

TOPPrax: Aspektorientierte Programmierung für die Praxis betrieblicher Softwareentwicklung

Ansprechpartner: Dr. Stephan Herrmann

Technische Universität Berlin

Franklinstr. 28/29

10587 Berlin

info@topprax.de – <http://topprax.de>

TU Berlin	TU Darmstadt	FhG FIRST	GEBIT Solutions	Daedalos Consulting
-----------	--------------	-----------	-----------------	---------------------

Kurzfassung

Das Projekt TOPPrax soll an realen Fallbeispielen zeigen, dass die *aspektorientierte* Softwareentwicklung reif ist, die Komplexität moderner Software beherrschbar zu machen.

Ausgewählt wurden die Ansätze *Object Teams* [Herrmann 02] und *Caesar* [Mezini, Ostermann 03], die versprechen, die Modularisierung von Software dort zu verbessern, wo herkömmliche Ansätze scheitern, weil verschiedene Aspekte bzw. Teilbereiche der Software sich überlappen, auf komplexe Art miteinander interagieren und scheinbar widersprüchliche Anforderungen an die Struktur der Software stellen. Die verbesserte Modularität soll Verständlichkeit, Wiederverwendung, Anpassbarkeit und Evolutionsfähigkeit erhöhen.

Während rein akademische Arbeiten in diesem Bereich bereits einen großen Nutzen andeuten, steht der tatsächliche Nachweis in der Praxis noch aus. Hier wird TOPPrax durch direkte *Vergleichsstudien* Bewertungsgrundlagen für zukünftige Entscheidungen liefern. Damit ein Praxiseinsatz realistisch wird, werden parallel Konzepte, Werkzeuge und Methodik für die genannten Ansätze vervollständigt.

1 Einleitung und Vorstellung des Themenkomplexes

Das Projekt TOPPrax soll neueste Techniken der aspektorientierten Softwareentwicklung in die Praxis betrieblicher Softwareentwicklung einführen und ihr Einsatzpotenzial bewerten.

Aspektorientierte Softwareentwicklung hat in den letzten Jahren große öffentliche Beachtung gefunden, da sie verspricht, eine klare modulare Struktur von Software auch dort zu ermöglichen, wo bisherige Techniken scheitern. Erste prototypische Sprachen wie AspectJ [Kiczales+ 01] und Hyper/J [Ossher, Tarr01] zeigen neuartige Lösungsansätze und haben in der jüngsten Vergangenheit als Basis gedient, erste partielle Erfahrungen in diesem Bereich zu sammeln.

1.1 Aspektorientierung

Unter dem Begriff "**Aspektorientierte Programmierung**" (AOP) wurde 1997 von Kiczales et al. [Kiczales+ 97] ein neues programmiersprachliches Konzept eingeführt, durch das die Probleme sogenannter *crosscutting concerns* gelöst werden sollen.

Zwei Aspekte (Anforderungen, Entwurfsentscheidungen oder dergl.) werden dann als *crosscutting* bezeichnet, wenn es mit herkömmlichen Mitteln nicht gelingt, beide Aspekte gleichermaßen

bei der Strukturierung zu berücksichtigen. Das häufigst genannte Beispiel sind die Aspekte "Funktionalität des Systems" und die zusätzliche Anforderung bestimmte Ereignisse in einer Datei zu protokollieren ("*logging*"). Aus der Verknüpfung beider Aspekte ergibt sich, dass *logging* nicht als eigenständiges Modul implementiert werden kann, sondern über das ganze System verstreut wird (*scattering*). Andersherum betrachtet werden viele Routinen des Systems eine Mischung aus Anwendungslogik und *logging* enthalten (*tangling*).

Durch *scattering* und *tangling* wird die Modulstruktur eines Systems aufgeweicht mit verheerenden Folgen für den Entwicklungsprozess und für die Softwarequalität. Wartung und Weiterentwicklung von Systemen setzen eine klare Modulstruktur voraus, weil sonst jede Änderung potentiell das gesamte System betrifft und dadurch Änderungen nicht unabhängig voneinander durchgeführt oder auch zurückgenommen werden können. Weiterhin sind unterschiedliche Varianten für verschiedene Kunden nicht beherrschbar, wenn sich die Unterschiede über zu große Teile des Quelltextes verteilen.

