Ansätze der Forschung zur Unterstützung der Entwicklung von Softwarekomponenten

Tobias Hildenbrand und Michael Schwind

Arbeitspapier 3/2005
Februar 2005

Arbeitspapiere in der Wirtschaftsinformatik

Universität Mannheim
Lehrstuhl für Wirtschaftsinformatik I/III
D-68131 Mannheim/Germany

Ansätze der Forschung zur Unterstützung der Entwicklung von Softwarekomponenten

Tobias Hildenbrand

Lehrstuhl für ABWL und Wirtschaftsinformatik I

Universität Mannheim

D-68131 Mannheim

hildenbrand@uni-mannheim.de

Michael Schwind

Lehrstuhl für Wirtschaftsinformatik III

Universität Mannheim

D-68131 Mannheim

schwind@uni-mannheim.de

30. September 2004

Zusammenfassung

Das vorliegende Arbeitspapier entstand aus einem Vortrag, welcher im Rahmen einer Veranstaltung der Wirtschaftsinitiative „Baden-Württemberg: Connected“ (bwcon) aus der Reihe „Forum Unternehmenssoftware“ gehalten wurde. Das übergeordnete Ziel dieser Veranstaltung war, die zukünftigen Perspektiven – Chancen und Herausforderungen – komponentenbasierter Software für Softwarefirmen und -anwender aufzuzeigen und zu diskutieren. Der Beitrag der Universität Mannheim bestand darin, die aktuellen Ansätze der Forschung zur Unterstützung der komponentenbasierten Softwareentwicklung im Überblick darzustellen, insbesondere eigene Ansätze und Projekte zu diesem Thema vorzustellen und einen Ausblick auf zukünftige Entwicklungen zu geben. Die wichtigsten Punkte des Vortrags sowie die Ergebnisse der anschließenden Diskussionsrunde mit Anwendern von Unternehmenssoftware, Softwareherstellern sowie weiteren Wissenschaftlern auf diesem Gebiet werden in diesem Papier zusammengefasst.

1 Einleitung

Während die Programmierung von Computersystemen lange Zeit auch nach der 1968 erstmals identifizierten „Software-Krise“ noch eher als Kunstform mit wissenschaftlichem Ansatz denn Ingenieursdisziplin galt ([13]), ist heute ein deutlicher Trend hin zur „Industrialisierung“ der Softwareentwicklung erkennbar (vgl. [16]). Die ständig wachsende Nachfrage nach umfassenden Anwendungen zur Unterstützung unterschiedlicher betrieblicher Aufgaben und Prozesse in nahezu allen Branchen machen die Software-Industrie selbst immer mehr zu einer Leitindustrie ([3]).
Unter der Bezeichnung „Softwaretechnik“ bzw. *Software Engineering* im englischen Sprachraum entstand ein eigener Forschungszweig der Informatik, welcher sich mit der Erstellung von Software nach ingenieurwissenschaftlichen Prinzipien beschäftigt. Das zentrale Paradigma bei der ingenieurmäßigen Softwareentwicklung, neben einem generell systematischen Vorgehen, ist die **Wiederverwendung**, welche sich in unterschiedlichen Ausprägungen in allen modernen Ingenieurwissenschaften widerspiegelt, ohne immer explizit als solche benannt zu sein (vgl. [4]). Die zwei grundlegenden Formen von Wiederverwendung sind in der Softwaretechnik Kompositions- sowie Generierungstechniken. Beispiele hierfür sind im Wesentlichen die komponentenbasierte Softwareentwicklung (KBSE) bzw. die modellgetriebene Entwicklung (*Model-Driven Development*, MDD) in Verbindung mit (semi-)automatischer Quelltextgenerierung anhand sogenannte „Modellcompiler“ ([12]).
Ziel des vorliegenden Arbeitspapiers ist es, den aktuellen Stand der Forschung hinsichtlich der Unterstützung von Komponentenentwicklungsaktivitäten durch entsprechende Wiederverwendungstechnologien, Methoden und -erkzeugen strukturiert darzustellen. Diese Darstellung soll darüber hinaus mittels Beispielen aus kürzlich abgeschlossenen und laufenden Forschungsprojekten der Universität Mannheim zu illustriert werden. Abschließend soll eine auf Forschungsergebnisse und Praxisanforderungen basierende, fundierte Prognose für die weitere Entwicklung der komponentenbasierten Softwaretechnik und deren erwartete Diffusion in der Praxis erstellt werden.
Zu diesem Zweck werden nach der Darstellung wesentlicher Grundlagen der Komponentenorientierung spezielle Vorgehensmodelle und Entwicklungswerkzeuge zur Erstellung von Softwarekomponenten kategorisiert und analysiert. Im Anschluss daran werden exemplarisch die aktuellen Forschungsvorhaben der Universität Mannheim, BOOSTER, OMEGA und COLLABAWÜ vorgestellt und in den wissenschaftlichen Kontext der KBSE eingeordnet. Ergänzend dazu, werden die forschungsgetriebenen Ansätze jeweils durch Aussagen von Praktikern und Ergebnisse der entsprechenden Diskussionsrunde. Den Abschluss bildet ein Ausblick auf die weitere Entwicklung der Softwa-

retechnik als Ingenieurwissenschaft, welcher auf Erkenntnissen aus sowohl universitärer Theorie als auch anwendungsnaher Praxis basiert.

