February 2005

Flexible Generierung neuer Geschäftsprozesse am Beispiel der ShopLab Toolbox

Karim Khakzar
Fachhhochschule Fulda

Joachim Salmann
Fachhhochschule Fulda

Thomas Berger
Institut für interdisziplinäre Forschung e.V.

Thomas Jöckel
Institut für interdisziplinäre Forschung e.V.

Hans-Martin Pohl
Institut für digitale Medien und Kommunikation GmbH

See next page for additional authors

Authors
Karim Khakzar, Joachim Salmann, Thomas Berger, Thomas Jöckel, Hans-Martin Pohl, and Wolfgang Frank

In: Ferstl, Otto K, u.a. (Hg) 2005. *Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety*; 7. Internationale Tagung Wirtschaftsinformatik 2005. Heidelberg: Physica-Verlag

ISBN: 3-7908-1574-8

Flexible Generierung neuer Geschäftsprozesse am Beispiel der ShopLab Toolbox

Karim Khakzar, Joachim Salmann
Fachhhochschule Fulda

Thomas Berger, Thomas Jöckel
Institut für interdisziplinäre Forschung e.V.

Hans-Martin Pohl, Wolfgang Frank
Institut für digitale Medien und Kommunikation GmbH

Zusammenfassung: In der Softwareentwicklung ist es heute von immer größerer Bedeutung, zeitnah und flexibel auf neue Anforderungen reagieren zu können. Neue Geschäftsprozesse müssen schnell und kostengünstig in der Software abgebildet werden. Für die ShopLab Toolbox, einer Plattform für kleine und mittlere Unternehmen für den Einsatz innovativer Einkaufsumgebungen, wurde aus diesem Grund ein graphisches Werkzeug entwickelt, mit dem es möglich ist, neue Geschäftsprozesse für die Toolbox schnell und einfach zu erzeugen. Bei der Generierung der Geschäftsprozesse kommen Entwurfsmuster zum Einsatz.

Schlüsselworte: Geschäftsprozesse, Entwurfsmuster

1 Die Bedeutung der flexiblen Generierung von Geschäftsprozessen

Eine der wichtigsten Aufgaben in der Softwareentwicklung ist die softwaretechnische Abbildung von Geschäftsprozessen. Geschäftsprozesse beschreiben geschäftliche Vorgänge zwischen Unternehmen bzw. innerhalb einer Unternehmung.

Geschäfte sind jedoch niemals statisch, sondern immer in Bewegung. Die Produkte, Prozesse und Ziele einer Unternehmung verändern sich im Laufe der Zeit. Auch unterscheiden sich die Geschäftsprozesse einzelner Unternehmen häufig stark von einander. Gerade kleine und mittelständische Unternehmen besitzen ganz eigene Geschäftsprozesse, die in der Software abgebildet werden müssen.

Aus diesem Grund ist es wichtig, dass eine Geschäftsanwendung, die ein Geschäft modelliert, flexibel ist und sich an verändernde Vorgaben und Begebenheiten an-

passen kann. Diese Flexibilität kann erreicht werden, indem man die Software-Teile, die ein Geschäft modellieren, in Business-Objects kapselt. Auf diese Weise kann eine Software flexibel, erweiterbar und wieder verwendbar werden und sich mit dem Geschäft selbst fortentwickeln. Eine Kapselung der Geschäftsprozesse in Business-Objects erzeugt jedoch nicht automatisch eine gute Geschäftsanwendung. Trotz moderner Systeme wie der J2EE, die den Einsatz von Business-Objects unterstützen, bleibt der Entwurf effizienter, skalierbarer, flexibler und wartbarer Anwendungen eine schwierige und komplexe Aufgabe. Um diese Aufgabe zu meistern, werden in der Softwareentwicklung Entwurfsmuster eingesetzt.

2 Das ShopLab Konzept

Das ShopLab Projekt (www.shoplab.info) hat sich zum Ziel gesetzt, unter Einsatz innovativer und multimedialer Technologien, die Vorteile der realen Einkaufswelt mit denen des virtuellen Einkaufs in so genannten hybriden Einkaufsumgebungen zu vereinen [Kha+03, S. 37ff]. Diese hybriden Einkaufsumgebungen sind für den Einsatz im innerstädtischen Einzelhandel konzipiert.

Das ShopLab Projekt wurde im September 2001 gestartet und wird im Rahmen des EU-Forschungsprogramms „Benutzerfreundliche Informationsgesellschaft (IST)“ gefördert. An dem Projekt ist ein Konsortium von insgesamt acht Partnern aus fünf europäischen Ländern unter Leitung der Fachhochschule Fulda beteiligt. Hauptziel des Projektes ist es zu zeigen, dass sich durch die Verschmelzung von realen und virtuellen Einkaufsumgebungen zu sogenannten „hybriden Shops“ Wettbewerbsvorteile erzielen lassen. Hierzu werden multimediale Schnittstellen entwickelt, die dem Kunden im Geschäft einen zusätzlichen Nutzen bieten und eine wirkliche Kaufentscheidungshilfe darstellen sollen [Boo+03, S. 606ff]. Dabei konzentriert sich das Projekt auf kleine und mittlere, traditionelle Einzelhandelsunternehmen, die ein individuelles Angebot und Sortiment besitzen und für die Kundennähe und persönliche Beratung im Vordergrund stehen. Da kleinere und mittlere Einzelhandelsunternehmen in der Regel keine großen Investitionen für Multimedia-Technologie in den Verkaufsräumen tätigen können, ist es wichtig, eine kostengünstige Lösung anzubieten, deren Konfiguration und Wartung mit vertretbarem Aufwand bewerkstelligt werden kann.

