).

Der Terminologieaspekt dient der zentralen Erfassung aller bei der Implementierung der Komponente verwendeten Fachbegriffe nebst Definition in einem Lexikon. Auf diese Weise kann nachvollzogen werden, ob eine Komponente im Rahmen einer Anwendungsentwicklung auch begrifflich für eine Auswahl in Frage kommt (vgl. [Ortn2000]). Insbesondere auf Grund regelmäßig fehlender fachlicher Standards ist die Spezifikation der fachlichen Semantik unablässig, wenn Entwickler sich vor möglichen (fachlichen) Überraschungen bei der Verwendung einer Komponente schützen wollen. Somit spielt sie bei der Auswahl einer Komponente eine zentrale Rolle.

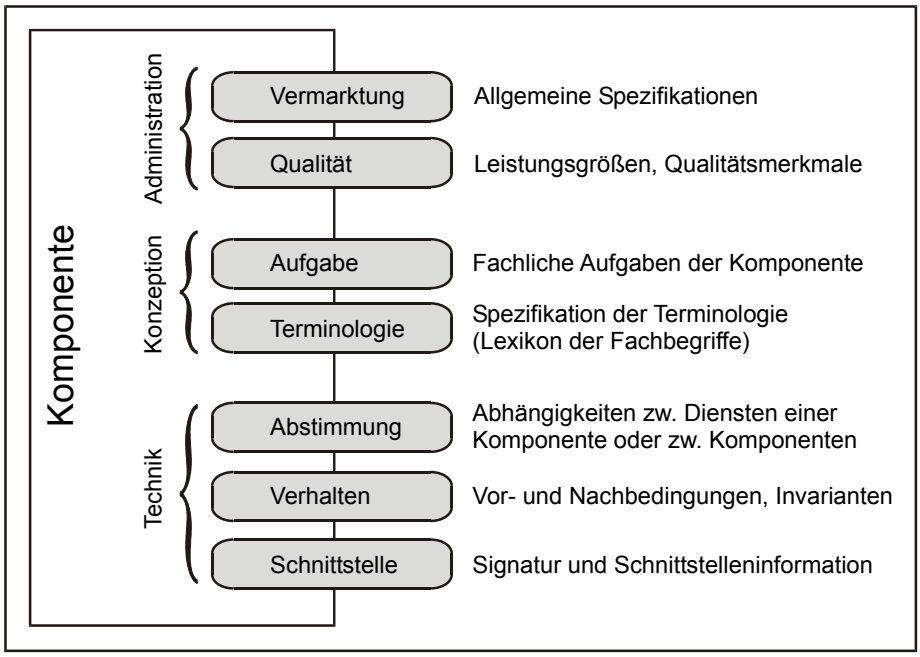

Abbildung 3 Einheitlicher Spezifikationsrahmen

Mit dem Schlagwort „Technik“ sind schließlich alle Aspekte zusammengefasst, die für die technisch korrekte Einbindung einer Komponente in eine Anwendung zu berücksichtigen sind. Dabei steht zunächst im Schnittstellenaspekt die Beschreibung der Schnittstelle einer

Komponente im Mittelpunkt. In diesem Aspekt wird dem Entwickler mitgeteilt, wie die einzelnen Dienste an der Schnittstelle benannt wurden (Methodennamen, Parameternamen und Datentypen), durch welche Dienste eine Aufgabe (aus dem Aufgabenaspekt) jeweils erfüllt wird, welche Ausnahmezustände dabei auftreten können etc. – es geht dabei also vornehmlich um die Beschreibung der Syntax. Hierfür werden vorzugsweise herstellerunabhängige Sprachen wie die Interface Description Language (IDL) bzw. die Web Services Description Language (WSDL) herangezogen (vgl. [Turo2002]).

Das Systemverhalten einer Komponente wird im gleichnamigen Verhaltensaspekt näher beschrieben. Hier geht es vor allem um die Formulierung von Invarianten und Vor- bzw. Nachbedingungen, die für einzelne Dienste der Schnittstelle gelten.

Vervollständigt wird die technische Spezifikation durch den Abstimmungsaspekt, der angibt, welche Abhängigkeiten es zwischen den einzelnen Diensten einer Komponente oder von den Diensten einer anderen Komponente gibt. Ein Beispiel für eine Abhängigkeit, von der die Komponente „Bilanzierung“ betroffen ist, wäre z.B. die (vereinfacht notierte) Aussage „…vor Aufruf der Bilanzierung muss eine Gewinn- und Verlustrechnung durchgeführt worden sein.“. Dies ist insbesondere dann hilfreich, wenn die Schnittstelle eine Vielzahl von Diensten anbietet, die teilweise voneinander abhängig sind (was in der Praxis nicht selten der Fall ist).