2 Grundlagen der Komponentenorientierung

Komponentenorientierte Programmierung (*Component-Oriented Programming*) wird in der Literatur häufig als logische Fortführung des objektorientierten Paradigmas dargestellt (z.B. bei [19]). Grundlegende Konzepte, wie Polymorphismus, modulare Kapselung, spätes Binden und Laden sowie Typen- und Modulsicherheit, lassen sich im Wesentlichen von Objekten auf Komponenten übertragen. Beispielsweise führt die modulare Kapselung (*Information Hiding*) bei grobgranularen Komponenten zu einem deutlich höheren Grad der Wiederverwendung, erschwert allerdings im Gegenzug auch die Wiederverwendbarkeit in unterschiedlichen Problemdomänen.
Die Definition des Arbeitskreises 5.10.3 der deutschen Gesellschaft für Informatik (Komponentenorientierte betriebliche Anwendungssysteme) greift die meisten der oben genannten Konzepte wieder auf und gestaltet sich folgendermaßen:

> „Eine **Komponente** besteht aus verschiedenartigen Software-Artefakten. Sie ist wiederverwendbar, abgeschlossen und vermarktbar, stellt Dienste über wohldefinierte Schnittstellen zur Verfügung, verbirgt ihre Realisierung und kann in Kombination mit anderen Komponenten eingesetzt werden, die zur Zeit der Entwicklung nicht unbedingt vorhersehbar ist.“ [21]

Eine **Geschäfts-** oder auch **Fachkomponente**, als Spezialfall einer Softwarekomponente, kapselt und bietet „eine bestimmte Menge von Diensten einer betrieblichen Anwendungsdomäne“ an [21]. Weitere Betrachtungen des Komponenten- und Fachkomponentenkonzepts finden sich u.a. auch bei Korthaus [14] sowie [9]. Während ein Großteil der einschlägigen Autoren Softwarekomponenten lediglich als ausführbare Kompositionseinheit betrachten, erscheint die holistische Sichtweise auf die Komponente als **logisches Konzept**, egal ob auf Modell-, Code- oder Bytecode- bzw. Binärebene, aus Sicht der Forschung als sinnvoller (vgl. [2]). Die hierarchisch-rekursive Zerlegung von Anwendungssystemen in „Stücklisten“ (*Containment Trees*, [2]) von Komponenten und Unterkomponenten ermöglicht eine bessere Abstraktion von Aussagen in der KBSE-Forschung, ohne beispielsweise die bisher nicht ausreichend beantwortete Frage nach der optimalen Granularität von Komponenten konkret beantworten zu müssen.

Zur Unterstützung der Entwicklung von und mit Komponenten in der Softwaretechnik haben sich daher in den letzten Jahren einige Technologien, Standards und Vorgehensmodelle entwickelt und wenige wirklich etabliert. Besonders ausgereift sind mittlerweile die serverseitigen Komponenten-Programmiermodelle und -Laufzeitumgebungen (*Application Server*). In diesem Bereich konkurrieren im Wesentlichen die Ansätze von Sun (*Enterprise Java Beans*, EJB), Microsoft (COM+ und .NET-*Assemblies*) sowie die eher generischen Standards der Object Management Group (OMG). Ebenfalls aus dem OMG-Umfeld stammt der gerade in der Verabschiedung befindliche UML2-Standard, welcher deutliche Neuerungen bei der Unterstützung der Modellierung von Komponenten und komponentenbasierten Softwaresystemen bietet (siehe z.B. [17]). Insbesondere die jetzt formale Definition der UML mit den Mitteln der Metamodellierungssprache *Meta Object Facility* (MOF) und dem Speicher- und Austauschformat *XML Metadata Interchange* (XMI) eröffnen neue Möglichkeiten zur modellzentrierten Komponentenentwicklung.
Weitere OMG-Standards zur Unterstützung der Entwicklung von Softwarekomponenten sind die Model-Driven Architecture (MDA) und die *Component Collaboration Architecture* (CCA) im Rahmen der *Enterprise Distributed Object Computing* (EDOC) Spezifikation. Um diese unterschiedlichen Technologien und Standards in einem Softwareentwicklungsprozess zu systematisieren, sind seit Ende der neunziger Jahre unterschiedliche Vorgehensmodelle zur komponentenorientierten Entwicklung von Softwaresystemen entstanden, die sich alle im Wesentlichen um die UML als Modellierungssprache gruppieren.