Die im Projekt entwickelte „ShopLab Toolbox“ besteht aus einer System-Plattform, der sogenannten „ShopLab Core Platform“, und verschiedenen individuellen Modulen, die an diese angeschlossen werden können. Um jedem Geschäft eine individuelle Prägung zu geben, soll die Toolbox möglichst einfach an unterschiedliche Geschäftstypen angepasst werden können. Im Rahmen des Projekts wurden deshalb zwei unterschiedliche, ganz auf die Bedürfnisse des jeweiligen Ladens angepasste Prototypen entwickelt. Für ein Geschäft für Maßhemden wurde ein „Interaktiver Spiegel“ und für ein Sportgeschäft ein „Interaktives Regal“ her-

gestellt [Kha+03, S. 37ff]. Der „Interaktive Spiegel“ ermöglicht es dem Benutzer, Kleidungsstücke virtuell anzuprobieren, ohne sie wirklich anziehen zu müssen. Er erleichtert somit dem Kunden die Kaufentscheidung gerade auch in Fällen, in denen keine Anprobe möglich ist. Das „Interaktive Regal“ unterstützt den Verkäufer von Sportschuhen während des Verkaufsprozesses. Ein Tool zum Produktvergleich, die Präsentation von Zusatzinformationen und Animationen erleichtern dem Kunden die Kaufentscheidung.

3 Die Architektur der ShopLab Toolbox

Bereits diese beiden Prototypen stellen sehr unterschiedliche Anforderungen an die zugrunde liegende Systemplattform. Die Plattform soll jedoch nicht nur in der Lage sein, mit diesen beiden Prototypen zusammenzuarbeiten, sondern soll in Zukunft auch ohne großen Aufwand völlig andersartige Module unterstützen. Aus diesem Grund wurde eine sehr flexible und modulare System-Architektur gewählt.

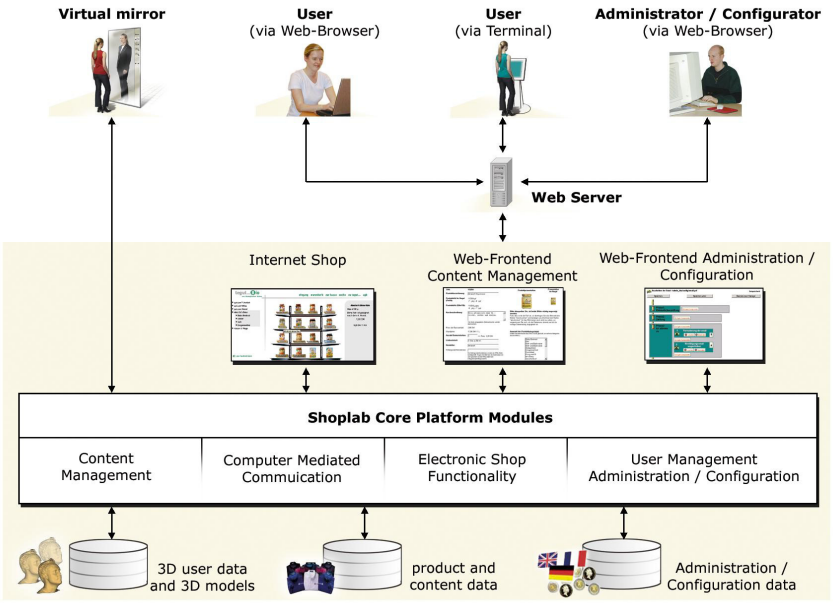

Abbildung 1: Die Architektur der „ShopLab Toolbox“

Die „ShopLab Toolbox“ besitzt eine typische 3-Schichten-Architektur. Diese besteht aus der Datenbankschicht, der Funktions- und Logikschicht und der Präsentationsschicht, die die Schnittstelle zum Kunden bzw. Administrator bereitstellt.

Abbildung 1 stellt die Systemarchitektur des ShopLab-Systems an Hand einiger Beispielmodule dar.

Die zugrunde liegende System-Plattform ist die „ShopLab Core Platform“. Sie basiert auf der J2EE-Architektur und besteht aus einem modularen und flexiblen Baukastensystem mit einzelnen Software-Modulen, z.B. für die Konfiguration und Administration, zur Speicherung multimedialer Inhalte, zur Unterstützung von e-Shopping-Funktionen und zur Anbindung an bestehende ERP-Systeme. Innerhalb dieser einzelnen Module wiederum werden durch die Zusammenarbeit verschiedenster Business-Objects die einzelnen Geschäftsprozesse abgebildet. Die Geschäftslogik wird dabei mit Enterprise JavaBeans (EJB 2.0) realisiert. Nach der Definition des Sun Java Centers (SJC) implementieren Business-Objects der J2EE sowohl die Geschäftsvorgänge als auch die Geschäftsdaten [Sun02]. Da die Geschäftsvorgänge in der J2EE von Session-Beans und die Geschäftsdaten von Entity-Beans implementiert werden, können somit sowohl Session- als auch Entity-Beans als Business-Objects bezeichnet werden. In der J2EE und damit auch in der „ShopLab Core Platform“ werden Geschäftsprozesse somit durch eine Zusammenarbeit von Session-Beans und Entity-Beans abgebildet, die als Business-Objects bezeichnet werden. Die Geschäftsmethoden befinden sich aber in der Regel in den Session-Beans.

Die Kommunikation zwischen den einzelnen Modulen der „ShopLab Core Platform“ findet über XML statt. Durch die Verwendung dieser standardisierten Beschreibungssprache und dem modularen Ansatz können Änderungen am System schnell und mit einem überschaubaren Aufwand realisiert werden. Neue multimediale und multisensorische Komponenten lassen sich ohne große Probleme einfügen, ohne dass ein komplettes Re-Design notwendig wäre. Beim Entwurf wurde der flexiblen und zukunftssicheren Anbindung von multimedialen Datenbanken besondere Aufmerksamkeit geschenkt. Über einen sogenannten Datenbankkonnektor mit standardisierten Schnittstellen lassen sich prinzipiell alle Arten von relationalen Datenbanken anschließen.

4 Kopplung existierender Business-Objects

Um die komplexen Geschäftsvorgänge oder Anwendungsfälle (Use Cases) innerhalb der „ShopLab Core Platform“ abbilden zu können, ist eine Zusammenarbeit bzw. eine Kopplung von mehreren Business-Objects nötig. Trotz des modularen und flexiblen Aufbaus der Plattform ist für die Implementierung einer solchen Kopplung immer noch Programmieraufwand nötig. Dieser Prozess soll jedoch weiter vereinfacht werden.

Da für die verschiedenen Einzelhändler immer neue Geschäftsvorfälle aus den Business-Objects der „ShopLab Core Platform“ zusammengesetzt werden müssen,

und dies möglichst schnell und einfach ohne großen Programmieraufwand geschehen soll, bietet sich ein Tool mit einer graphischen Oberfläche für diese Aufgabe an.