Wie gezeigt wurde, unterstützt der beschriebene Spezifikationsrahmen die ausführliche Dokumentation der Außensicht einer Komponente durch insgesamt sieben Aspekte. Diese stellen zugleich den Kern des Standards dar, der in jedem auf diesem basierenden Werkzeug unterstützt werden sollte. Darüber hinaus ist er für Erweiterungen auf Grund seines aspekteorientierten Aufbaus offen. Genauer ausgeführt sind die einzelnen Aspekte und die jeweils zum Einsatz empfohlenen Sprachen in [Turo2002]. Daher soll an dieser Stelle auf weitere Details verzichtet und stattdessen einige Anwendungen des Spezifikationsrahmens vorgestellt werden.

4 Vorgehensmodell und Werkzeugunterstützung

Der vorgenannte Spezifikationsrahmen dient nicht nur der Beschreibung von Komponenten, sondern ist zugleich auch die Basis für die Entwicklung von Vorgehensmodellen und die Implementierung von Werkzeugen. Im Folgenden zeigt ein einfaches, jedoch somit auch allgemein verwendbares Vorgehensmodell, wie unter Zuhilfenahme des Spezifikationsrahmens komponentenorientierte Anwendungen flexibel entwickelt werden können. Dabei zeigt sich die zentrale Bedeutung eines standardisierten Spezifikationsrahmens sowie einer entsprechenden Komponenten-Terminologie, die in das Vorgehensmodell aufgenommen wurden.

4.1 Das Multipfad-Vorgehensmodell

An der komponentenorientierten Anwendungsentwicklung sind IT-Spezialisten in den Rollen „Komponentenhersteller“, „Komponentenkonsument“ und „Komponentenmanager“ beteiligt. Der Komponentenhersteller implementiert und vertreibt Komponenten. Der Komponentenkonsument sucht für ihn passende Komponenten, um daraus Anwendungen zu bauen. Er wird daher im Folgenden auch als Anwendungsentwickler bezeichnet. Der Komponentenmanager verwaltet Komponenten in einer zentralen Ablage (Komponenten-Repository).

Im Falle eines unternehmensinternen Komponentenmarkts sind alle drei Parteien Angehörige des Unternehmens und können sogar in Personalunion auftreten. Darüber hinaus wäre es

denkbar, dass die Anwendungsentwickler den Fachbereichen des Unternehmens zugeordnet sind, die Komponentenhersteller hingegen eine zentrale (IT-) Abteilung bilden. Der Komponentenmanager administriert in diesem Fall ein Unternehmens-Repository. Im Falle eines offenen Marktes können die Rollen jedoch auseinander fallen. Der Komponentenmanager ist in diesem Fall der Betreiber eines offenen Komponenten-Verzeichnisses (Kataloges) oder eines Komponenten-Marktplatzes (Handelsplatzes). Zusätzlich setzen Produzent und Konsument idealerweise jeweils ein eigenes Unternehmens-Repository ein, um die entwickelten bzw. verwendeten Komponenten zu dokumentieren.

Ein einfaches Vorgehensmodell für die Anwendungsentwicklung gliedert sich in sechs Phasen: „Voruntersuchung“, „Fachentwurf mit Klassifikation“, „Systementwurf“, „Implementierung“, „Konfigurierung“ und „Stabilisierung“. Bei der Anwendungsentwicklung können diese auf unterschiedlichen Pfaden durchlaufen werden (vgl. Abbildung 4). So sind bei der Entwicklung von Individualsoftware alle diese Phasen zu durchlaufen. Für die komponentenorientierte Anwendungsentwicklung ergeben sich jedoch unter Umständen Abkürzungen, die zu einer Zeit- und Kostenersparnis führen.