3 Vorgehensmodelle

Das zentrale Betrachtungsobjekt der meisten (komponentenorientierten) Vorgehensmodelle – oder auch Entwicklungsmethodiken – sind die durchzuführenden **Aktivitäten** (vgl. Abbildung 1). Diese werden in den meisten Fällen hierarchisch zu sachlogischen und/oder zeitlichen **Phasen** gebündelt. Überdies spezifizieren Vorgehensmodelle in der Regel Rollenmodelle, in denen Aktivitäten bestimmte **Rollen** zugeordnet werden. **Techniken**, wie z.B. die Use-Case-Modellierung in der UML, unterstützen die effiziente Manipulation unterschiedlicher **Ressourcen** und Artefakte im Entwicklungsprozess. Diese Ressourcen werden zumeist durch festgelegte **Notationen** beschrieben und erfüllen entsprechende **Richtlinien und Standards** – beispielsweise handelt es sich bei der UML um eine Standardnotation, welche nur ansatzweise methodische Richtlinien definiert.
Seit der Einführung der *Perspective*-Methode der Firma *Select* [1] findet

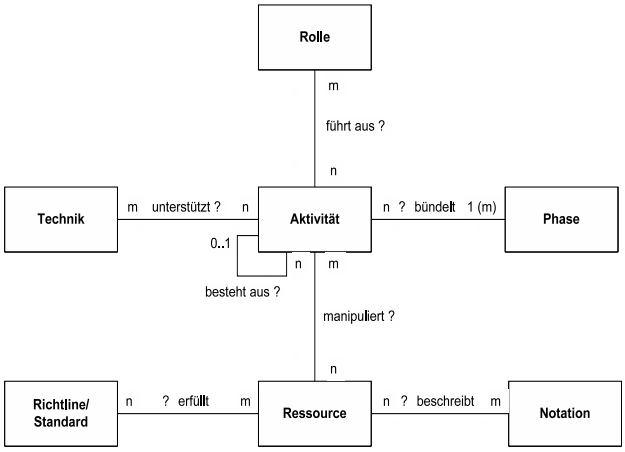

Abbildung 1: Grundlegende Elemente eines Vorgehensmodells

man in der Softwaretechnik-Literatur zahlreiche, ausgewiesen komponentenorientierte Vorgehensmodelle. Abbildung 2 zeigt die Entwicklung dieser Kategorie von Vorgehensmodellen in chronologischer Anordnung. Im Folgenden soll kurz auf die einzelnen Modelle eingegangen werden – speziell im Hinblick auf deren Beitrag zur Unterstützung der Entwicklung von einzelnen Software-Komponenten. Hierbei soll besonders die dabei vorgenommene Beschreibung von komponentenspezifischen Aktivitäten untersucht werden. Diese bestehen im Wesentlichen aus Komponentenidentifikation (Dekomposition der Anwendungsarchitektur), Komponentenspezifikation (z.B. Schnittstellen und Verhalten), Komponentensuche und -beschaffung, technischer und fachlicher Komponentenanpassung sowie der Komponentenintegration (auch Komposition oder Montage genannt). Diese Aufstellung orientiert sich u.a. an dem 1999 von Turowski erstmals vorgestellten „Komponentenlebenszyklus“ [20].

Wie in Abschnitt 2 bereits angesprochen, orientieren sich frühe Vorgehensmodelle für die komponentenbasierte Softwareentwicklung, wie das von Allen und Frost (1998, siehe [1]) beschriebene *Perspective*-Modell, an der Definition einer Komponente als „*executable unit of code*“ (S. 4). Komponenten werden hier, ausgehend von zusammenhängenden Diensten, durch eine Bottom-up-Vorgehensweise identifiziert. Unabhängig von der eigentlichen Anwendungsentwicklung (*Solution Process*) wird ein eigenes Prozessmodell für die Erstellung einzelner Komponenten vorgestellt (*Component Process*). Außerdem werden die Aktivitäten „Komponentensuche“ und „Komposition“ besonders detailliert beschrieben.