1.2 Schwachpunkte existierender Ansätze

Das Projekt TOPPrax gründet sich auf diese Erfahrungen und auf die Analyse der folgenden Schwachpunkte des noch jungen Ansatzes:

- Bisherige Sprachen sind z.T. Ad-Hoc-Lösungen, insofern sie bewährte Prinzipien des Sprachdesigns zugunsten neuer Ausdrucksmächtigkeit vernachlässigen.
- Bisherige Sprachen sind z.T. fixiert auf eine bestimmte Klasse von Problemen. Dadurch ist fragwürdig, in wieweit sie als allgemeine Programmiersprachen dienen können.
- Werkzeugunterstützung für aspektorientierte Softwareentwicklung beschränkt sich derzeit noch auf das allernötigste - häufig nicht viel mehr als ein Compiler.
- Es ist noch unklar, wie ein Softwareentwurf unter Berücksichtigung aspektorientierter Techniken aussehen soll. Gleiches gilt für die Dokumentation von aspektorientierten Programmen.
- Metriken für die Feststellung (bestimmter Aspekte) von Softwarequalität sind noch nicht auf die neuen Techniken angepasst.
- Aspektorientierung konkurriert mit der Technik des Refactoring, insofern, als beide Ansätze versprechen, die Evolution von Software zu begünstigen. Auch wenn beide Ansätze sich konzeptionell sehr gut ergänzen können, ist noch unklar, wie sie zusammen eingesetzt werden können.
- Es existiert noch keine umfassende Methodik der aspektorientierten Softwareentwicklung. Bislang dominiert die technikorientierte Sicht, die überwiegend verschiedene Lösungen diskutiert, ohne Hilfestellung zu geben, mit welcher Vorgehensweise eine gute Lösung systematisch entwickelt werden kann.
- Es fehlt noch an Belegen, dass der aspektorientierte Ansatz ökonomisch vorteilhaft ist, da die bekannten Beispiele zumeist recht klein und unzureichend ausgewertet sind.

1.3 Beiträge des Projektes

Durch die Auswahl von konsolidierten Programmiermodellen sollen Fehlentscheidungen der frühen Sprachentwürfe korrigiert werden. Ausgewählt für das Projekt wurden die Modelle Object Teams [Herrmann 02] und Caesar [Mezini, Ostermann 03], die darüber hinaus die Vorteile etlicher existierender Ansätze vereinen.

In dieser Ausgangssituation soll TOPPrax auf drei Ebenen zur Reifung des aspektorientierten Ansatzes beitragen:

1. Der Ansatz soll vervollständigt werden, indem auf eine umfassende Werkzeugunterstützung hingearbeitet wird, Zusammenhänge zu anderen Aktivitäten der Softwareentwick-

lung erarbeitet werden (z.B. Refactoring), sowie durch eine allgemeine Entwicklungsmethode.

2. Der Ansatz soll in realen Fallbeispielen eingesetzt werden. Kurzfristiges Ziel dieser Feldstudie ist das Aufdecken von Lücken in Technik und Methodik. Weiterhin dient der Praxiseinsatz als Grundlage für die anschließende Bewertung (s. nächster Punkt).
3. Der Praxiseinsatz soll nach unterschiedlichen Kriterien analysiert und bewertet werden. Unter dem Dach einer Analyse des wirtschaftlichen Nutzens sollen Fragen beantwortet werden wie:
 - Unter welchen Voraussetzungen sind die neuen Techniken einsetzbar?
 - Haben die aspektorientierten Lösungen eine höhere Qualität als herkömmliche?
 - Sind aspektorientierte Programme besser verständlich und weiterentwickelbar als herkömmliche?
 - Wie hoch liegen die Aufwände für Schulung, Entwicklung und Evolution?

Um den Praxiseinsatz bei den beteiligten Firmen vorzubereiten, werden umfangreiche Schulungsunterlagen erstellt, die universell einsetzbar sind:

- Grundlagenschulung zur Vermittlung der Sprachkonzepte und des Umgangs mit dem Compiler und dem Laufzeitsystem anhand einfacher Beispiele.
- Vertiefungsschulung zur Vermittlung von methodischen Konzepten und des Umgangs mit der integrierten Entwicklungsumgebung anhand komplexerer Beispiele.

