Technische Universität Dresden – Fakultät Informatik
Professur Multimediatechnik, Privat-Dozentur Angewandte Informatik

Prof. Dr.-Ing. Klaus Meißner
PD Dr.-Ing. habil. Martin Engelien
(Hrsg.)

an der
Fakultät Informatik der Technischen Universität Dresden

unter Mitwirkung des
Bundesministeriums für Bildung und Forschung,
Programm Innovative Arbeitsgestaltung und der
Gesellschaft für Informatik e.V.
GI-Regionalgruppe Dresden

am 28. und 29. September 2006 in Dresden
http://www-mmt.inf.tu-dresden.de/geneme2006/
geneme@mail-mmt.inf.tu-dresden.de

B.3 Die Logistik-orientierte Objekt-Plattform LOOP: Komponentenorientierte Softwareentwicklung vor dem Hintergrund fluider Organisation

Gunter Teichmann, Benjamin Dittes
SALT Solutions GmbH

1. Einleitung

Das Geschäftsfeld der SALT Solutions GmbH ist der Entwurf und die Implementierung von IT-Lösungen für Logistik, Handel und Produktion sowie die Integration dieser Lösungen in die Geschäftsprozesse und Systemlandschaften ihrer Kunden. Stand dabei in der Vergangenheit die Auswahl und Einführung passender Standardsoftware oder die Implementierung optimal zugeschnittener Individualsoftware im Mittelpunkt, beobachten wir insbesondere im Marktumfeld der Kontraktlogistik ein wachsendes Interesse an Lösungen, die sich dynamisch an immer schneller auftretende Anforderungsänderungen anpassen lassen. Dieses Interesse resultiert aus einem zentralen Trend zur „High-End"-Kontraktlogistik, der davon gekennzeichnet ist, dass immer umfassendere und komplexere Dienstleistungen von Logistikunternehmen übernommen werden, die im Sinne eines „Business On Demand" mit immer kürzeren Reaktionszeiten bis hin zur sofortigen Reaktion auf Kundenbedürfnisse erbracht werden.

Kostendruck und geringe Margen in der Branche erzwingen den Einsatz der Information Technology (IT). Eine von der SALT Solutions GmbH gemeinsam mit der Fraunhofer Arbeitsgruppe für Technologien in der Logistik durchgeführte Studie kommt zu dem Ergebnis, dass in der Branche der Einsatz der Information Technology als wesentliches Differenzierungsmerkmal für die Zukunft bewertet wird: Nur mit Kompetenz in IT sind zukünftige Kontrakte zu gewinnen. Ein weiteres Ergebnis der Studie ist, dass jedoch gerade dafür keine geeigneten Standardsoftwareprodukte am Markt erhältlich sind. [Kla05]

Die verfügbaren Softwareprodukte für Geschäftssoftware adressieren die klassische Industrie. Die logistischen Prozesse werden eher am Rande abgebildet. Würde man das wachsende Leistungsspektrum branchenübergreifend agierender Kontraktlogistiker mit den jeweils passenden Systemen unterstützen, entstünde eine komplexe Systemlandschaft, die nur schwer zu handhaben wäre und außerdem einen nicht tolerierbaren Kostenanstieg zur Folge hätte. Das ist insbesondere für kleine und mittlere

Kontraktlogistiker von großer Bedeutung, da für diese Unternehmen der IT-Einsatz einen besonders starken Kostenfaktor darstellt.

Eine weitere Ursache für die Nichtverfügbarkeit geeigneter Standardsoftware sehen wir nicht zuletzt darin, dass logistische Prozesse und Kooperationen zunehmend fluiden Charakter annehmen: Zeitlich befristet schließen sich Lieferanten, Endkunden, Produzenten und Logistikdienstleister zu Verbünden zusammen, die Merkmale virtueller Unternehmen aufweisen. [Neu05-a]

Klassische Geschäftssoftware kann den damit verbundenen neuen Anforderungen an die Unterstützung fluider Organisationsformen nur unzureichend gerecht werden. Besser geeignet scheinen Systeme, die nicht fix auf die Erfüllung bestimmter fest definierter Anforderungen hin entwickelt werden, sondern von vornherein auf die leichte und kostengünstige Anpassbarkeit an neue Anforderungen hin optimiert sind. Genau hier haben wir mit unserem Konzept für die Logistik-orientierte Objekt-Plattform (LOOP) angesetzt: Wir wollten ein schlankes Baukastensystem speziell für die Kontraktlogistik schaffen, mit dem sich immer wieder neue maßgeschneiderte Lösungen erzeugen lassen.