Der *Catalysis*-Ansatz von D'Souza und Wills hingegen wählt einen komplett anderen Ansatz zur Unterstützung der KBSE [6]. Anstatt Phasen und Ak-

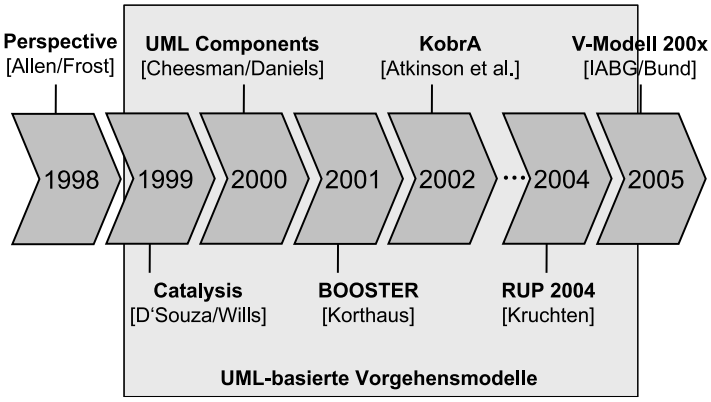

Abbildung 2: Komponentenorientierte Vorgehensmodelle im Zeitverlauf

tivitäten in einem Prozessmodell zu spezifizieren, werden lediglich „Prozessmuster“ (*Process Patterns*) vorgegeben, mit denen flexibel entscheidungsorientierte Entwicklungsprozesse konstruiert werden können. Anders als bei *Perspective* wird der Komponentenentwicklungsprozess nie völlig losgelöst von der Anwendungsentwicklung betrachtet. Komponenten werden als logisches Konzept sowohl auf der Geschäfts- oder Domänenebene, der Spezifikationsebene als auch der detaillierteren Entwurfsebene verwendet, was für die Nachvollziehbarkeit (*Traceability*) der Geschäftskomponentenentwicklung sorgt. Als Notation für die Komponentenmodellierung wird bei diesem Ansatz bereits die UML in teilweise proprietär erweiterter Form verwendet (siehe auch „*UML Components*“ [5]).
Aufbauend auf dem *Business Component*-Konzept von Herzum und Sims [9] fokussiert der BOOSTER-Ansatz von Korthaus, welcher im Rahmen einer Dissertation an der Universität Mannheim entstanden ist [14], speziell die Entwicklung betrieblicher Anwendungssysteme mittels Geschäftskomponenten. Der Komponentenbegriff wird hierbei zunächst in einem UML-basierten Metamodell bzw. UML-Profil (BOOSTER*Core) formalisiert und dem eigentlichen Vorgehensmodell (BOOSTER*Process) als „Modellierungsinstrumentarium“ und Konzeptionsrahmen beigefügt. BOOSTER*Core weist dabei signifikante Ähnlichkeiten mit der etwa zur gleichen Zeit entstandenen EDOC-Spezifikation der OMG auf.
Ebenfalls im universitären Umfeld, im Rahmen eines Forschungsprojekts, entstand die von Atkinson u.a. entwickelte KobrA-Methodik zur komponentenbasierten, modellzentrierten Anwendungsentwicklung [2]. Wie eingangs erwähnt, stehen bei diesem Vorgehensmodell in erster Linie die hierarchische und rekursive Spezifikation und die MDA-basierte Realisierung von Komponentenarchitekturen in Form von „Komponentenbäumen“ im Vorder-

grund. Aufbauend auf dem *KobrA Component Model* werden hauptsächlich Methoden und Techniken zur UML-basierten Komponentenspezifikation, -anpassung und -komposition detailliert beschrieben.
In der aktuellen Version des Rational Unified Process (RUP 2004, vgl. [15]) finden sich, nicht zuletzt unterstützt durch die komponentenorientierten Erweiterungen der UML, zunehmend Techniken, Richtlinien und Standards zur Entwicklung von und mit Softwarekomponenten. Auch die für 2005 geplante Neuauflage des V-Modells der Industrie- und Anlagen Baugesellschaft (IABG)[1] soll vermehrt Hilfestellung bei der Erstellung qualitativ hochwertiger Komponenten bieten.