An das Tool stellen sich aufgrund der speziellen Architektur der ShopLab Toolbox die im Folgenden beschriebenen Anforderungen.

4.1 Vorgaben

Geschäftsprozesse bzw. Geschäftsvorfälle innerhalb der „ShopLab Core Platform“ setzen sich häufig aus mehreren Einzelvorgängen zusammen. Ein Geschäftsvorfall „Zahlungsvorgang“ kann sich z.B. aus den Einzelvorgängen „Ermittle Rechnungsbetrag“, „Prüfe Kundendaten“ und „Belaste Konto“ zusammensetzen. Diese Einzelvorgänge werden häufig in einzelnen Business-Objects gekapselt. Methoden dieser Business-Objects repräsentieren dann die Geschäftsvorfälle. Mit Hilfe des zu erstellenden graphischen Tools soll es möglich sein, diese Methoden der Business-Objects so zu koppeln, dass sie einen übergeordneten Geschäftsvorfall modellieren. Dabei wird gefordert, dass die einzelnen Methoden-Aufrufe der Business-Objects in beliebiger Reihenfolge angeordnet werden können und austauschbar sind.

Durch die Kopplung der einzelnen Business-Objects soll es ermöglicht werden, dass eine Anwendung, die einen Geschäftsvorfall nutzen will, diesen durch den Aufruf einer einzelnen übergeordneten Methode ausführen kann und sich nicht mehr um die darunterliegenden Business-Objects kümmern muss.

Des Weiteren soll es möglich sein, mehrere Aufrufe von Business-Objects in einer Transaktion zusammenzufassen. So soll z.B. ein Geschäftsvorfall „Zahlungsvorgang“ immer vollständig ausgeführt werden. Dies setzt aber voraus, dass alle daran beteiligten Business-Objects ihre Aufgaben vollständig und erfolgreich beenden. Tritt ein Fehler in einem der Business-Objects auf, so müssen alle bereits ausgeführten Aufgaben der anderen Business-Objects rückgängig gemacht werden. Dies kann garantiert werden, wenn all diese Aufgaben innerhalb einer Transaktion ablaufen. Daraus folgt, dass die Methode, die den übergeordneten Geschäftsvorfall repräsentiert, eine neue Transaktion starten muss, der alle Methodenaufrufe der daran beteiligten Business-Objects angehören.

Die Anwendung soll Software-Komponenten generieren, die innerhalb der „ShopLab Core Platform“ eingesetzt werden können. Deshalb müssen die zu erzeugenden Komponenten an die Besonderheiten der Plattform angepasst sein.

Eine weitere wichtige Vorgabe ist, dass die Business-Objects bei der Kopplung nicht verändert werden dürfen. Das bedeutet, wenn die Business-Objects zu einem übergeordneten Geschäftsvorfall zusammengesetzt werden, soll ihr Quellcode unverändert bleiben. Es dürfen daher keine Änderungen an ihrem Quellcode not-

wendig sein. Dies ist wichtig, da sie eventuell noch an anderen Stellen verwendet werden.

Das graphische Tool, von nun an „ShopLab Editor“ genannt, soll nach erfolgreicher Kopplung der einzelnen Business-Objects den daraus resultierenden Quellcode und alle anderen benötigten Dateien automatisch erzeugen.

Um die Business-Objects auf möglichst sinnvolle und effiziente Art und Weise zu koppeln, sollen Entwurfsmuster eingesetzt werden.

Aufgrund der sehr spezifischen Anforderungen der ShopLab Toolbox an die zu generierenden Klassen und der Vorgabe, die existierenden Business-Objects mit Hilfe von Entwurfsmustern zu koppeln, sowie dem beschränktem Budget im ShopLab Projekt, konnten existierende CASE-Tools nicht verwendet werden. Stattdessen wurde mit dem „ShopLab Editor“ eine eigene Lösung erstellt.

4.2 Auswahl der geeigneten Entwurfsmuster

Für die Kopplung der Business-Objects wurden die folgenden Entwurfsmuster auf ihre Eignung innerhalb der „ShopLab Toolbox“ untersucht:

1. Facade-Muster
2. Befehls-Muster
3. Vermittler-Muster
4. Session-Facade-Muster
5. EJB-Command-Muster

Bei den ersten drei Entwurfsmustern handelt es ich um klassiche Entwurfsmuster der „Gang of Four“ [Gam+96]. Die letzten beiden Muster sind spezielle Entwurfsmuster für die J2EE, die auf deren Anforderungen zugeschnitten sind.

Jedes dieser Muster beschreibt einen unterschiedlichen Ansatz, wie die Zusammenarbeit von Objekten bzw. Business-Objects realisiert werden kann. In den folgenden Absätzen werden die einzelnen Muster kurz beschrieben. Die detaillierte Beschreibung und Funktionsweise der einzelnen Entwurfsmuster kann in der Literatur nachgelesen werden [Gam+96; Bien02].

Das Facade-Muster bietet eine einheitliche Schnittstelle zu einer Menge von Schnittstellen eines Subsystems. Die Fassadenklasse definiert eine abstrakte Schnittstelle, welche die Benutzung des Subsystems vereinfacht. So kann eine Fassade z.B. verwendet werden, um mehrere Methodenaufrufe von Objekten des Subsystems hinter einem Methodenaufruf der Fassade zu verbergen.

Das Befehls-Muster kapselt einen Befehl als ein Objekt. Durch seinen Einsatz wird es ermöglicht, Anfragen an unbekannte Anwendungsobjekte zu richten, in-

dem die Anfrage selbst zum Objekt gemacht wird. Es erleichtert außerdem, Operationen in eine Ausführungskette zu stellen.

Das Vermittler-Muster definiert ein Objekt, welches das Zusammenspiel einer Menge von Objekten in sich kapselt. Indem das Vermittler-Muster verhindert, dass Objekte explizit aufeinander Bezug nehmen, fördert es lose Kopplung. Vermittler ermöglichen es, das Zusammenspiel von Objekten zu variieren, ohne die Objekte selbst verändern zu müssen.