Dieses Baukastensystem sollt sich in erster Linie dadurch auszeichnen, dass es flexibel auf neue und geänderte Anforderungen reagieren kann. Neue Prozesse sollen ohne Programmierung aus vorhandenen Bausteinen zusammengesetzt werden können, und Änderungen im Prozess sollen sich durch den Austausch oder eine geänderte Anordnung einzelner Bausteine ebenfalls ohne Programmierung abbilden lassen. Die Beschreibung der Prozesse und Lösungen soll in einer domänenspezifischen Sprache erfolgen, aus der dann modellgetrieben die eigentlich ausführbare Software durch automatisierte Transformationen generiert werden kann.

Die eigentliche Herausforderung bestand darin, Konzepte zu entwickeln, wie man Domänenlogik quasi „zerlegen“ kann, um sie in Form wieder verwendbarer Fachkomponenten gießen zu können und die fachlichen Prozesse als Fachkomponenten-Kommunikations-Protokoll zu formulieren.

2. Lösungsansatz

Bei der Suche nach einem geeigneten Lösungsansatz haben wir uns von der Idee der feature-getriebenen Entwicklung (FDD) inspirieren lassen. FDD legt zunächst eine Gesamtarchitektur (ein Domänen-Objektmodell) und anschließend eine Liste von

Features fest. Ein Feature ist ein „kleines“ Stück kundenbewerteter Funktionalität [Pal02].

Zunächst haben wir also ein logistisches Domänenmodell erstellt, das sozusagen den größten gemeinsamen Nenner darstellt, den alle Logistikdienstleister per se erfüllen. Dieses Modell bildet das „fachliche Rückgrat“ von LOOP. Es trifft nur drei sehr allgemeine Grundannahmen:

- Ein Logistikdienstleister erbringt logistische Dienstleistungen, die im Wesentlichen aus dem Transport, dem Umschlag und der Lagerung von Waren (TUL) bestehen. Diese Dienstleistungen werden im Kundenauftrag erbracht, es existiert ein logistischer Auftragsbegriff.
- Zur Erbringung dieser Dienstleistung ist eine Logistikinfrastruktur vorhanden, die aus einer Menge von Logistikstützpunkten und Transportmitteln unterschiedlicher Art und Größe besteht, die zwischen den Stützpunkten verkehren.
- Die Logistikstützpunkte sind entweder reine Umschlagspunkte, oder es sind zusätzlich Lagerkapazitäten angeschlossen (in diesem Fall existieren zusätzliche Lagerprozesse, die im Modell integriert betrachtet werden).

Im nächsten Schritt haben wir die von uns in den vergangenen Jahren realisierten Projekte analysiert und den Versuch unternommen, die potenziellen Unterschiede in Strukturen und Prozessen jeweils so zu abstrahieren, dass sie das Prozessmodell nicht von vornherein komplizierter als unbedingt erforderlich werden lassen.

Beispielhaft soll dies anhand eines Problems aus der Versandlogistik verdeutlicht werden: Hier besteht ein wesentlicher Unterschied in der Art der Abwicklung je nachdem, ob ein Versandartikel in einem Stück versendet werden kann, oder ob er aus mehreren Teilen besteht. Bei der Modellierung hat man zwei Möglichkeiten: Entweder man unterscheidet zwei Teilmodelle, oder man abstrahiert von den konkreten Versandartikeln zu einer abstrakten Versandeinheit. Wir befürworten die zweite Variante und betrachten die konkrete Zuordnung von Versandartikeln zu Versandeinheiten als Kandidaten für ein kundenspezifisches Feature.

Ausgangspunkt der Betrachtung war die Frage, wie man Domänenlogik in Form wieder verwendbarer Bausteine zerlegen kann.