4 Entwicklungswerkzeuge

Eine wichtige Neuerung in der UML2-Spezifikation sind die so genannten Kompositionsstruktur- oder auch Architekturdiagramme [11]. Sie ermöglichen in erster Linie eine detailliertere Modellierung der inneren Struktur von Klassen und Komponenten [18]. Die Veränderungen bei Sequenz- und Zustandsdiagrammen erlauben eine detailliertere Beschreibung des dynamischen Systemverhaltens. Dadurch eignet sich die aktuelle Fassung der UML besser als ihre Vorgänger für Codegenerierungsansätze. Teilweise wurden die Neuerungen der UML 2 bereits von den Herstellern entsprechender Modellierungs- und Entwicklungswerkzeuge, z.B. Gentlewares Poseidon[2], aufgegriffen. Im Folgenden werden einige Werkzeuge kurz vorgestellt, die im Rahmen eines komponentenbasierten und modellzentrischen Entwicklungsansatzes Verwendung finden können.
Die zuvor beschriebenen Methoden und Techniken zur Entwicklung von und mit Softwarekomponenten in den oben genannten Vorgehensmodellen werden durch eine Vielzahl von Werkzeugen unterstützt. Diese lassen sich unterscheiden in solche, die besonders zur Unterstützung bestimmter Vorgehensmodelle geeignet sind, z.B. IBM/Rational XDE zur Unterstützung eines am Rational Unified Process orientierten Entwicklungsprozesses oder die *Select Business Solutions* zur Unterstützung der *Perspective*-Methodik, und in solche, die unabhängig vom Einsatz eines bestimmten Vorgehensmodells sind.
Interessante Entwicklungen sind vor allem im Umfeld der quelloffenen Entwicklungsumgebung Eclipse[3] zu beobachten. Als Beispiele seien hier die Werk-

[1] `http://www.v-modell.iabg.de/`
[2] `http://www.gentleware.de/`
[3] `http://www.eclipse.org`

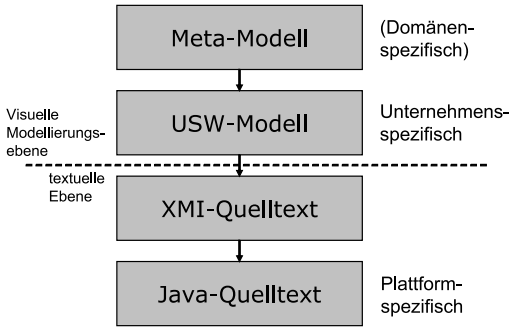

Abbildung 3: Modellzentrierte Anwendungsentwicklung mit OMEGA

zeuge *EclipseJ2EE*[4] oder auch JBoss-IDE[5] zur Entwicklung von EJB-Komponenten genannt. Darüber hinaus existieren zahlreiche Projekte, die die Realisierung modellzentrischer Entwicklungsansätze zum Ziel haben. Eine umfangreiche Übersicht über entsprechende Plugins findet sich auf einschlägigen Eclipse-Plugin-Märkten[6]. Unabhängig von der gewählten Entwicklungsumgebung arbeitet das Codegenerator-Framework AndroMDA[7], das die Generierung von Programmcode aus UML-Modellen ermöglicht. Ein Überblick über frei verfügbare und kommerzielle Werkzeuge zur Codegenerierung findet sich auf den Webseiten des *Code Generation Network*[8].

5 Aktuelle Forschungsansätze

Im bisher vorgestellten Kontext von speziellen KBSE-Vorgehensmodellen und -Entwicklungswerkzeugen finden an der Universität Mannheim momentan einige aktuelle Forschungsarbeiten und -projekte statt. Mit dem Booster-Ansatz [14] wurde bereits eine grundlegende Forschungsrichtung erwähnt, welche in andere Vorhaben und Projekte einfließen und weiterentwickelt werden soll. Erste Ansätze, Booster weiter in Richtung eines modellzentrierten, kollaborativen Ansatzes zur Entwicklung von Unternehmenssoftware zu entwickeln, werden von Hildenbrand und Korthaus (2004) vorgestellt [10].
Mit der *Ontological Metamodel Extension for Generative Architectures* (OMEGA) wird momentan im Rahmen einer Dissertation im Bereich der (meta-)modellbasierten Entwicklung von Anwendungssoftware, insbesondere Web-Applikationen, aktiv geforscht [8]. Ziel dieser Arbeit ist die automatische

[4] `http://www.eclipsej2ee.com/`
[5] `http://www.jboss.org/products/jbosside`
[6] z.B. unter `http://eclipse-plugins.2y.net/`
[7] `http://www.andromda.org/`
[8] `http://www.codegeneration.net/`