Beim Session-Facade-Muster wird eine Session-Bean als Fassade benutzt, die die Komplexität der Interaktionen zwischen den Business-Objects verbirgt. Die Session Facade verwaltet die Geschäftsobjekte und bietet eine einheitliche Schnittstelle für die Clients. In ihrer Funktionalität entspricht sie dem klassischen Fassade-Muster der „Gang of Four".

Das EJB-Command-Muster kapselt die Geschäftslogik in kleinen Befehls-Beans. Wird das Befehls-Muster auf EJBs angewandt, so erzielt es vergleichbare Resultate wie das Session-Facade-Muster, mit dem Unterschied, dass keine schwergewichtigen und unflexiblen Komponenten entstehen, wie beispielsweise Session-Beans.

Der Einsatz jedes dieser Muster bringt bestimmte Vor- bzw. Nachteile mit sich. Deshalb kann nicht pauschal entschieden werden, welches der beschriebenen Muster die beste Lösung zur Kopplung von Business-Objects bietet. Diese Entscheidung hängt von dem Kontext ab, in dem das Entwurfsmuster eingesetzt werden soll.

Aus diesem Grund wurden die oben genannten Entwurfsmuster auf ihre Verwendbarkeit im speziellen Kontext der „ShopLab Core Platform" untersucht.

Das Vermittler-Muster konnte schnell ausgeschlossen werden, da beim Vermittler-Muster den Business-Objects eine Referenz auf das Vermittler-Object übergeben werden muss. Eine solche Referenzübergabe ist in den bereits existierenden Business-Objects aber nicht vorgesehen. Um sie zu verwirklichen, wären Änderungen der existierenden Business-Objects notwendig gewesen. Solche Änderungen widersprechen aber den Vorgaben und sind somit unzulässig.

Zur Kopplung der existierenden Business-Objects blieben also noch das Befehls-Muster bzw. EJB-Command-Muster und das Fassade-Muster bzw. Session-Facade-Muster. Da die „ShopLab Core Platform" auf der J2EE-Technologie basiert, sollten von diesen vier möglichen Mustern die Muster verwendet werden, die bereits für einen Einsatz in der J2EE-Plattform optimiert sind, also das EJB-Command-Muster oder das Session-Facade-Muster. Diese beiden Muster unterscheiden sich vor allem in Bezug auf das Transaktionsmanagement und die Art der Implementierung.

Das Session-Facade-Muster ermöglicht die einfache Realisierung eines Transaktionsmanagements. Das EJB-Command-Muster bietet dagegen keine einfache Rea-

lisierung eines Transaktionsmanagements, da es einfache Java-Klassen verwendet, für die es in der J2EE kein automatisches Transaktionsmanagement gibt. In Bezug auf die Realisierung eines Transaktionsmanagements ist das Session-Facade-Muster dem EJB-Command-Muster auf jeden Fall vorzuziehen.

Bei der Implementierung des Session-Facade-Musters muss eine zusätzliche Schicht in Form von Fassade-Objekten eingefügt werden. Die Session Facade ruft dann die Geschäftsmethoden der Business-Objects auf. Bei der Implementierung des EJB-Command-Musters muss eine zusätzliche Schicht von Befehls-Beans eingefügt werden, welche die Methoden der Business-Objects aufrufen. Diese Befehls-Beans sind vergleichbar mit den Fassade-Objekten. Zusätzlich wird beim EJB-Command-Muster noch ein Auslöser-Objekt benötigt, das die einzelnen Befehle nacheinander aufruft. Dieses Auslöser-Objekt stellt eine weitere zusätzliche Schicht dar, die beim Session-Facade-Muster nicht benötigt wird.

Da bei der Implementierung des Session-Facade-Musters eine Schicht weniger benötigt wird, führt es zu einer performanteren Lösung als das EJB-Command-Muster. Des Weiteren ist es einfacher zu implementieren, da die zusätzliche Schicht von Befehls-Beans nicht realisiert werden muss, sondern die EJBs direkt aus der Session Facade aufgerufen werden können.

Da das Session-Facade-Muster sowohl beim Transaktionsmanagement als auch bei der Implementierung Vorteile gegenüber dem EJB-Command-Muster hat, wurde für die Kopplung der existierenden Business-Objects der ShopLab Core Platform das Session-Facade-Muster verwendet.

Abbildung 2 zeigt die Struktur eines Geschäftsvorfalls bei Verwendung des Session-Facade-Musters. Der Client greift auf die Session Facade zu, die den Geschäftsvorfall repräsentiert. Der Zugriff auf die einzelnen Business-Objects erfolgt durch die Session Facade.

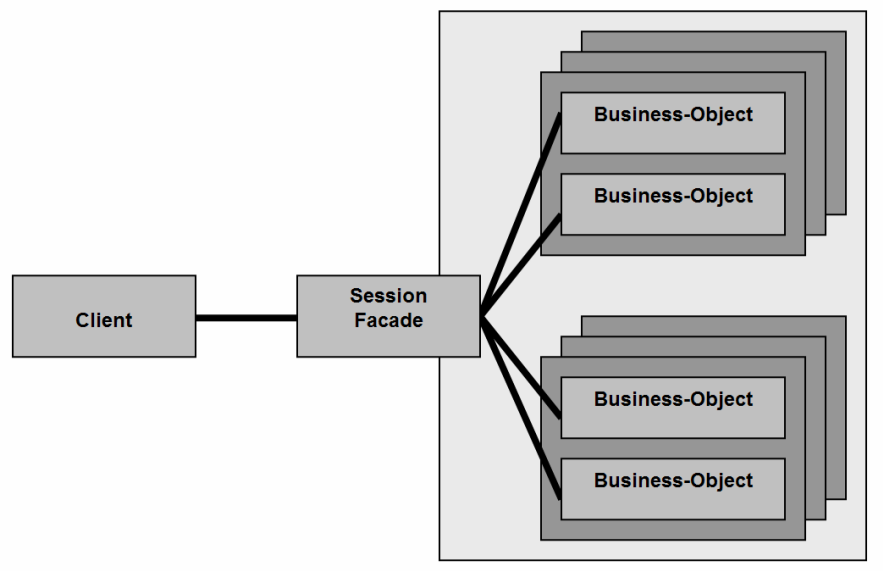

Abbildung 2: Funktionsweise des Session-Facade-Musters

5 Der ShopLab Editor

Der „ShopLab Editor“ (siehe Abbildung 3) zur graphischen Kopplung existierender Business-Objects wurde in Java programmiert. Er nutzt das SDK 1.4.1. Die graphische Oberfläche des „ShopLab Editors“ wurde mit Hilfe der Swing-Klassen realisiert.