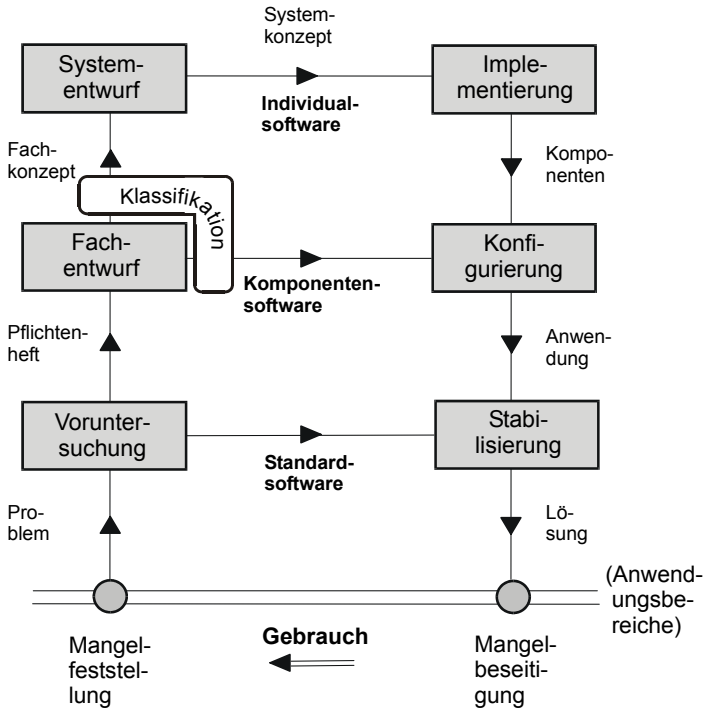

Abbildung 4 Flexibles Multipfad-Vorgehensmodell der komponentenorientierten Anwendungsentwicklung

Während der Voruntersuchung wird zunächst vom Anwendungsentwickler ein Pflichtenheft erstellt, das neben der Aufgabenstellung auch eine Machbarkeitsstudie enthält. Gelingt es ihm, zur Erfüllung der Aufgabenstellung eine passende Standardsoftware einzusetzen, kann er die Entwicklung bereits hier beenden und gelangt in die Stabilisierung. Anderenfalls begibt er sich in die Phase des fachlichen Entwurfs, in der er (im Dialog mit den Endanwendern) fachliche Aussagen über die zu schaffende Anwendung ermittelt. Die Phase endet mit der Erstellung des Fachkonzepts, das aus einer Aussagensammlung auf der Basis geklärter Fachbegriffe besteht. Mit dieser Aussagensammlung wird der Anwendungsentwickler nun in die Lage ver-

setzt, passende Komponenten für eine Anwendung zu suchen. Dazu muss er die Aussagen jedoch derart ordnen, dass sie ihm konkrete Vorgaben für die Suche nach möglichen Komponenten geben. Man nennt diesen Schritt auch Klassifikation.

Bei der Klassifikation spielt nun der verwendete Spezifikationsrahmen bzw. die verwendete Komponenten-Terminologie eine Rolle, da durch diese ein Klassifikationsschema determiniert werden kann. Durch dieses wird festgelegt, nach welchen Aspekten die Aussagen aus dem Fachkonzept für die Suche nach Komponenten zu gliedern sind. Abbildung 5 zeigt ein zweidimensionales Klassifikationsschema, das Aussagen einmal in solche über die (in dieser Arbeit genannten) verschiedenen Komponentenarten differenziert. Zum anderen werden die Aussagen den verschiedenen Aspekten des beschriebenen Spezifikationsrahmens zugeordnet.

Mit einem solchen Schema wendet sich der Anwendungsentwickler anschließend an einen Komponentenmanager, der ihn auf dessen Basis bei der Suche nach geeigneten Komponenten unterstützen kann. Gelingt es dem Anwendungsentwickler dabei, alle nötigen Komponenten zu finden (und zu erwerben), kann er aus diesen seine Anwendung konfigurieren und anschließend testen. Gelingt ihm dies nicht, muss er den Bau der fehlenden Komponenten bei einem Produzenten in Auftrag geben (oder selbst besorgen). Nur in dem Fall wird die Phase „Implementierung“ noch durchlaufen.