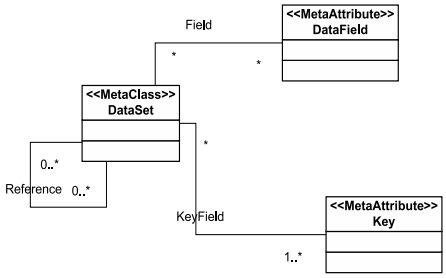

Abbildung 4:
Abstraktes Metamodell

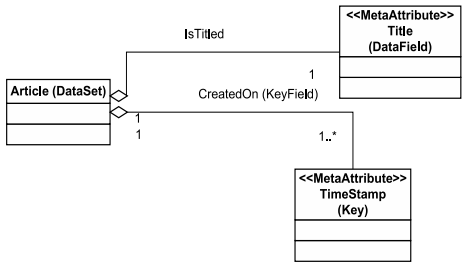

Abbildung 5:
Domänenspezifische Instanziierung

Quelltextgenerierung aus MOF/XMI-Modellinformationen, welche formal in mehreren Metamodellebenen (Metamodellhierarchien) spezifiziert werden [7]. In Verbindung mit einem komponentenbasierten Ansatz ergeben sich durch die Modellierung ontologischer Metamodellebenen die Möglichkeit, domänenspezifische Modellbausteine auf unterschiedlichen Abstraktionsniveaus wieder zu verwenden. Abbildung 3 zeigt den schematischen Ablauf eines OMEGA-Entwicklungsprozesses.
Ähnlich wie der MDA-Ansatz der OMG arbeitet OMEGA mit der systematischen Stratifikation von Modellebenen. Nachfolgend soll der Entwicklungsprozess aus Abbildung 3 anhand eines konkreten Beispiels verdeutlicht werden.
In diesem Fall soll eine Web-Applikation zur Verwaltung eines Gebrauchtwagenmarktes schrittweise anhand von unterschiedlichen Metamodellebenen entwickelt werden. Die Abbildungen 4 und 5 zeigen noch recht abstrakte Metamodelle, wobei die Metaklasse `DataSet` durch die Klasse `Article` instanziiert wird. Abbildung 6 stellt die komplette Architektur der anvisierten Web-Applikation, welche durch schrittweise Instanziierung der einzelnen Modellkonstrukte entwickelt wurde, als grafisches Modell dar.
Diese Modelldaten lassen sich dann mit einem Werkzeug, welches Modelldaten MOF-konform speichert, in das standardisierte XMI-Format umwandeln, und somit werkzeugunabhängig austauschen und weiterverarbeiten. Zur Verwaltung der bisher grafisch dargestellten (Meta-)Modelldaten wird bei OMEGA das im NetBeans-Kontext entstandene *Metadata Repository* (MDR)[9] eingesetzt. Der folgende Ausschnitt aus dem Abbildung 6 zugrunde liegenden Modell im XMI-Format soll verdeutlichen, wie die Modellelemente eindeutig spezifiziert werden können:

```
<model>
```

[9]http://mdr.netbeans.org/

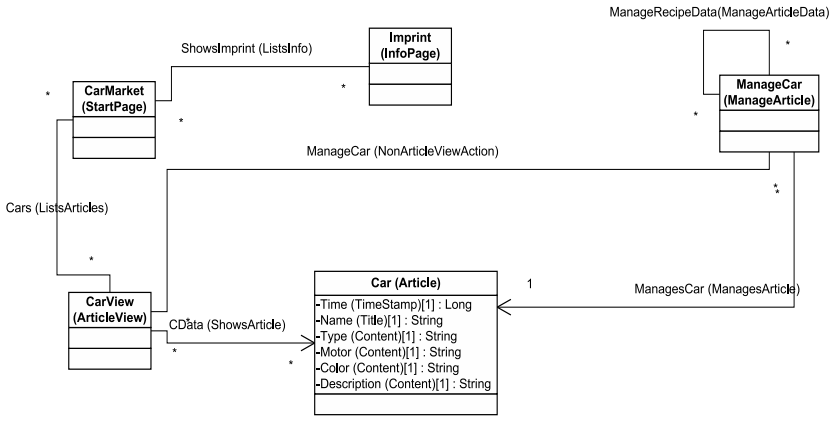

Abbildung 6: Unternehmensspezifisches Modell einer Web-Applikation

```
...
  <Class name="Car">
    <MetaClass name="Article" />
    <MetaMetaClass name="DataSet" />
 <Attribute name="Time">
      <MetaAttribute name="TimeStamp" />
      <MetaMetaAttribute name="Key" />
   <multiplicity>
        <lowerBound>1</lowerBound>
        <upperBound>1</upperBound>
        <isOrdered>false</isOrdered>
        <isUnique>false</isUnique>
      </multiplicity>
      <scope>instance_level</scope>
      <type>Long</type>
      <visibility>private</visibility>
      <isChangeable>false</isChangeable>
    </Attribute>
 <Attribute name="Name">
      <MetaAttribute name="Title" />
      <MetaMetaAttribute name="DataField" />
      ...
</model>
```

In der bisherigen prototypischen Implementierung der OMEGA-Transformationswerkzeuge wird das Modell mittels XSLT-Templates in kompilierbaren Java-Quelltext umgewandelt. Das nachfolgende Code-Fragment verdeutlicht,

wie der Artikel „Car“ als EJB-`EntityBean` realisiert wird:

```
public abstract class CarBean implements EntityBean {
    protected EntityContext _entityContext;
      ...
      public abstract String getName();
      public abstract void setName(String Name);

      public abstract String getType();
      public abstract void setType(String Type);

      public abstract String getMotor();
      public abstract void setMotor(String Motor); }
```