Der erste Teil unseres Ansatzes zur Beantwortung dieser Frage lautet:

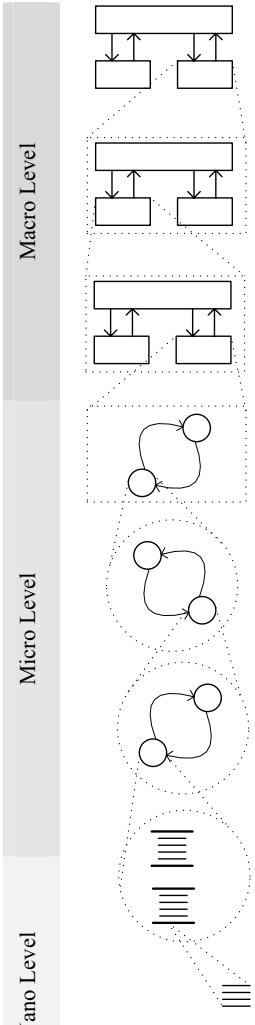
- Modellierung des Kerns der Domäne als Integrationsmodell und vollständige Implementierung der Logik dieses Kerns als bereits lauffähige Minimallösung.
- Identifikation von potenziellen Features, die diesen Kern erweitern, aber erst beim konkreten Einsatz genau definiert werden.
- Bereitstellung von Mechanismen und Tools, die ein nachträgliches automatisiertes „Einbauen" neuer Features nach dem Prinzip modellgetriebener Softwareentwicklung unterstützen. Neue Komponenten können durch diesen Einbaumechanismus bei ihrer Erstellung durch das Hinzufügen dieser Features automatisiert darauf vorbereitet werden, innerhalb der Kernlogik der Domäne bestimmte Rollen einzunehmen.

Eine weitere Herausforderung bestand in der Frage, woraus der Baukasten eigentlich bestehen soll. Der Begriff der „Komponente" erschien uns bei genauerer Betrachtung durch seinen Gebrauch im Kontext von Technologien wie J2EE und .net nicht geeignet, da er kleinere Einheiten beschreibt, als die von uns gesuchten Fachkomponenten.

Eine wichtige Inspiration bei der Beantwortung dieser Frage fand sich in dem von Ralf Westphal vorgeschlagenen neuen Architekturmodell der Softwarezellen. [WES05], [WES06-a], [WES06-b]

Ausgangspunkt seiner Überlegungen sind ganz ähnliche Fragestellungen, wie: „Was ist eine verteilte Anwendung?" oder gar „Was ist überhaupt eine Anwendung?" und eine damit einhergehende Kritik an Schichtenarchitekturen. Westphal schlägt unter der Bezeichnung „Software-Universum" ein allgemeines Rahmenwerk vor, das auf dem Konzept einer selbstähnlichen Holarchie aus (mindestens) sieben Ebenen von Holons[1] basiert (siehe Abbildung 1).

[1] Ein Holon ist eine Entität, die Ganzes (griechisch *holos* für „das Ganze") und gleichzeitig Teil eines Ganzen (griechisch *on* für „Teil von") ist. Der Begriff wurde 1967 von Arthur Koestler geprägt [Koe90]



Holarchieebene (Bezeichnung bei Westphal)	Beispiele in LOOP	Paradigma
Software Societies	**LOOP im Ganzen**	Serviceorientierte Architektur
Software Solution	**LOOP-Instanz (z. B. eines Kunden)**	
Software Cell / Application	**LOOP-Applikation (z. B. LOOP Warehouse, LOOP Transport)**	
Software Components	**Komponenten einer LOOP-Applikation (z. B. Wareneingang)**	Komponentenbasierte Entwicklung
Software Assemblies	**Ein LOOP-Namespace**	
Software Types	**Die Business Objekte**	Objektorientierte Programmierung
Software Subroutines	**Statusübergänge und Transaktionen auf Business Objekten**	
Software Statements	**C# Statements**	

Abbildung 1: Das Software-Universum als Holarchie und seine Ausprägung in LOOP

Dieses Rahmenwerk erschien uns für unsere Zwecke sehr geeignet, da es auch ermöglicht, bereits vorhandene Softwareartefakte unterschiedlicher Abstraktionsebenen in unseren Baukasten einzugliedern.

Wenn unser Baukasten aus einer solchen Holarchie selbstähnlicher Softwarebausteine bestehen soll, lässt sich die Frage nach der Zerlegung der Domänenlogik präzisieren:

- „Wie kann die Logik eines fachlichen Domänenmodells auf eine Holarchie wieder verwendbarer Einheiten auf unterschiedlichen Ebenen transformiert werden?“
- „Wie kann in einer derartigen Holarchie mit einer zu erwartenden explosiv steigenden Zahl von Abhängigkeiten Kompatibilität und Konsistenz der einzelnen Einheiten gesichert werden?“

Einer Antwort auf diese Fragen soll uns der zweite Teil unseres Lösungsansatzes näher bringen. Wir sind davon ausgegangen, dass sich auf den niedrigen Ebenen einer Holarchie die Anwendung von Mustern in der Praxis sehr gut bewährt hat. Unter Berücksichtigung der Selbstähnlichkeit der Holons auf unterschiedlichen Ebenen haben wir den Versuch unternommen, Muster auf allen Ebenen der Holarchie zu identifizieren.