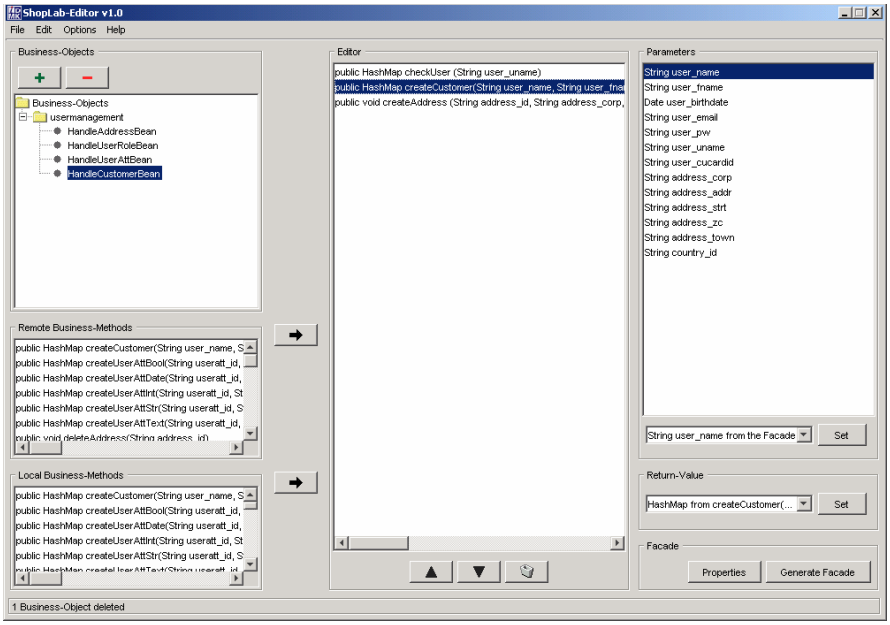

Abbildung 3: Der„ShopLab Editor“

Um die Kopplung der Business-Objects zu ermöglichen, benötigt der „ShopLab Editor“ Informationen über die Geschäftsmethoden der zu koppelnden Objekte, bei denen es sich im Falle der „ShopLab Platform“ um Session-Beans handelt.

Diese Informationen erhält er, indem er das zur Session-Bean gehörende Remote-Interface bzw. Local-Interface ausliest. Eine Session-Bean muss immer über ein Remote-Interface oder ein Local-Interface verfügen. Unter Umständen verfügt sie über beide Interfaces. Im Remote- bzw. Local-Interface sind die Geschäftsmethoden der Session-Bean definiert, die für die Außenwelt sichtbar sind. Durch das Auslesen der Methoden-Definitionen der Interfaces erhält der ShopLab Editor Informationen über die Geschäftsmethoden der Session-Bean. Er kennt nun den Methodennamen, die zu übergebenden Parameter, den Rückgabewert und die ausgelösten Exceptions jeder einzelnen Methode (siehe Abbildung 4).

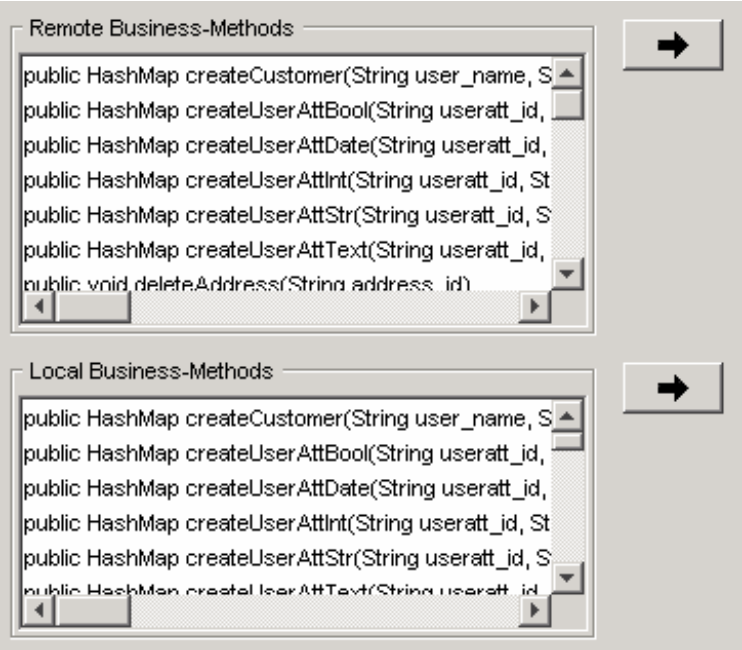

Abbildung 4: Die ausgelesenen Methoden der Business-Objects

Die Session Facade, die generiert wird, enthält genau eine Geschäftsmethode, deren Name im ShopLab Editor bestimmt werden kann. Diese Geschäftsmethode der Fassade repräsentiert den Geschäftsvorfall, der sich aus beliebig vielen Einzelvorgängen zusammensetzen kann. Der Fassaden-Methode können beliebig viele Parameter unterschiedlichster Typen übergeben werden. Der Rückgabewert der Methode kann ebenfalls frei gewählt werden.

Innerhalb der Fassaden-Methode der Session-Facade werden alle Aufrufe der Geschäftsmethoden der Business-Objects gekapselt. Das bedeutet, dass mit dem Aufruf der Fassaden-Methode eine ganze Reihe von Methoden nacheinander aufgerufen werden kann.

Im ShopLab Editor können die Geschäftsmethoden aller Business-Objects in der Reihenfolge angeordnet werden, in der sie später von der Fassade aufgerufen werden sollen. Zum späteren Generieren des Quellcodes ist es erforderlich, dass alle Parameter der Geschäftsmethoden gesetzt sind. Als solcher Parameter einer Methode kann z.B. der Rückgabewert einer vorhergehenden Methode dienen. Aber auch die Parameter der Fassaden-Methode können genutzt werden, um sie den Geschäftsmethoden der Business-Objects als Parameter zu übergeben.