Da es sich beim gezeigten Beispiel um eine sehr einfache Anwendung handelt, kommt eine entscheidende Hürde bei der modellbasierten Entwicklung von Unternehmenssoftware nicht zum Tragen: die Modellierung von Dynamik und Verhalten. Zu diesem Zweck wird momentan die Tauglichkeit von erweiterten Aktivitäts- und Zustandsdiagrammen geprüft und entsprechende Transformationswerkzeuge entwickelt.
Thematisch eng vermascht mit den beiden Forschungsvorhaben BOOSTER und OMEGA startete am 1. Juli 2004 das Forschungsprojekt „CollaBaWü“. Es geht hierbei um die kollaborative, komponentenbasierte Entwicklung von Unternehmenssoftware in der Finanzdienstleistungsdomäne Baden-Württembergs[10]. An dem Projekt sind daher namhafte Banken und deren IT-Tochterunternehmen und -Zulieferer beteiligt, um einen regen Austausch von theoretischem und Praxis-Know-how zu gewährleisten. Das Vorhaben untergliedert sich in vier Teilprojekte mit jeweils unterschiedlichen aber komplementären Zielsetzungen. Angestrebt wird

- die grundlegende Erforschung von Methoden, Prozessen und Werkzeugen zur kollaborativen, zwischenbetrieblichen Erstellung von Unternehmenssoftware,
- das Sammeln und Entwickeln geeigneter Techniken und Werkzeuge zur modellzentrierten Erstellung von komponentenbasierten betrieblichen Anwendungssystemen,
- die Schaffung einer Domänen-Ontologie für den Finanzdienstleistungs-/Banken-Bereich in Verbindung mit einer entsprechenden Komponenten-Beschreibungssprache und

[10] `http://www.collabawue.de/`

- die betriebswirtschaftliche, ökonomische und juristische Evaluation von Geschäftsmodellen für die kollaborative Erstellung von Unternehmenssoftware aus Softwarekomponenten.

Innerhalb der Projektlaufzeit von 3 Jahren sollen u.a. ein Referenzmodell zur kollaborativen Entwicklung von Softwarekomponenten, entsprechende prototypische Realisierungen der Werkzeugunterstützung und erste Beispielkomponenten in Zusammenarbeit mit den Praxispartnern entstehen.

6 Schlussfolgerung

Die KBSE bildet derzeit ein aktives und dynamisches Forschungsfeld in der Softwaretechnik und es herrscht eine klar erkennbare Tendenz hin zur modellgetriebenen Entwicklung von Softwarekomponenten. Zwar wird diese Entwicklung mittlerweile immer besser durch entsprechende Standards wie UML2, MOF und XMI unterstützt, doch fehlt bisher in den meisten Fällen immer noch eine angemessene Unterstützung durch integrierte Werkzeuge – vor allem zur Anwendung komplexer Entwicklungsmethodiken und Vorgehensmodelle.
Selbst wenn die technische Umsetzung der Visionen aus der aktuellen MDD- und KBSE-Forschung erreicht werden kann, besteht noch ein beträchtlicher Bedarf an Adoptionsforschung, um die Diffusion dieser neuen Methoden und Technologien in die Praxiswelt zu gewährleisten. Ein kritischer Punkt hierbei sind die unterschiedliche Annahmen über grundlegende Konzepte, wie z.B. „Komponente“ oder auch „Software-Modell“, in Forschung und Praxis. Weitere offene Fragestellungen, welche die Forschung in Zukunft beantworten möchte, sind u.a.:

- Welche Vorteile hat Komponentensoftware gegenüber Individualsoftware bzw. reiner Standardsoftware aus Anwendersicht?
- Inwieweit sind Anwenderunternehmen aus dem Finanzdienstleistungsbereich überhaupt bereit, Wissen in Form von Komponenten und anderen Softwareartefakten auszutauschen?
- Werden komponentenbasierte und modellzentrierte Ansätze in Zukunft weiter konvergieren oder konkurrieren?
- Wie kann ein Zielkonflikt zwischen individualisierter USW-Lösung und maximaler Wiederverwendung von Komponenten aufgelöst werden?
- Ist bei den Softwareanbietern bereits eine aufkommende Wiederverwendungskultur feststellbar?