Um diese Muster zu finden, sind wir von einem niedrigen Abstraktionsniveau (bei Westphal der Ebene der Software-Typen) als Systembasis ausgegangen und haben von Ebene zu Ebene weiter abstrahiert. Dieser Lösungsweg spiegelt sich letztendlich auch im Aufbau der logistik-orientierten Objekt-Plattform LOOP wieder und soll in den folgenden Ausführungen nachvollzogen und veranschaulicht werden.

3. Aufbau von LOOP

Die in LOOP realisierten Basis-Strukturen mit dem niedrigsten Abstraktionsniveau haben wir Business-Objekte (BO) getauft. Alle in den höheren Schichten verwendeten Strukturen sind ohne Bezug zu mindestens einem Business-Objekt nicht lebensfähig.

Die Abbildung 2 zeigt die grundlegenden Bestandteile der Plattform.

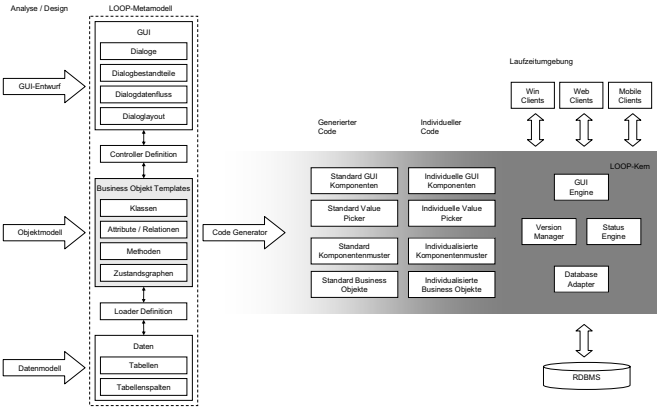


Abbildung 2: Der grundlegende Aufbau von LOOP

3.1 Business-Objekte und Templates

Business-Objekte sind Klassen, die sich durch die folgenden Eigenschaften auszeichnen:

- BO besitzen eine Identität.
- BO bestehen aus Feldern. Jedes Feld ist entweder von einem einfachen Datentyp (Integer, Long, Double, DateTime) oder von einem komplexen Datentyp (ein anderes BO). Ein Feld kann auch einen Verweis auf eine Liste von BO enthalten.
- BO können unterschiedliche Zustände annehmen und haben einen Lebenszyklus. Die Beschreibung der möglichen Zustände und Zustandsübergänge erfolgt durch Zustandsübergangsdiagramme.
- BO haben Methoden, die jeweils von einem oder mehreren Statusübergängen aufgerufen werden können.

Die Beschreibung eines Business-Objekts mit seinen Feldern, Methoden und Zustandsübergangsgrafen erfolgt über standardisierte Templates, die alle notwendigen strukturellen Informationen über die konkreten Business-Objekte formal hinreichend spezifizieren, um daraus durch automatisierte Transformationen lauffähige Bibliotheken generieren zu können. Um aber erste einfache Business-Anwendungen aus den Business-Objekten komponieren zu können, müssen sie zwei weitere Voraussetzungen

erfüllen: Sie müssen persistent gemacht werden und in einer Benutzeroberfläche dargestellt werden können. Zu diesem Zweck haben wir jedem Template zwei weitere beschreibende Konstrukte hinzugefügt:

- Loader ordnen jedem Feld eines BO eine Spalte in einer Datenbanktabelle (oder einem anderen Speichermedium) zu.
- Controller kapseln die internen Strukturen eines BO, um eine intelligente Anzeige in standardisierten GUI-Elementen zu ermöglichen.