Nachdem die einzelnen Geschäftsmethoden der Business-Objects zu einem Geschäftsvorfall angeordnet wurden und alle Parameter gesetzt sind, generiert der Editor die Klasse der Session-Facade und alle weiteren benötigten Dateien. Das

heißt im Detail, dass eine Session-Bean erzeugt wird, die die Session-Facade repräsentiert. Des Weiteren werden die dazugehörigen Interfaces erzeugt: das Home- und das Remote-Interface sowie das lokale Home- und das lokale Remote-Interface.

Um die Session-Facade nutzen zu können, muss ihre Geschäftsmethode innerhalb der „ShopLab Core Platform“ von einem ShopLab-Event-Handler aufgerufen werden. Dieser Methodenaufruf muss manuell in den Quelltext des entsprechenden Event-Handlers eingefügt werden. Der automatische Eintrag des Methodenaufrufs in den Quelltext wird vom ShopLab Editor Version 1.0 noch nicht unterstützt.

Das folgende Sequenzdiagramm verdeutlicht noch einmal den Aufruf eines Geschäftsvorfalls innerhalb der „ShopLab Core Platform“ unter Verwendung der vom „ShopLab Editor“ generierten Session Facade.

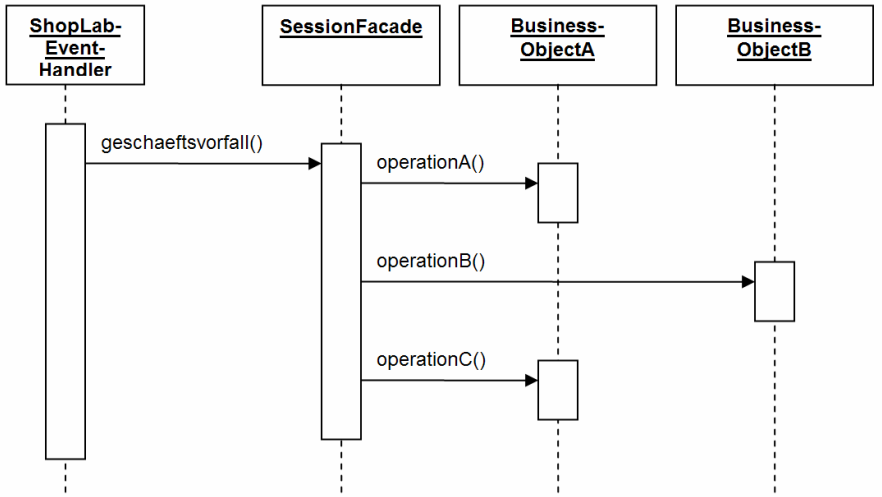

Abbildung 5: Aufruf eines Geschäftsvofalls innerhalb der „ShopLab Core Platform“

Für den Einsatz der generierten Session-Facade ist es notwendig, dass sie als Session-Bean in den Deployment-Deskriptor der entsprechenden JAR-Datei eingetragen wird. Dieser Eintrag wird vom ShopLab Editor in der Version 1.0 noch nicht automatisch vorgenommen, sondern muss manuell eingetragen werden.

Im Deployment-Deskriptor kann auch angegeben werden, ob es sich bei der Session-Facade um eine zustandsbehaftete oder um eine zustandslose Session-Bean handeln soll. Da der ShopLab Editor in der Version 1.0 nur eine Geschäftsmethode pro Session-Facade erstellt, macht die Verwendung einer zustandsbehafteten Session-Bean keinen Sinn. Bei nur einer Methode muss kein Konversationszustand zwischen den Methodenaufrufen gespeichert werden. Folglich sollte eine

zustandslose Session-Bean verwendet werden. Des Weiteren können im Deployment-Deskriptor auch die Transaktionsattribute der Fassade eingestellt werden.

6 Beispiel: Registrieren eines neuen Kunden

Zur Verdeutlichung der Funktionsweise des ShopLab Editors sollen in einem einfachen Beispiel die Geschäftsmethoden „Überprüfung des Benutzernamens“, „Anlegen eines Kunden“ und „Anlegen einer Addresse“ zu einem übergeordneten Geschäftsprozess „Registrierung eines Kunden“ zusammengefasst werden.

Dafür werden die Session-Beans im jeweiligen Package ausgewählt. Session Beans sind im ShopLab System durch die Vorsilbe „Handle“ kenntlich gemacht. Die beiden Session Beans „HandleCustomerBean“ und „HandleAddressBean“ aus dem Package „usermanagement“ enthalten die für diesen Geschäftsprozess benötigten Geschäftsmethoden.

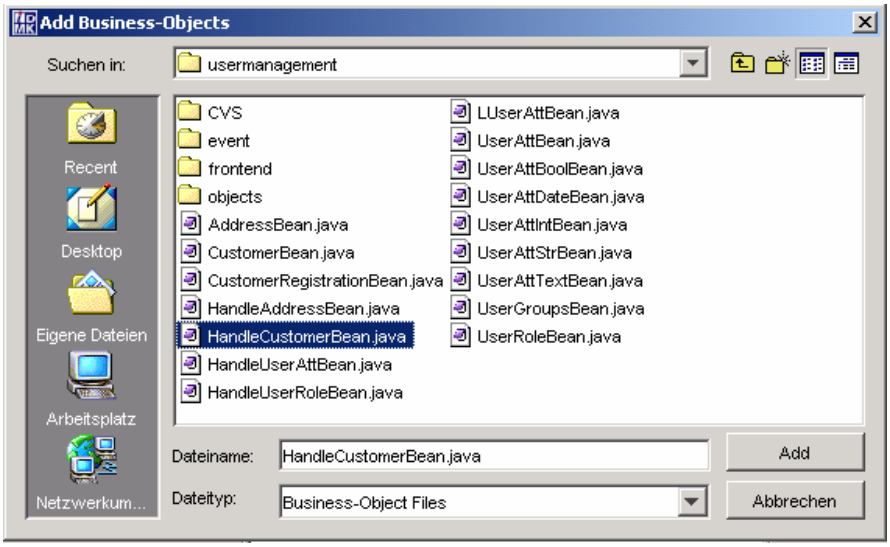

Abbildung 6: Auswahl der Session Beans

Die Geschäftsmethoden „Überprüfung des Benutzernamens“ (checkUser) und „Anlegen eines Kunden“ (createCustomer) aus der HandleCustomerBean und „Anlegen einer Addresse“ (createAdress) aus der HandleAddressBean werden ausgewählt und in den Editor geladen. Dazu können die Methoden des Remote-Interfaces oder des Local-Interfaces genutzt werden, sofern diese vorhanden sind.