1.4 Einfließende Ansätze

Die Programmiermodelle Object Teams [Herrmann 02] und Caesar [Mezini,Ostermann 03] sind Neuentwicklungen der TU Berlin und der TU Darmstadt, die verschiedene aktuelle Ansätze aus den Bereichen der konventionellen Objektorientierung und der Aspektorientierung aufgreifen, vereinen und verbessern. Alle verwendeten neuen Konzepte dienen dem Ziel, die Modularität komplexer Software zu verbessern, um damit Verständlichkeit, Wartbarkeit und Erweiterbarkeit der Software zu steigern.

Kollaborations-Module. Ansätze für den kollaborationsbasierten Entwurf führen Kollaborationen als Module ein, die das Zusammenspiel verschiedener Klassen ("Rollen") kapseln. Dies basiert u.a. auf der Beobachtung, dass Klassen als Module in großen Systemen zur Strukturierung nicht ausreichen und erlaubt erstmals eine direkte Umsetzung des kollaborationsbasierten Entwurfes im Sinne von z.B. Catalysis [DeSouza,Wills 98].

A-posteriori Integration. Subject-oriented programming (vgl. [Ossher,Tarr 01]) gehört zu den ersten Ansätzen, die sprachliche Unterstützung für die nachträgliche Integration von Modulen bereitstellen. Dies ist von zentraler Bedeutung, um die Unabhängigkeit von Modulen zu fördern: nur wo Adaptierung von Schnittstellen systematisch unterstützt wird, können unabhängig voneinander entwickelte Module gemeinsam eingesetzt und individuell weiterentwickelt werden.

Bidirektionale Schnittstellen. Architekturbeschreibungssprachen und Komponententechniken wie CCM (Corba Component Model) führen die konzeptionelle Unterscheidung von *expected* und *provided interfaces* ein. Der Effekt ist, dass auch ein client-Modul explizit Anforderungen an seinen Benutzungskontext stellen kann.

Vererbung von Kollaborationen. Die Sprachen beta bzw. gbeta [Ernst 99] führen das Konzept der *virtual classes* ein, das es erlaubt, eine Gruppe (Kollaboration) von sich gegenseitig referenzierenden Klassen in einem Schritt zu spezialisieren unter Vermeidung der Typprobleme, die andere Programmiersprachen in dieser Situation aufweisen (Kovarianz).

Flexible Kopplung. Prototypbasierte Sprachen wie Self [Ungar,Smith 87] bieten die Möglichkeit, eine Vererbungsbeziehung zwischen Objekten zu definieren, was in vielen Veröffentlichun-

gen auch als Rollenobjekte bezeichnet wird. Während Self eine klassenlose Programmiersprache ist, kombinieren neuere Ansätze wie Lava [Kniesel 99] die Klassenvererbung mit dem Konzept von Rollen mit Delegation. Die lose Kopplung, die hierdurch erreicht wird, bietet eine Flexibilität, die bei konventionellen Techniken nur mit erheblichem Mehraufwand möglich ist.

Nicht-invasive Adaptierung. Aspektorientierte Sprachen wie AspectJ [Kiczales+ 01] führen das Konzept des Aspekt-Webens (engl. *weaving*) ein, wodurch zusätzlicher Code in bestehende Klassen eingefügt werden kann, ohne jedoch deren Quelltext verändern zu müssen.

Dynamische Aspektaktivierung. Dynamische Aspektsprachen wie AspectS [Hirschfeld 02] erlauben das dynamische Aktivieren und Deaktivieren von Aspekten zur Laufzeit. Dies trägt der Beobachtung Rechnung, dass sich Software in verschiedenen Situationen unterschiedlich verhalten soll. Das explizite Ausprogrammieren von Bedingungen und Alternativen durchzieht in konventionell entwickelten Programmen die gesamte Struktur und wirkt sich dadurch negativ auf Verständlichkeit und Wartbarkeit aus. Aspektaktivierung konzentriert Fallunterscheidungen an sehr wenige Stellen im Programm und überlässt den Rest dem Laufzeitsystem (analog dem dynamischen Binden von Methoden).

1.5 Eigener Ansatz

Die gerade beschriebenen Ansätze werden in unterschiedlichem Grad in beiden von TU-Berlin und TU-Darmstadt entwickelten Programmiermodellen, Object Teams [Herrmann 02] und Caesar [Mezini,Ostermann 03], realisiert.