3.2 Der LOOP-Kern: Lebensraum für Business-Objekte

Um aus den bisher beschriebenen Basisstrukturen ein lauffähiges System zu erzeugen, bedarf es einer Ausführungsumgebung, die der LOOP-Kern bereitstellt. Er besteht aus Loader-Implementationen, die das Laden der BO aus einem permanenten Datenspeicher in den Hauptspeicher realisieren und datenbanknahe Transformationen durchführen. Controller-Implementationen realisieren die Darstellung der BO in den verschiedenen Benutzeroberflächen (Windows-Anwendungen, Webbrowser oder andere).
Eine weitere Aufgabe des LOOP-Kerns ist die internationalisierte Erzeugung der Benutzeroberflächen aus dynamisch miteinander verschalteten standardisierten Oberflächenelementen. Auch das transaktions-orientierte Management der Statusübergänge und die Sicherstellung konsistenter Änderungen bei Multi-User-Systemen wird durch den LOOP-Kern übernommen.

3.3 Domänenübergreifende Komponentenmuster

Mit den Basis-Strukturen der BO haben wir einen Ausgangspunkt für die Abstraktion in die höheren Ebenen der Holarchie vorliegen. Zunächst soll das Zusammenwirken mehrerer BO in einem einfachen gemeinsamen Kontext untersucht werden. Durch die Möglichkeit der Referenzierung bzw. Aggregation können bereits einfache Komponentenmuster im Zusammenspiel einzelner BO abgeleitet werden. So kann beispielsweise mit einem BO Auftragskopf, das einen Verweis auf eine Liste von einzelnen Auftragspositionen enthält, eine Kopf-/Positionsstruktur definiert werden. Dieses Muster stellt zunächst ein reines Strukturmuster dar. Allerdings lassen sich gewisse Aussagen über ein mögliches zugehöriges Verhaltensmuster treffen: Ist eine Auftragsposition nicht lieferbar, soll auch der gesamte Auftrag nicht ausgeliefert werden. Dieses Verhaltensmuster lässt sich mit dem Konzept der gekoppelten Zustandsgrafen sehr allgemein beschreiben. Mit der Umsetzung dieses Konzepts als erstes Komponentenmuster in LOOP war der Grundstein für das weitere Vorgehen gelegt.

3.4 Von der Struktur zum Verhalten

Wir haben daraufhin in LOOP einen Mechanismus realisiert, den wir „Feature Separation and Injection“ genannt haben. Dieser Mechanismus ermöglicht es, nicht nur Struktur-, sondern auch Verhaltensmuster für das Zusammenwirken mehrerer BO auf abstraktem Niveau zu definieren und mit einem nicht-invasiven Verfahren auf die konkreten BO abzubilden.

Als Mittel zur Umsetzung haben wir bewusst auf das klassische objektorientierte Konzept der Vererbung verzichtet, da wir Vererbung für kundenspezifische Anpassungen der BO verwenden und auf Mehrfachvererbung verzichten wollten, um Mehrdeutigkeiten im Rahmen des Diamond-Problems zu umgehen. Auch wollten wir einem BO ermöglichen, Rollen in mehreren Verhaltensmustern einzunehmen. Stattdessen benutzen wir zum Einsatz der Muster eingeschränkte Mehrfachvererbung, indem wir über spezielle Codegeneratoren den erforderlichen Code in die Implementierungen der beteiligten BOs quasi „hineingenerieren“. Bei diesem generierten Code handelt es sich nicht um die Implementierung des Musters selbst, sondern um „Glue Code“ zur Verknüpfung der BO-Implementierung mit der Implementierung des Musters zur Laufzeit.

Im Rahmen des Entwurfs haben wir eine Reihe struktureller Muster identifiziert, die sich durch einen hohen Allgemeinheitsgrad auszeichnen:

- Klassifikator/Gruppierer,
  „Klassifizierendes BO“ als Spezialfall des BO, das sich dadurch auszeichnet, dass es fast keine eigenen Daten hat, sondern sich durch Gruppierung anderer BO definiert (z. B. Warengruppen, Gefahrenklassen, ...)
- „Virtuelles BO“ als dynamisch entstehendes und wieder verschwindendes BO (z. B. Konto, Quant, Saldo und Bestand)
- „Komposites BO“ ist ein Muster für BO, die ihren Wertevorrat aus dem Kreuzprodukt anderer BO beziehen. Die klassischen Bewegungsdaten realisieren i. d. R. dieses Muster (z. B. Bestellung = Teilmenge eines Kreuzproduktes aus Kunde, Artikel und Lieferant)

Standen bei der Findung dieser bisher identifizierten domänenübergreifenden Muster strukturelle Eigenschaften im Vordergrund, so sind für die Findung domänenspezifischer Muster nun komplexere Vorgänge interessant, bei denen mehrere BO bei der Erfüllung einer gemeinsamen Aufgabe beteiligt sind. Als grundlegendes Denkmodell zur Systematisierung solcher Vorgänge hat sich die Anwendung des Operand-Operator- Operationsprinzips als sehr hilfreich erwiesen. [Neu06]

Es geht davon aus, dass jedes System eine Aufgabe erfüllt. Eine Aufgabe wird als Struktur über Operationen verstanden. Operationen werden auf einem oder mehreren Operanden ausgeführt und ändern deren Zustände.