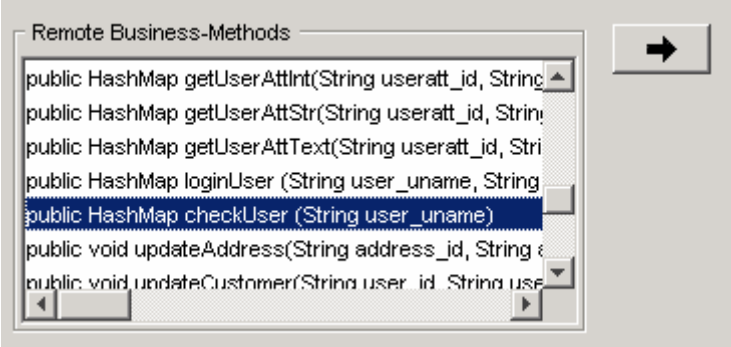

Abbildung 7: Auswahl der Methode „checkUser“ aus dem Remote-Interface

Die ausgewählten Methoden erscheinen im Editor. Die dazugehörigen Parameter, die den Methoden übergeben werden müssen, werden im Parameter-Fenster angezeigt. Den Parametern können dann Parameter der neuen, übergeordneten Fassaden-Methode bzw. Rückgabewerte von in der Ausführungsreihenfolge weiter oben stehenden Geschäftsmethoden zugewiesen werden.

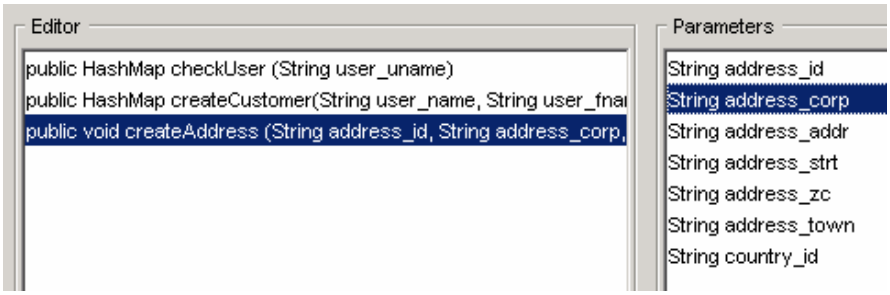

Abbildung 8: Auswahl des Parameters „Address_Corp“ aus der Methode „createAddress“

Wenn allen Parametern Werte zugewiesen worden sind und der Rückgabewert der neuen Fassaden-Methode definiert ist, können weitere Eigenschaften der Fassade deklariert werden. Als Rückgabewert kommen die Eingangsparameter der neuen Fassaden-Methode, Rückgabewerte der einzelnen Geschäftsmethoden oder void (kein Rückgabewert) in Frage.

Als weitere Eigenschaften der Fassade können außerdem Name der Klasse, Package und Speicherort sowie der Name der Geschäftsmethode und deren Parameter festgelegt werden.

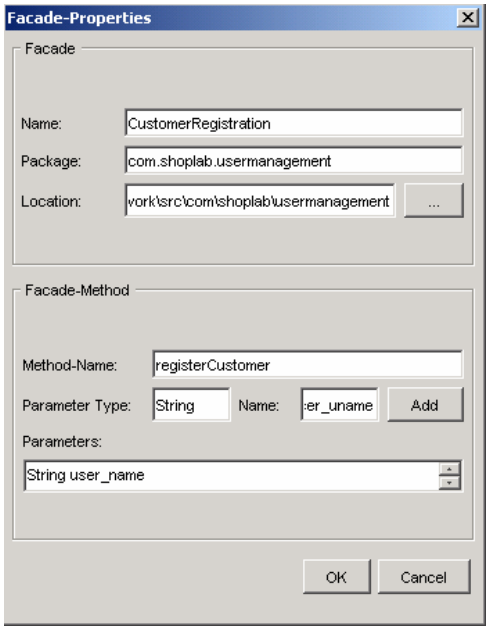

Abbildung 9: Eigenschaften der Fassade

Im Anschluss daran erzeugt der ShopLab Editor automatisch den Quellcode der Fassaden-Klasse inkl. der dazugehörigen Interfaces. Der folgende Ausschnitt aus dem Quellcode der Fassaden-Klasse verdeutlicht die Aufrufe der einzelnen Business-Objects innerhalb der Fassaden-Methode. Sonstige Anweisungen wie JNDI-Lookup, das Finden der Home-Interfaces und erzeugen von Instanzen der Session-Beans wurden aus Gründen der Einfachheit entfernt.

```
public class CustomerRegistrationBean implements SessionBean {

   …

   public HashMap registerCustomer(String user name, String
      user fname, Date user birthdate, String user email, String
      user pw, String user uname, String user cucardid, String
      address corp, String address adr, String address strt,
      String address zc, String address town, String country id)
      throws RemoteException, ShopLabExceptionCollection {

      /* An dieser Stelle wird ein JNDI-Lookup durchgeführt, das Ho-
         me-Interface gesucht und eine Instantz der Session-Bean er-
         zeugt */

      // Aufruf einer Methode des Business-Objects HandleCustomer
      HashMap return_checkUser  = handlecustomerRemote.checkUser
```

```
         (user_name);
      // Aufruf einer Methode des Business-Objects HandleCustomer
      HashMap return_createCustomer =
         handlecustomerRemote.createCustomer(user_name, user_fname,
         user_birthdate, user_email, user_pw, user_uname,
         user cucardid));

      /* An dieser Stelle wird ein JNDI-Lookup durchgeführt, das Ho-
         me-Interface gesucht und eine Instantz der Session-Bean er-
         zeugt */

      // Aufruf der Methode des Business-Objects HandleAddress
      handleaddressRemote.createAddress(addressID, address corp,
         address adr, address strt, address zc, address town,
         country id);

      return return createCustomer;
   }
}
```

Abbildung 10: Ausschnitt aus der Klasse CustomerRegistrationBean

Es wird deutlich, wie einfach die Generierung eines neuen Geschäftsprozesses realisiert werden kann. Natürlich können zur Registrierung neuer Kunden je nach Anwendungsfall weitere Geschäftsmethoden in die Fassaden-Methode integriert werden. Es ist auch möglich, eine einmal erstellte Konfiguration abzuspeichern, so dass kleine Anpassungen für neue Kunden sehr schnell umgesetzt werden können.