1.5.1 Konzepte

Beiden Programmiermodellen ist gemein, dass es neue Modulkonstrukte für Mengen kollaborierender Klassen gibt. Diese Modulkonstrukte haben ähnliche Eigenschaften wie Klassen (Instanziierung, Vererbung, Polymorphie). Anders als eine Klasse, die das Verhalten einer Menge definiert, beschreiben diese Module jedoch Mengen zusammenarbeitender Abstraktionen. A-posteriori-Integration dieser Module wird durch spezielle Bindungs- oder Konnektorkonstrukte realisiert, die dazu dienen, die Funktionalität des Moduls nicht-invasiv in neue Kontexte einzuflechten. Ferner stellen beide Programmiermodelle Konstrukte bereit, mit denen die Aktivierung gebundener Kollaborationsmodule zur Laufzeit geschehen kann.

Unterschiede zwischen den Programmiermodellen gibt es sowohl auf konzeptioneller Ebene als auch in der Implementierung. Durch sog. *Collaboration Interfaces* werden in Caesar die Bindungen von Kollaborationsmodulen von der konkreten Implementierung eines solchen Moduls getrennt, wodurch es möglich wird, Bindungen mit verschiedenen Implementierungen wiederzuverwenden. Weitere Unterschiede betreffen die Art, wie ein Kollaborationsmodul an einen neuen Kontext gebunden wird. In Object Teams wird hierzu eine eigene deklarative Subsprache verwendet, während in Caesar die Verbindung im Wesentlichen in "pure Java" realisiert wird.

Auf der Implementierungsebene liegt ein Unterschied darin, dass in Object Teams konventionelle Kompilation mit *weaving* zur Ladezeit verknüpft wird, während Caesar einen konventionellen Compiler mit dem *bytecode weaver* der Sprache AspectJ verknüpft. Beide Techniken erlauben es, Klassen und Pakete zu adaptieren, deren Quelltext nicht verfügbar ist. Das *load-time-weaving* der Object Teams-Implementierung erleichtert ferner das Konfigurationsmanagement dadurch, dass keine verschiedenen Varianten (mit und ohne Aspekte) derselben Klassen gespeichert werden müssen, da diese erst beim Programmstart erzeugt werden.

1.5.2 Beispiel Flugbuchungssystem

Ein Überblick über zentrale Konzepte wird durch das Beispiel in Abbildung 1 illustriert und mit den unter 1.4 vorgestellten Konzepten in Bezug gesetzt. Hier wird gezeigt wie ein existierendes

Flugbuchungssystem (Paket *domain*) um die zusätzliche Funktionalität eines Bonusprogramms ergänzt wird, ohne dass existierende Klassen verändert werden (**A-Posteriori-Integration**). Die Kollaboration *FlightBonus* (**Kollaborationsmodul**) implementiert dabei die neue Funktionalität basierend auf den Konzepten *Subscriber* (Teilnehmer am Bonusprogramm) und *BonusItem* (Element, für dessen Erwerb Bonuspunkte vergeben werden). Die *«adapt»* Beziehung zeigt, dass der zusammengesetzte Aspekt *FlightBonus* die *domain* adaptiert (**nicht-invasive Adaptierung**). Dabei wird *Subscriber* als Rolle von *Passenger*, und *BonusItem* als Rolle von *Segment* zugeordnet (**flexible Kopplung**). Auf Methodenebene (in diesem Diagramm nicht dargestellt) werden zwei Arten von Bindungen eingesetzt (**bidirektionale Schnittstelle**): einige in *FlightBonus* benötigte Methoden werden direkt an die zugeordneten Klassen aus der *domain* weitergeleitet. Andere Methoden werden in Aspektmanier in existierende Methoden der *domain* eingeflochten. Das Beispiel zeigt fernerhin, wie ein derartiger Geschäftsfall implementiert werden kann, so dass er für beliebige Domänen anwendbar ist: die Abstraktion *Bonus* implementiert die Interaktion zwischen *Subscribern* und *BonusItems*, ohne sich dabei auf Begriffe der Domäne zu beziehen. Durch konsistente Verfeinerung der Kollaboration *Bonus* (**Vererbung von Kollaborationen**) wird die wiederverwendbare Komponente an die spezielle Domäne angepasst. Da ein System u.U. die Bonusprogramme mehrerer Fluggesellschaften unterstützen soll und da sich Passagiere individuell für diese Programme registrieren müssen, ist die Aktivierung des Aspektes von verschiedenen Parametern abhängig, die erst zur Laufzeit des Systems bekannt sind (**dynamische Aspektaktivierung**). Die Unabhängigkeit der Kollaboration *Bonus* und der Domäne ist ein charakteristisches Merkmal der Ansätze Object Teams und Caesar, das mit klassisch objektorientierten Sprachen nicht erreicht werden kann. Die Unabhängigkeit soll die Grundlage für Wiederverwendung und erheblich verbesserte Wartbarkeit bilden.