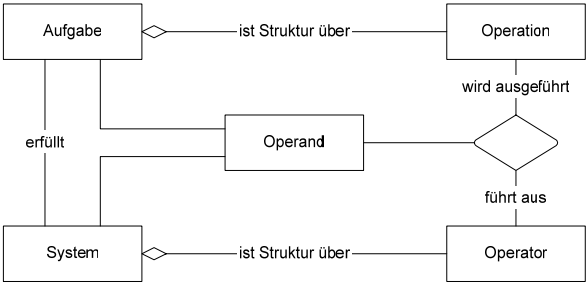


Abbildung 3: Operation, Operator und Operand

Mit den BO haben wir bereits die Kandidaten für die Operanden definiert: BO können unterschiedliche Zustände annehmen und haben einen Lebenszyklus. Die möglichen Zustände und Zustandsübergänge sind in den Zustandsübergangsdiagrammen der BO beschrieben.

Operatoren dienen der Verbindung in die reale Welt, in unserem Kontext sind sie als mobile Datenerfassungsgeräte oder auch als automatische Fördertechnik denkbar.
Um die Findung geeigneter Musterkandidaten zu erleichtern, verwenden wir die folgende Systematisierungsvorlage.

Aufgabe	System	Operator	Operation	Operand
Der Zweck, was getan werden soll.	Zugeordnetes Holon auf der niedrigsten Ebene, welche die gesamte Aufgabe mit seinen Operationen allein abdecken kann.	Ausführende Einheit: Ein Operator führt Operationen im konkreten System (bzw. der realen Welt) aus.	Zerlegung der Aufgabe in mehrere Einheiten, die jeweils für die Transformation von Operanden in einem Stück zuständig sind	Veränderte Einheit: Durch Operationen wird der Zustand der Operanden verändert

Tabelle 1: Systematisierungsvorlage zur Findung potenzieller Muster

Man stelle sich beispielsweise eine Einlagerung als Operation vor: Man möchte einen Gegenstand (z. B. eine Palette) in einen anderen Gegenstand (z. B. ein Hochregal) hineinstellen und nachträglich wieder finden. Mit dieser Einlagerung ist mindestens ein Zustandsübergang auf beiden beteiligten Operanden verbunden: Der Zustand der Palette wechselt von „nicht eingelagert“ auf „eingelagert“, der Zustand des Regalfachs wechselt von „frei“ auf „belegt“.

Damit die Operation gelingt, müssen bestimmte Vorbedingungen erfüllt sein: Die Außenmaße der Palette müssen kleiner sein als die Innenmaße des gewünschten Regalfachs und das Gewicht der Palette darf das zulässige Gesamtgewicht im Regalfach nicht überschreiten.

Wir wollen eine solche Operation nicht auf den konkreten Klassen ihrer Operanden implementieren, sondern das Muster der Operation selbst auf einer abstrakten Ebene. Diese Implementierung kann dann via „Feature Separation and Injection“ von den beteiligten BO in ihrer Rolle als Operanden der jeweiligen Operation benutzt werden. Zu diesem Zweck ist es erforderlich, von der konkreten Operation zu abstrahieren.

3.5 Vom Konkreten zum Abstrakten

Betrachtet man die Operation der Einlagerung von einem abstrakteren Standpunkt, lassen sich die Operanden dadurch beschreiben, dass sie bestimmte Fähigkeiten, wie „kann etwas Kleineres aufnehmen“ bzw. „kann in etwas Größeres aufgenommen werden“ haben und bestimmte damit im Zusammenhang stehende Merkmale (nämlich Länge, Breite, Höhe und Gewicht) aufweisen müssen. Die Operation selbst lässt sich dann als Muster beschreiben: „Nimm das Kleinere und stecke es in das Größere und beachte dabei die Maße und Gewichte“.