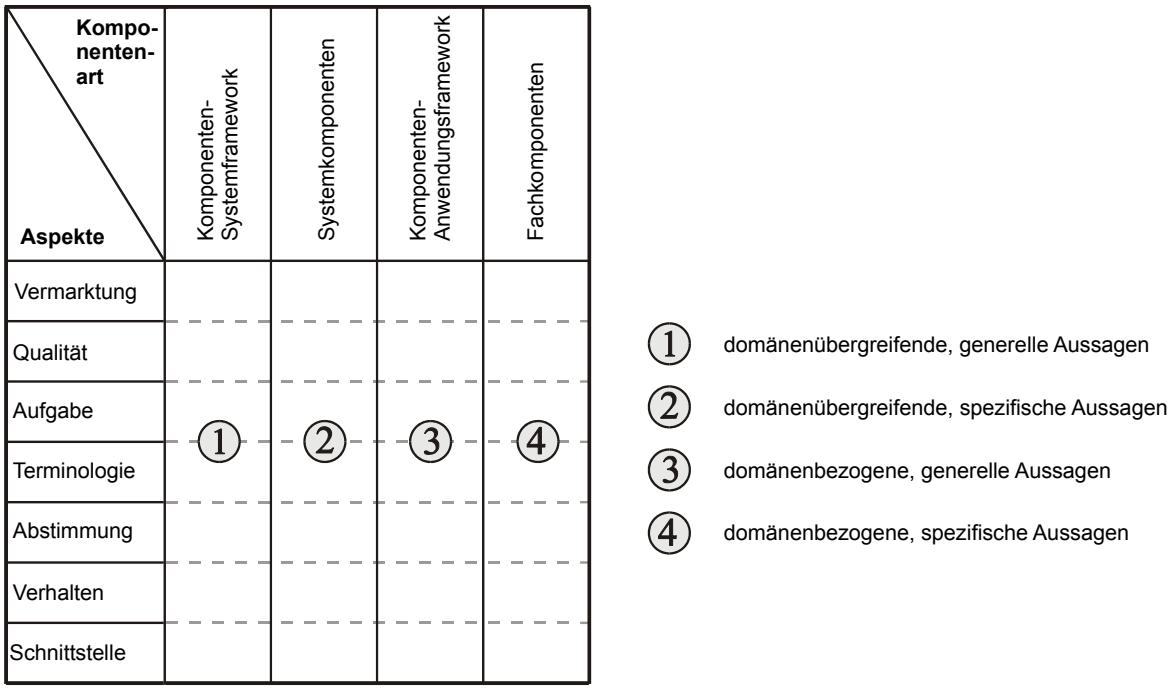

Abbildung 5 Klassifikationsschema für die Suche nach Komponenten

Schon aus diesem einfachen Vorgehensmodell wird ersichtlich, dass die komponentenorientierte Anwendungsentwicklung im Allgemeinen zu einem erheblich flexibleren Entwicklungsprozess führt. Detaillierte Organisations- und Vorgehensmodelle für die komponentenorientierte Anwendungsentwicklung können auf seiner Basis von den Unternehmen individuell erarbeitet werden (einen Ansatz hierfür liefert [Sahm2000]).

4.2 Werkzeugunterstützung durch Komponenten-Repositories

Die komponentenorientierte Anwendungsentwicklung lässt sich an vielen Stellen von Werkzeugen unterstützen. Von zentraler Bedeutung sind dabei diejenigen für die Verwaltung und Administration der Komponenten in einem Unternehmen. Diese Werkzeuge, die Komponenten-Repositories genannt werden, dokumentieren, welche Komponenten entwickelt bzw. in welchen Anwendungen verwendet wurden. Diese Funktionalität ist vor allem dann sinnvoll, wenn eine Komponente einen Versionswechsel erfährt und die betroffenen Systeme gewartet werden müssen. Schließlich dienen sie als Teilelager für die Anwendungsentwicklung und unterstützen Entwickler bei der Suche nach geeigneten Komponenten. Sowohl für den Komponentenproduzenten als auch den -konsumenten ist der Einsatz eines Komponenten-Repository daher sinnvoll. Ein Komponenten-Repository ist eine speziell zur Administration von Komponenten entwickelte Datenbankanwendung. Hierzu liefert sie ein entsprechendes Metaschema mit (vgl. [Ortn1999b]), das die Art der Dokumentation bestimmt. Der im Rahmen dieser Arbeit beschriebene Spezifikationsrahmen ist daher auch für die Entwicklung solcher Werkzeuge von Bedeutung. Er kann als Kern eines solchen Metaschemas dienen, das ggf. um weitere Aspekte (wie die Dokumentation der Anwendungen) zu ergänzen ist.

Über die beschriebene Funktionalität hinaus wäre vor allem für die Phasen „Klassifikation“ und „Konfiguration“ eine Unterstützung des Entwicklers durch das Komponenten-Repository wünschenswert. So ist es denkbar, dass die Auswahl einer optimalen Kombination von Komponenten für eine zu entwickelnde Anwendung auf Basis des Klassifikationsschemas automatisiert erfolgt. Hinter dieser Aufgabenstellung verbirgt sich ein komplexes Optimierungsproblem, das sich mit Methoden des Operations Research (zumindest annähernd) lösen lässt. Leider kann man zeigen, dass eine exakte optimale Lösung nur für Systeme, die aus relativ wenigen Komponenten zusammenzusetzen sind, mit vertretbarem Aufwand zu ermitteln ist (vgl. [Kauf2000]). Somit kommen hierfür bevorzugt heuristische Verfahren zum Einsatz, die nicht immer die beste Lösung liefern. Ein viel versprechender Ansatz, der zudem noch garantiert optimale Lösungen liefert, basiert auf dem Einsatz von Variantenstücklisten (vgl. [Wede1981]) und ist derzeit Gegenstand eines Forschungsprojektes. Die Konfigurierung eines optimalen Systems aus verschiedenen alternativen Komponenten lässt sich jedoch auch ohne Automatisierung durch Methoden wie den Morphologischen Kasten (vgl. Abbildung 6) hinreichend unterstützen. Dieser ermöglicht die Analyse verschiedener Lösungen und die Auswahl einer optimalen Kombination von Komponenten.