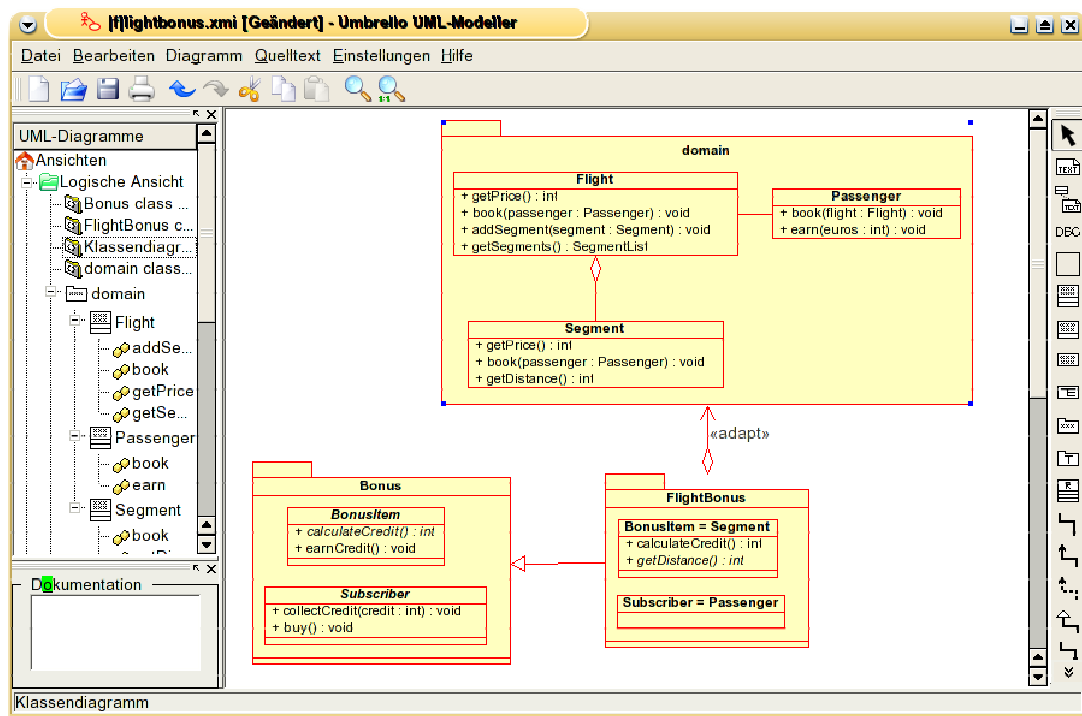


Abbildung 1: Beispiel *FlightBonus*

1.6 Fallstudie

Im Mittelpunkt des Projektes steht die Anwendung neuer Techniken im Projektgeschäft unter realen Bedingungen. Geplant sind je zwei Beispiele in den Sprachen Object Teams und Caesar.

1.6.1 Anwendungsdomänen

Der Gegenstand der Entwicklung wird in Abhängigkeit von der Auftragslage aus den Bereichen Logistik, Supply-Chain-Management und/oder Replenishment ausgewählt. Neben den inhaltlichen Bereichen werden folgende technische Aspekte bei den Fallstudien bearbeitet: servlet-basierte Webanwendungen, die Anbindung objektorientierter Applikationen an relationale Datenbanken sowie komponentenbasierte Systeme (EJB oder CCM).