7 Weiteres Vorgehen

Der im Rahmen des ShopLab Projektes entwickelte „ShopLab Editor“ soll in seinem Funktionsumfang noch um einige sinnvolle Punkte erweitert werden, die in der ersten prototypischen Implementierung nicht vorgesehen waren und somit auch nicht realisiert wurden.

In der aktuellen Version des „ShopLab Editors“ müssen sowohl die Eintragungen im Deployment-Deskriptor als auch die Aufrufe der Fassaden-Methode in den ShopLab-Event-Handlern noch von Hand vorgenommen werden. Dies bedeutet Programmierarbeit, die in einer neueren Version durch automatische Eintragungen ersetzt werden soll. Das automatische Durchführen dieser Eintragungen ist eine der sinnvollen Erweiterungen, die die Arbeit mit dem „ShopLab Editor“ zusätzlich erleichtern werden. Die notwendigen Informationen für den Deployment-Deskriptor wie z.B. die Angaben über die Session Fassade, die Art der Bean (zustandslos oder zustandsbehaftet) und die Transaktionsattribute der Fassaden-

Methode könnten innerhalb der graphischen Oberfläche spezifiziert werden. Auch die Stellen im Quelltext der ShopLab-Event-Handler, an denen die Fassaden-Methode aufgerufen werden soll, könnten bereits innerhalb des „ShopLab Editors“ ausgewählt werden. Die Eintragungen würden dann beim Generieren der Session-Facade automatisch an den entsprechenden Stellen eingefügt.

Die aktuelle Version des Editors erzeugt immer eine Session-Facade, die nur eine Fassaden-Methode enthält. In einer Weiterentwicklung wäre es unter Umständen sinnvoll, ähnliche Geschäftsmethoden in einer Session-Facade zusammenzufassen, anstatt sie wie bisher in separaten Session-Facaden unterzubringen.

Eine weitere sinnvolle Ergänzung des Funktionsumfangs wäre das Einfügen eigener Codezeilen in die Fassaden-Methode. Zurzeit enthält der Quellcode der Fassaden-Methode nur die sequentiellen Methodenaufrufe der Business-Objects. Es ist zwar möglich, den Rückgabewert einer Methode als Parameter einer andern Methode zu übergeben, es ist aber nicht möglich, den Rückgabewert innerhalb der Fassaden-Methode zu bearbeiten oder von ihm den weiteren Ablauf innerhalb der Fassaden-Methode abhängig zu machen (z.B. in Form einer „if“-Abfrage). Durch das Einfügen eigener Codefragmente in die Fassaden-Methode könnte der Funktionsumfang und die Flexibilität der Fassaden-Methode vergrößert werden. Dazu müsste der „ShopLab Editor“ so verändert werden, dass er das Einfügen von Codezeilen unterstützt.

Der „ShopLab Editor“ in seiner jetzigen Form erzeugt Quellcode, der für die Verwendung innerhalb der „ShopLab Toolbox“ optimiert ist. Eine Verwendung der generierten Klassen außerhalb der „ShopLab Toolbox“ ist momentan aufgrund einiger spezifischer Abschnitte im Quelltext nicht oder nur schwer möglich. Um den „ShopLab Editor“ auch in anderen Bereichen einsetzen zu können, wird über eine Parametrisierung dieser spezifischen Abschnitte nachgedacht. Eine spätere Version des „ShopLab Editors“ könnte um diese Funktionalität erweitert werden.

8 Zusammenfassung

Mit Hilfe des „ShopLab Editors“ wird die Generierung neuer Geschäftsprozesse innerhalb der ShopLab Toolbox erheblich erleichtert. Die direkte Programmierarbeit kann deutlich reduziert werden. Dadurch ist das ShopLab Projekt in der Lage, schnell und flexibel auf neue Anforderungen zu reagieren und die „ShopLab Core Platform“ mit den benötigten Geschäftsprozessen auszustatten. An Hand zweier konkreter Beispiele aus dem Einzelhandel, dem „Interaktiven Spiegel“ und dem „Interaktiven Regal“, konnte im Rahmen des von der EU geförderten Forschungsprojektes ShopLab der erfolgreiche Einsatz des Editors demonstriert werden [Kha+03].

Literatur

[Bien02] Bien, A.: J2EE Patterns, Entwurfsmuster für die J2EE. Addison Wesley, 2002.

[Boo+03] Booth, S.; Westerman, S.; Khakzar, K.; Berger, T.; Pohl, H.-M.; Dubravcova, K.: The Development of 'Hybrid' Multimodal Shopping Systems within a 'Rapid Ethnographic' Methodology. In: Proceedings of HCI Conference. Kreta, Griechenland, 2003, S. 606 - 610.

[Gam+96] Gamma, E.; Helm, R.; Johnson, R.: Entwurfsmuster - Elemente wieder verwertbarer objektorientierter Software. Addison Wesley, 1996.

[Kha+03] Khakzar, K.; Pohl, H.-M.; Frank, W.; Berger, T.; Jöckel, T.; Feßler, M.: Neue multimediale Verkaufs- und Erlebnisräume in den traditionellen Ladengeschäften der Innenstädte. In: Hildebrand, K. (Hrsg.) HMD – Praxis der Wirtschaftsinformatik: IT-Lösungen im Handel. dpunkt.verlag: Heidelberg, 2003, S. 37- 44.

[Sun02] Sun Microsystems: Sun Java Center J2EE Patterns. http://developer.java.sun.com/developer/technicalArticles/J2EE/patterns, Abruf am 2002-08-01.