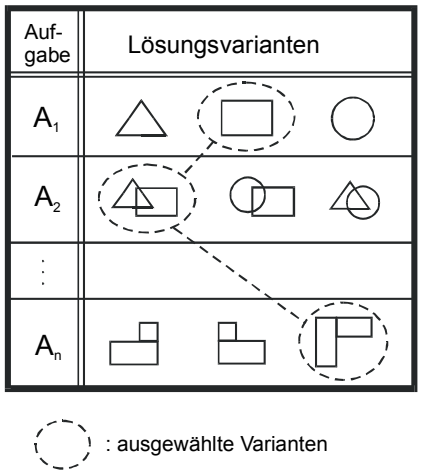

Abbildung 6 Morphologischer Kasten

5 CompoNex – ein Marktplatz für den Handel mit Software-Komponenten über das Internet

Eine weitere wesentliche Voraussetzung für den Erfolg der komponentenorientierten Anwendungsentwicklung ist das Vorhandensein eines Komponentenmarktes, der für einen reibungslosen Austausch von Software-Komponenten zwischen Anbietern und Nachfragern sorgt. Hier zeichnet sich insbesondere durch die sich etablierende Technologie der XML Web-Services ein spürbarer Fortschritt ab. Bestandteil dieser Technologie ist UDDI (Universal Description, Discovery and Integration), ein Standard für offene Verzeichnisse, in denen Web-Services registriert werden können (vgl. [UDDI2000]). Damit existiert für das Auffinden von Komponenten, die als XML Web-Service ansprechbar sind, bereits ein zentrales Komponenten-Repository. Ein Nachteil der Verzeichnisse, die auf dem Standard UDDI basieren, ist jedoch die Tatsache, dass sie sich auf Komponenten des Typs XML Web-Service beschränken und darüber hinaus lediglich die technische Schnittstelle spezifizieren (also nur den Schnittstellenaspekt bei der Spezifikation berücksichtigen). Hierdurch wird die Beurteilung einer Komponente durch den Entwickler erschwert.

Am Fachgebiet Wirtschaftsinformatik I der TU Darmstadt wurde im Rahmen eines Dissertationsprojektes (prototypisch) ein elektronischer Marktplatz für den Handel mit Software-Komponenten jeder Art entwickelt, der in einem Spin-Off vermarktet wird. Dieser Marktplatz trägt den Namen CompoNex (Component Exchange) und berücksichtigt die Tatsache, dass es sich bei Software-Komponenten um erklärungsbedürftige Güter handelt, deren Handel durch spezielle Dienste unterstützt werden sollte. So implementiert er als Basis des Komponenten-Kataloges den hier vorgestellten Spezifikationsrahmen und verfügt über eine geordnete (aspektorientierte) Darstellung der Außensicht von Komponenten. Er unterstützt darüber hinaus den Vertrieb sämtlicher Komponentenarten, die in Abschnitt 2 vorgestellt wurden und beschränkt sich dabei nicht auf Komponenten einer bestimmten Technologie. Durch die Implementierung des Spezifikationsrahmens integriert sich der Marktplatz in das geschilderte Vorgehensmodell und ermöglicht die Suche nach Komponenten auf Basis des Klassifikationsschemas. Darüber hinaus diente die Implementierung während der Standardisierungsarbeiten zugleich als „Proof of Concepts“.

5.1 Architektur und Dienste am Marktplatz

CompoNex basiert auf einem datenbankgestützten Komponenten-Repository, das den hier erwähnten Spezifikationsrahmen als Metaschema implementiert und für jede Komponente die Dokumentationsaspekte „Vermarktung“, „Qualität“, „Aufgabe“, „Terminologie“, „Abstimmung“, „Verhalten“ und „Schnittstelle“ zur Verfügung stellt. Es unterscheidet darüber hinaus zwischen Fach- und Systemkomponenten bzw. Komponenten-Anwendungsframeworks und Komponenten-Systemframeworks und gibt Auskunft darüber, ob eine Komponente logisch oder physisch wieder verwendet werden kann.

Der darauf aufbauende Komponentenkatalog kann von einem Marktplatzbesucher schrittweise durchstöbert werden, wobei er zunächst die gewünschte Komponentenart wählt und sich anschließend eine Anwendungsdomäne bzw. generische Funktionalität aussucht. Auf diese Weise kann er sich zu den gewünschten Komponenten vorarbeiten, wobei konzeptionelle (und nicht technologische) Überlegungen wie die Komponentenart oder die Funktionalität (Aufgabe) der Komponente im Vordergrund stehen.