Damit ist das Muster nicht mehr auf die Einlagerung beschränkt, sondern lässt sich auch auf andere Strukturen der gleichen Ebene der Holarchie anwenden, z. B. die Beladung eines Fahrzeugs oder einen Kommissioniervorgang.

Im nächsten Abstraktionsschritt wird dieses Prinzip auch auf die höheren und niedrigeren Ebenen der Holarchie übertragen. Im Mittelpunkt steht stets die Frage, welche Funktionalität für mehr als eine Ausprägung verwendet werden kann. Ausgangspunkt für die Beantwortung sind die Fähigkeiten und Merkmale der Operanden, die für eine Operation erforderlich sind. An dieser Stelle kommt uns für die Modellierung eine grundlegende Eigenschaft der Holone entgegen, nämlich ihre

Selbstähnlichkeit. Durch sie eröffnet sich die Möglichkeit, Muster in unterschiedlichen Ebenen der Holarchie einheitlich zu beschreiben, zu implementieren und dadurch auch auf unterschiedlichen Abstraktionsebenen wieder zu verwenden.

3.6 Findung domänenspezifischer Muster

Wir haben das folgende allgemeine Vorgehen zur Findung domänenspezifischer Muster in mehreren Abstraktionsebenen entwickelt:

1. Die Holarchie definieren. Für jede Ebene sind Kriterien zu formulieren, z. B. soll sich eine Anwendung dadurch auszeichnen, dass sie separat laufen kann.
2. Die Operationen pro Holarchie definieren und beschreiben. Für jede Operation sind die Fähigkeiten, Merkmale der beteiligten Operanden sowie die Bedingungen und Suboperationen der Operationen festzulegen.
3. Aus den Operationen durch Abstraktion Muster ableiten, die über mehrere Holarchieebenen vorkommen können.
4. Codegeneratoren je Muster für die verschiedenen Holarchieebenen bauen. Dies ist deshalb erforderlich, da die Kommunikationsprotokolle der unterschiedlichen Holarchieebenen voneinander abweichen können (z. B. Methodenaufrufe auf den unteren Ebenen vs. SOAP-Nachrichten zwischen Anwendungen auf den höheren Ebenen).
5. Tools zur Zuordnung der konkreten BO-Attribute zu den Merkmalen der Operationsmuster bereitstellen.

3.7 Ableitung kundenspezifischer Lösungen

Mit dem beschriebenen Vorgehen sind wir in der Lage, einen Softwarebaukasten bereitzustellen, der die grundlegenden Funktionalitäten einer Anwendungsdomäne (in unserem Fall der logistikspezifischer Geschäftsanwendungen) auf mehreren Holarchieebenen zur Verfügung stellen kann.

Aus den domänenspezifischen Mustern können im Prototyping-Verfahren konkrete Lösungen komponiert werden. Die tatsächliche Ausprägung der Business-Objekte wird in einem kundenspezifischen Objektmodell beschrieben und per „Feature Injection“ in die Plattform eingeklinkt. Aus den Beschreibungen der konkreten Business-Objekte werden bereits lauffähige LOOP-BO-Klassen generiert, die wahlweise auch als Basisklassen zur Überladung mit kundenspezifischen Zusatzfunktionalitäten über Einfachvererbung weiterentwickelt werden können.

3.8 Weitere Anpassungen per „Plug & Play“

Da das Zusammenspiel der Konstrukte auf allen Ebenen der Holarchie formal nach dem gleichen Prinzip beschrieben ist, sind wir in der Lage, die Anforderungen an Software zur Unterstützung fluider Organisation hinsichtlich einfacher Rekonfigurierbarkeit sehr elegant zu erfüllen.

Wir wollen auf allen Ebenen der Holarchie grafische Tools bereitstellen, mit deren Hilfe sich Workflows aus Prozessbausteinen zusammenfügen lassen, die dann im „Plug & Play“-Verfahren in die bestehenden Lösungen eingefügt werden können.

4. Softwareentwicklung mit LOOP und fluide Organisation

Interessanterweise sehen wir uns mit der Bereitstellung von LOOP für die Softwareentwicklung in den Projekten als Softwareentwicklungsfirma selbst mit den Merkmalen einer fluiden Organisation konfrontiert:

Die Einführung einer mit LOOP erstellten Unternehmenslösung erfolgt i. d. R. als Projekt unter Mitwirkung des Kunden. In den einzelnen Projektphasen arbeiten Mitarbeiter sowohl des Softwareanbieters als auch des Kunden in Teams zusammen, die immer wieder neu zusammengestellt werden. Dadurch kann es geschehen, dass aus unterschiedlichen Projekten Anforderungen resultieren, die miteinander im Widerspruch stehen. Eine projektspezifische Modifikation von LOOP birgt allerdings die Gefahr, dass damit die Kompatibilität einzelner LOOP-Lösungen verloren geht.