1.6.2 Vorgehensweise

In einer initialen Phase müssen die aspektorientierten Lösungen redundant entwickelt werden, da das bestehende Entwicklungsrisiko und die Kundenvorgaben eine Auslieferung dieser Lösungen noch verbieten. Es wird sich dabei jeweils um die Parallelentwicklung zweier Lösungen handeln. Sollte dieses Vorgehen nicht über die gesamte Projektlaufzeit möglich sein, wird überbrückungsweise auf existierende Lösungen zurückgegriffen, die aspektorientiert umgestaltet werden. Bei der Entwicklung der aspektorientierten Lösung wird in vier Stufen vorgegangen:

1. Programmieren isolierter Beispiele, um den Umgang mit den Techniken zu trainieren
2. Anwenden der Techniken auf reale Aufgabenstellungen der redundanten Entwicklung
3. Umarbeiten der Ergebnisse von Schritt 2 nach realen Änderungswünschen
4. Durchführen eines Pilotprojektes mit Auslieferung der aspektorientierten Lösung in Absprache mit einem innovationsoffenen Kunden

Schritt 3 ist besonders wichtig, da die angestrebten Verbesserungen der aspektorientierten Techniken sich besonders darauf beziehen, dass entwickelte Software leichter geändert und weiterentwickelt werden kann.

Schritt 4 kann nur durchgeführt werden, wenn die vorigen Ergebnisse dazu geeignet sind, einen Kunden von den Vorteilen und der Beherrschbarkeit dieser Techniken zu überzeugen.

1.6.3 Evaluation

Unabhängig vom Vorgehen wird nach abgeschlossener Arbeit die aspektorientierte Lösung mit der herkömmlichen vergleichend evaluiert. Dabei werden unterschiedliche Kriterien berücksichtigt, wobei dem wirtschaftlichen Nutzen ein besonderes Gewicht zugesprochen wird:

- Qualität des Codes
- Verständlichkeit
- Wartbarkeit und Erweiterbarkeit
- Aufwände für Schulung, Entwicklung und Evolution

Für die zuverlässige Bestimmung von Eigenschaften des Codes werden z.T. Metriken benötigt. Im Projekt werden Metriken für aspektorientierte Programmierung ausgehend von objektorientierten Metriken entwickelt, evaluiert und durch Werkzeuge unterstützt.

1.7 Begleitende Forschung

Weiterentwicklung der Sprachen. Gemeinsamkeiten und Unterschiede von Object Teams und Caesar sollen zunächst auf konzeptioneller Ebene, später auch basierend auf ersten Erfahrungen im praktischen Einsatz herausgearbeitet werden.

Angestrebt wird anschließend eine Zusammenführung der jeweils besten Eigenschaften, damit nicht nur die methodischen Ergebnisse, sondern möglichst auch alle entwickelten Werkzeuge in einer einheitlichen Umgebung eingesetzt werden können.

Modellierung. Modellierungsnotationen sollen aus bestehenden Ansätzen kombiniert werden, so dass eine optimale Unterstützung von der Use-Case-Analyse bis hin zur Implementierung erreicht werden kann. Neben Standarddiagrammen der UML können hier zum Einsatz kommen:

- *Use-case-maps* [Buhr,Casselmann 96], die die initiale Kluft zwischen Use-Cases und detaillierteren Modellen überwinden können.
- Theme/UML [Clarke 02] bzw. UML for Aspects (Arbeitstitel: UFA) [Herrmann 02a] für die Erfassung von Aspekt-Modulen.
- Varianten von *message-sequence-charts* (z.B. *life sequence charts*), durch die das dynamische Verhalten jeweils eines Aspekt-Moduls möglichst präzise erfasst werden soll.

Methodik. Begleitend zu der Entwicklung der Fallstudien sollen Entscheidungsprozesse bei der Modellierung beobachtet und festgehalten werden. Diese Beobachtungen sollen mit bewährten Methoden der objektorientierten Entwicklung kombiniert werden, und münden in ein Methodenhandbuch für den Einsatz aspektorientierter Techniken.

Qualitätssicherung. Testverfahren und -werkzeuge müssen auf die neuen Sprachen ausgedehnt werden. **Metriken** für objektorientierte Programme sollen zunächst syntaktisch auf die neuen Sprachen ausgedehnt werden. Im zweiten Schritt sollen dann spezialisierte Metriken entwickelt werden, die den angemessenen Einsatz der neuen Konstrukte messen sollen.