Darüber hinaus bietet der Marktplatz zwei Suchfunktionen, um gezielt nach einzelnen Komponenten zu suchen. Die Volltextsuche ermöglicht die einfache Angabe von Schlüsselwörtern, die dann in den Spezifikationsaspekten „Vermarktung“, „Aufgabe“ und „Terminologie“ gesucht werden. Eine parametrisierbare Suche ermöglicht die Angabe detaillierter Suchwerte in den vorgenannten Aspekten, so dass die Suche stärker eingeschränkt werden kann. Auf diese Weise wird es zum Beispiel möglich, nach Komponenten, die die Aufgabe „Bilanzierung“ implementieren, als Enterprise Java Bean gefertigt sind und von SAP stammen, zu suchen.

Die Suchfunktion ist dabei mit semantischer Intelligenz ausgestattet, so dass sie in der Lage ist, semantisch verwandte Schlüsselwörter gewichtet in die Suche mit einzubeziehen. Sie wird Bezug nehmend auf das vorgenannte Beispiel erkennen, dass Komponenten der Domäne „Finanzbuchführung“ in die Suche aufzunehmen sind und somit auch Komponenten, welche die Aufgabe „Gewinn- und Verlustrechnung“ implementieren, einbeziehen (und mit einem niedrigeren Rang versehen).

Aufbauend auf dieser Suchfunktion bietet der Marktplatz einen Subskriptionsdienst (basierend auf einem Publish/Subscribe Algorithmus), über den Suchabfragen periodisch ausgeführt und deren Ergebnisse dem Besucher als Abonnement (per E-Mail, SMS etc.) zugestellt werden können. Auf diese Weise kann er automatisch auf dem aktuellen Stand gehalten werden.

Bei der Implementierung des Marktplatzes für den Komponentenhandel war ferner zu berücksichtigen, dass die zugrunde liegenden Geschäftsprozesse möglichst direkt unterstützt werden. So erfolgt die Benutzung des Marktplatzes vorwiegend während des Entwicklungsprozesses (in der Phase „Fachentwurf mit Klassifikation“), wodurch eine direkte Integration in die CASE Tools des Anwenders (beispielsweise sein Entwicklungswerkzeug oder sein Komponenten-Repository) erforderlich wurde. CompoNex wurde daher als XML Web-Service implementiert und stellt somit selbst eine Software-Komponente dar, die in andere Anwendungen eingebunden werden kann.

Im Rahmen der Implementierung wurde der Marktplatz sowohl auf die Benutzung über eine webbasierte Schnittstelle (die unter www.componex.biz verfügbar ist) ausgelegt sowie exemplarisch in das CASE Tool Microsoft Visual Studio .NET über ein Plug-In integriert. Abbildung 7 zeigt diese beiden Benutzungsoberflächen in der Zusammenschau. Anwendungen von

Drittanbietern steht zur Integration des Marktplatzes die (in ihrer Außensicht nach dem Spezifikationsrahmen) dokumentierte Schnittstelle des CompoNex Web-Service zu Verfügung. Auf diese Weise ist der Komponentenmarktplatz ohne Medienbruch zu dem Zeitpunkt ansprechbar, zu dem er auch tatsächlich nachgefragt wird.