Es stellt sich somit für uns die Herausforderung, vor dem Hintergrund einer solch fluiden Organisation LOOP selbst in einer geeigneten Art und Weise so weiterzuentwickeln, dass möglichst nicht nur aktuelle Anforderungen erfüllt werden, sondern auch zukünftige Anforderungen, die sich erst beim Einsatz von LOOP in einem anderen Projekt ergeben. Aus diesem Grund haben wir den LOOP-Kern selbst nach dem skizzierten Verfahren entwickelt und strikt nach Releases organisiert. Ein Release ist die eindeutige Kennzeichnung aller Objekte eines Namensraumes. Die Releases haben bei der Wiederverwendung von Code eine tragende Rolle: Benutzt ein Projekt die Ergebnisse eines anderen (wie es beim LOOP-Kern überall der Fall ist), so ist es an dessen Aufbau und Funktionsweise angepasst. Damit der referenzierte Namensraum trotzdem weiterentwickelt werden kann, werden in LOOP andere Projekte zusätzlich zur Bezeichnung auch anhand einer Release identifiziert und an das eigene Projekt gebunden. Durch eine in LOOP integrierte Releaseverwaltung können so Projekte mit

referenziellen Abhängigkeiten abgetrennt in einem eigenen Zyklus weiterentwickelt werden, ohne auf den Zyklus der referenzierten Objekte Rücksicht nehmen zu müssen.

Damit dieses Vorgehen funktioniert, werden die Business-Objekte des LOOP-Kerns bei der Erzeugung eines neuen Release komplett kopiert und die referenzierenden Projekte greifen zunächst weiterhin auf die alten Versionen zurück. Parallel zur Entwicklung des neuen LOOP-Release werden immer auch Migrationstools entwickelt, die die strukturellen Änderungen zwischen dem LOOP-Release *n* durch Anpassung der auf LOOP-Release *n-1* erstellten Lösungen nachvollziehen.

5. Zusammenfassung

Dieser Beitrag stellt einen neuartigen und experimentellen Ansatz für die komponentenorientierte Softwareentwicklung vor, der derzeit bei der SALT Solutions GmbH entwickelt wird. Im Zentrum dieses Ansatzes steht die Entwicklung eines Software-Baukastensystems, das auf die leichte und kostengünstige Erzeugung neuer Softwarelösungen und deren Anpassung an immer wieder neue Anforderungen hin optimiert ist.

Im Rahmen eines von der Sächsischen Aufbaubank mit Mitteln der Europäischen Union geförderten Forschungsvorhabens wird derzeit die Logistik-orientierte Objekt-Plattform LOOP auf Basis dieses Ansatzes als Prototyp entwickelt.

Literatur

[Kla05] P. Klaus: IT-Trends in der Kontraktlogistik, Studie zum 2. Münchner Logistik-Dialog, 2005

[Koe90] A. Koestler: The Ghost in the Machine, Penguin reprint edition, 1990

[Neu05-a] D. Neumann, J.F. Schaible: Fluide Organisation von Informationssystemen in der Logistik am Beispiel der Lufthansa Technik Logistik GmbH, Workshop GeNeMe 2005

[Neu05-b] D. Neumann, G. Teichmann, F. Wehner, M. Engelien: VU-Grid – Integrationsplattform für virtuelle Unternehmen, Workshop GeNeMe 2005

[Neu06] D. Neumann: Modellierung Fluider Organisationen und ihrer informationstechnischen Unterstützung, TUDpress, 2006

[Pal02] S.R. Palmer, J.M. Felsing: Practical Guide to Feature-Driven-Development, Prentice-Hall, 2002

[Wes05] R. Westphal: Softwarezellen: Ein Architekturmodell für Software in Netzwerken, Artikel in OBJEKTspektrum 5/2005

[Wes06-a] R. Westphal: Softwarezellen – Moderne Softwaresysteme modellieren und produzieren, OOP2006

[Wes06-b] R. Westphal: Software Cells, http://weblogs.asp.net/ralfw/category/9899.aspx?Show=All