Die Entwicklung von und mit Softwarekomponenten hat vor allem für IT-Dienstleister bestimmter Branchen, wie beispielsweise Finanzdienstleister und Banken, den Vorteil, dass sie Komponenten (z.B. zur Transaktionsüberwachung und Betrugsbekämpfung) entwickeln und diese dann auch an andere Kunden als den eigentlichen Auftraggeber verkauft werden können. Überdies bergen solche Softwarekomponenten im Vergleich zu rein monolithischen Anwendungsblöcken für die Kunden den Vorteil der besseren Wartbarkeit und Aktualisierbarkeit trotz immer komplexerer Systemlandschaften und Unternehmenssoftwaresysteme.
In der Praxis sind zuweilen zwar erste Ansätze zur Wiederverwendung von Softwareartefakten vorhanden, von einer wirklichen „Wiederverwendungskultur“ kann man aber in den meisten Fällen noch nicht sprechen. Dafür sind die Rahmenbedingungen von Softwareprojekten momentan auch noch gar nicht ausgelegt, da beispielsweise keine entsprechenden Anreizsysteme für die Schaffung von Wiederverwendbarkeit vorhanden sind.

Literatur

[1] P.R. Allen and S. Frost.
Component-Based Development for Enterprise Systems : Applying the Select Perspective.
Cambridge University Press, 1998.

[2] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel.
Component Based Product Line Engineering with UML.
Component Software Series. Addison-Wesley, 2002.

[3] H. Balzert.
Lehrbuch der Software-Technik – Software-Entwicklung, volume 1.
Spektrum, 2000.

[4] T.J. Biggerstaff and C. Richter.
Reusability framework, assessment, and directions.
In T.J. Biggerstaff and A.J. Perlis, editors, *Software Reusability: Vol. 1, Concepts and Models*, pages 1–17. ACM Press, 1989.

[5] J. Cheesman and J. Daniels.
UML Components : A Simple Process for Specifying Component-Based Software.
Component Software Series. Addison-Wesley, 2001.

[6] D.F. D'Souza and A.C. Wills.
Objects, Components, and Frameworks with UML : The Catalysis Approach.
Object Technology. Addison-Wesley, 1999.

[7] R. Gitzel and T. Hildenbrand.
A Taxonomy of Metamodel Hierarchies.
März 2004.

[8] R. Gitzel, I. Ott, and M. Schader.
Ontological Metamodel Extension for Generative Architectures (OMEGA).
Arbeitspapier 1, Universität Mannheim, 2004.

[9] P. Herzum and O. Sims.
Business Component Factory – A Comprehensive Overview of Component-Based Development for the Enterprise.
Wiley, 1999.

[10] T. Hildenbrand and A. Korthaus.
A model-driven approach to business software engineering.
In N. Callaos, editor, *Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics*, volume 4 of *Information Systems, Technologies and Applications*, pages 74–79. International Institue of Informatics and Systemics (IIIS), Juli 2004.

[11] M. Jeckle, C. Rupp, J. Hahn, B. Zengler, and S. Queins.
UML 2.0: Evolution oder Degeneration.
OBJEKTspektrum, (3):12–20, Juni/Juli 2004.

[12] T. Kühne.
Automatisierte softwareentwicklung mit modellcompilern.
Thema Forschung, 1, 2003.

[13] D.E. Knuth.
Computer programming as an art.
Communications of the ACM, 17(12):667–673, Dezember 1974.

[14] A. Korthaus.
Komponentenbasierte Entwicklung computergestützter betrieblicher Informationssysteme, volume 20 of *Informationstechnologie und Ökonomie.*
Peter Lang, 2001.

[15] P.B. Kruchten.
The Rational Unified Process : An Introduction.
Object Technology Series. Addison-Wesley, 2004.

[16] D.G. Messerschmitt and C. Szyperski.
Software Ecosystem: Understanding an Indispensable Technology and Industry.
MIT Press, September 2003.

[17] J. Rumbaugh, I. Jacobson, and G. Booch.
The Unified Modeling Language Reference Manual.
Object Technology Series. Addison-Wesley, Juni 2004.

[18] C. Schröder and T. Weilkiens.
Projekterfahrungen mit der UML 2.0.
OBJEKTspektrum, (3):23–28, Mai 2004.

[19] C. Szyperski, D.W. Gruntz, and S. Murer.
Component Software : Beyond Object-Oriented Programming.
Component Software Series. Addison-Wesley, 2002.

[20] K. Turowski.
Ordnungsrahmen für komponentenbasierte betriebliche anwendungssysteme.
In K. Turowski, editor, *Tagungsband des 1. Workshops komponentenorientierte betriebliche Anwendungssysteme (WKBA 1)*, pages 3 – 14, 1999.

[21] K. Turowski, editor.
Vereinheitlichte Spezifikation von Fachkomponenten. Deutsche Gesellschaft für Informatik, 2002.