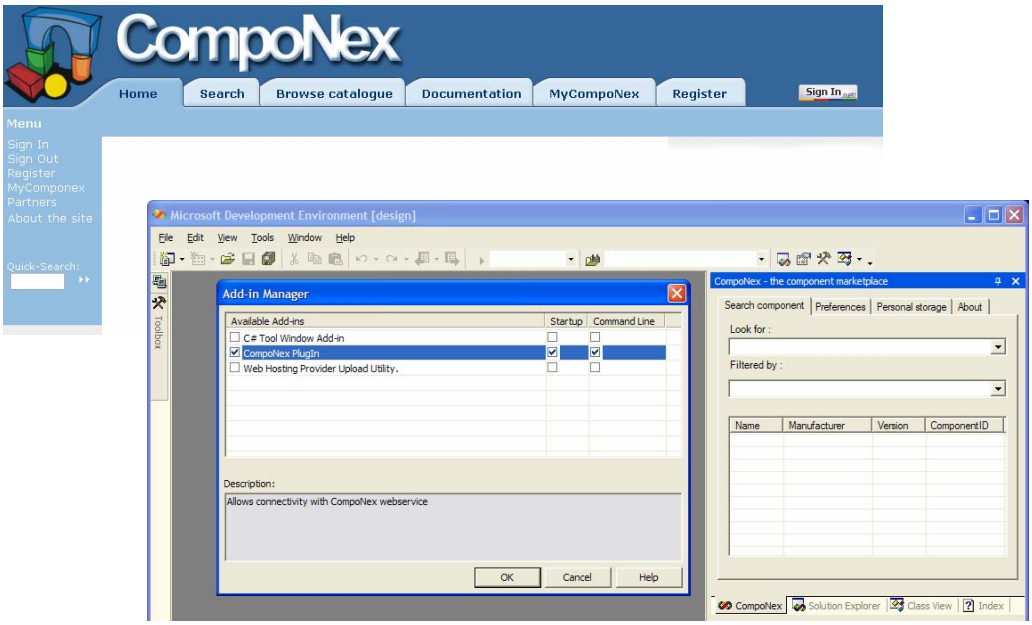

Abbildung 7 Web-Benutzungsschnittstelle und CASE Tool Integration

Aus den vorgenannten Anforderungen während der Systementwicklung ergibt sich für den Marktplatz eine komplexe Architektur, die (auszugsweise) in Abbildung 8 dargestellt ist. Die gezeigte Architektur setzt auf einem komponentenorientierten Middleware-Framework für die Entwicklung von E-Commerce-Anwendungen auf, das unter dem Namen E-NOgS (Electronic New Organon {Server, Service, Servant}) zuvor bereits am Fachgebiet entwickelt wurde.

Darüber hinaus verfügt der Marktplatz über ein differenziertes Geschäftsmodell (vgl. [Merz2002]), das Zusatzdienste wie den Subskriptionsdienst gegen eine Gebühr offeriert und dem Anbieter von Komponenten auf einer Provisionsbasis erfolgreich verkaufte Güter in Rechnung stellt. Darüber hinaus kann (auf Wunsch des Anbieters) die Speicherung von Komponenten in ein eigenes Komponentenlager des Marktplatzes vorgenommen werden. Somit werden insbesondere kleinere Komponentenanbieter vom Aufbau einer eigenen Vertriebsorganisation entlastet und können in den Komponentenmarkt eintreten.

Durch die detaillierte Spezifikation der Komponenten und die angebotenen Zusatzdienste hebt sich der Marktplatz von anderen am Markt befindlichen Komponentenkatalogen (wie beispielsweise UDDI) ab und besitzt somit auch die Chance, sich erfolgreich am entstehenden Komponentenmarkt zu etablieren. Für seine Konzeption und die Implementierung wurde er im Rahmen eines Industrie-Wettbewerbs der Firma Microsoft bereits mit einem Preis ausgezeichnet.

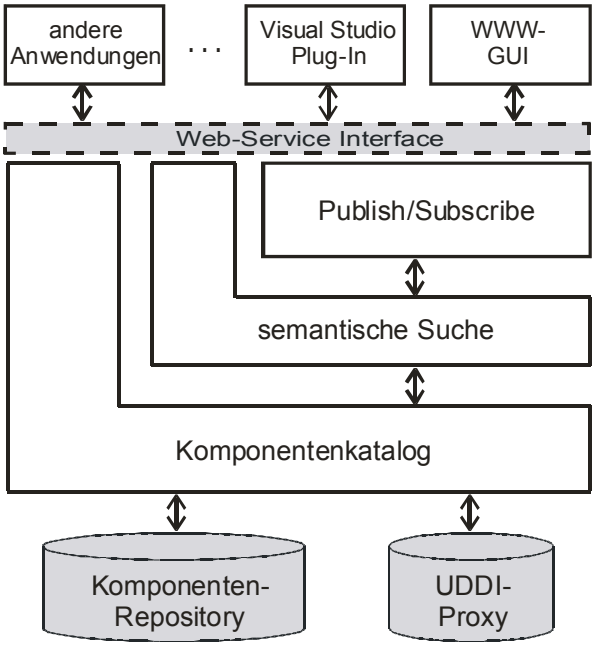

Abbildung 8 Architektur von CompoNex (in Auszügen)

6 Ausblick

Im Rahmen dieses Beitrags wurde gezeigt, dass die einheitliche Spezifikation von Komponenten eine Schlüsselbedeutung für die komponentenorientierte Anwendungsentwicklung besitzt. Mit der Etablierung von Standards in den Themenfeldern „Spezifikation“ und „Komponententechnologie“ (Web-Services als Kommunikationsplattform) lassen sich wesentliche Fortschritte erreichen. Lediglich auf dem Gebiet der fachlichen Verschiedenheiten (Terminologien) sind Standards bislang kaum in Sicht. Einen möglichen Ausweg aus dieser Situation zeigt indes der hier geschilderte Spezifikationsrahmen, der die verwendete Semantik explizit dokumentiert und dadurch ggf. den Bau von Übersetzern ermöglicht. Darüber hinaus zeichnen sich in einigen Branchen (wie dem deutschen Versicherungswesen) Bewegungen mit dem Ziel ab, fachliche Standards im Hinblick auf die Schaffung eines offenen Komponentenmarkts zu etablieren.

Die Komponentenorientierung benötigt für ihren Durchbruch eine umfassende Unterstützung durch Werkzeuge (einige Aufgaben wurden in Abschnitt 4 bereits angesprochen) und Methoden. Insbesondere die Schaffung einer Modellierungssprache, welche die hier genannten (konzeptionellen) Komponentenarten als Architekturbestandteile unterstützt, wäre (beispielsweise als Erweiterung der UML) wünschenswert. Darüber hinaus bleibt im Hinblick auf die Verbreitung einer eher konzeptionellen Komponenten-Terminologie eine gewisse Überzeugungsarbeit gegenüber den IT-Experten zu leisten, die bislang eine eher implementierungsnahe Terminologie verwenden. Diese ist (analog zu dem bekannten Beispiel der Datenverwaltungssysteme) jedoch für die Architekturbeschreibung und Modellierung unzureichend.

Literatur

[Bett2001] *Bettag, U.:* Web-Services. In: Informatik Spektrum 24 (2001) 5, S. 302 – 304.

[Grif1998] *Griffel, F.:* Componentware – Konzepte und Techniken eines Softwareparadigmas. dpunkt-Verlag, Heidelberg 1998.

[IBM2000] *IBM Corporation (Hrsg.):* IBM San Francisco Overview. IBM Corporation, 2000. http://www.ibm.com/software/ad/sanfrancisco, Abruf am 1.3.2002.

[Kauf2000] *Kaufmann, T.:* Entwurf eines Marktplatzes für heterogene Komponenten betrieblicher Anwendungssysteme. Berlin 2000.

[McKi2001] *McKie, S.:* Web Services – A Manager's Guide. Content Can, 2001.

[Merz2002] *Merz, M.:* E-Commerce und E-Business – Marktmodelle, Anwendungen und Technologien. dpunkt-Verlag, Heidelberg 2002.

[Orfa1996] *Orfali, R. et al:* The Essential Distributed Objects Survival Guide. John Wiley & Sons, New York 1996.

[Ortn1999a] *Ortner, E. et al:* Anwendungssystementwicklung mit Komponenten. In: IM Information Management & Consulting 14 (1999) 2, S. 35 – 45.

[Ortn1999b] *Ortner, E.:* Repository Systems. Teil 1: Mehrstufigkeit und Entwicklungsumgebung. In: Informatik Spektrum 22 (1999) 4, S. 235 – 251. Teil 2: Aufbau und Betrieb eines Entwicklungsrepositoriums. In: Informatik Spektrum 22 (1999) 5, S. 351 – 363.

[Ortn2000] *Ortner, E.:* Terminologiebasierte, komponentenorientierte Entwicklung von Anwendungssystemen. In: *Flatscher, R. et al (Hrsg.):* Tagungsband 2. Workshop Komponentenorientierte betriebliche Anwendungssysteme. Wien 2000, S. 1 – 17.

[Raut2001] *Rautenstrauch, C.; Turowski, K.:* Common Business Component Model (COBCOM): Generelles Modell komponentenbasierter Anwendungssysteme. In: *Buhl, H. U. et al (Hrsg.):* Information Age Economy. Physica Verlag, Heidelberg 2001.

[Sahm2000] *Sahm, S.:* Organisationsmodelle zur komponentenorientierten Anwendungsentwicklung. In: *Turowski, K. (Hrsg.):* Modellierung und Spezifikation von Fachkomponenten: Workshop im Rahmen der MobIS 2000 Modellierung betrieblicher Informationssysteme. Siegen 2000, S. 17 – 40.

[Souz1998] *d'Souza, D.; Wills, A.:* Objects, Components and Frameworks with UML – The Catalysis Approach. Addison-Wesley, Reading (Massachusetts) 1998.

[Szyp1998] *Szyperski, C.:* Component Software: Beyond Object-Oriented Programming. Addison-Wesley, Harlow 1998.

[Turo2001] *Turowski, K.:* Spezifikation und Standardisierung von Fachkomponenten. In: Wirtschaftsinformatik 43 (2001) 3.

[Turo2002] *Turowski, K. (Hrsg.):* Vereinheitlichte Spezifikation von Fachkomponenten. Gesellschaft für Informatik (GI), Arbeitskreis 5.10.3, Augsburg, 2002. http://www.fachkomponenten.de, Abruf am 20.4.2002.

[UDDI2000] *UDDI Organization (Hrsg.):* UDDI Technical White Paper. UDDI Standards Organization, September 2000. http://www.uddi.org, Abruf am 1.3.2002.

[Wede1981] *Wedekind, H.; Müller, T.:* Stücklistenorganisation bei einer großen Variantenzahl. In: Angewandte Informatik 23 (1981) 9, S. 377 – 383.