2 Projektstatus

In den ersten Monaten des Projektes wurden vorrangig drei Bereiche bearbeitet:

- Entwicklung je einer integrierten Werkzeugumgebung für beide Programmiersprachen
- Schulung für Mitarbeiter der Firmen GEBIT und Daedalos
- Definition der ersten Fallstudie

2.1 Werkzeugumgebungen

Für beide Programmiersprachen, ObjectTeams/Java und CaesarJ, lagen bei Projektbeginn prototypische Compiler vor. Bereits erste Kontakte mit den Industriepartnern haben unterstrichen, dass dies für einen produktiven Einsatz der Sprachen in keinsten Weise ausreichen kann. In enger Absprache zwischen Werkzeugentwicklern und den Anwendern der Werkzeuge wurde eine Prioritätenliste aufgestellt, die im Laufe des Projektes abgearbeitet wird, wobei die dringendsten Wünsche bereits umgesetzt wurden.

2.1.1 Integration des Compilers

Im ersten Schritt wurden die Compiler in die von den beteiligten Firmen verwendete Entwicklungsumgebung Eclipse integriert. Diese Integration existiert nun in zwei unterschiedlichen Formen: der CaesarJ-Compiler sowie der prototypische ObjectTeams/Java-Compiler sind jeweils lose integriert, in der Form, dass lediglich der Aufruf des ansonsten eigenständigen Compilers automatisiert wurde. Die Informationen für Konfigurationsdateien und/oder Aufrufparameter werden dabei aus den Einstellungen in Eclipse übernommen. Für ObjectTeams/Java ist ein konsolidierter Compiler in Entwicklung, der eine direkte Erweiterung des Eclipse-eigenen Java-Compilers darstellt. Dieser Compiler wird noch enger in die Umgebung integriert sein, so dass insbesondere die komfortablen Mechanismen des inkrementellen Kompilierens direkt auf Ob-

jectTeams/Java übertragbar werden. Für die Konsolidierung des Compilers wurde im Projekt eine umfangreiche Testsuite erstellt, deren nahezu 700 Tests überprüfen, ob der Compiler mit der Sprachdefinition konform ist.

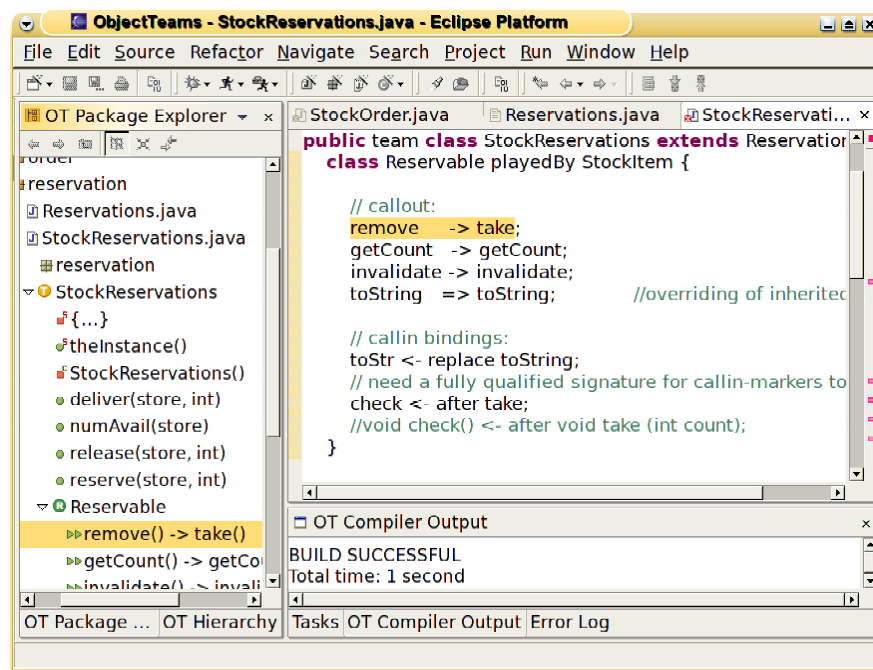
2.1.2 Unterstützung beim Editieren und Navigieren

Für beide Sprachen wurden *wizards* entwickelt, mit denen die jeweils neuen Elemente wie *Teams* oder *Collaboration Interfaces* erzeugt werden können, bzw. bei der Erstellung eines neuen Projektes werden technisch notwendige Voreinstellungen für Eclipse gesetzt.

Da es in der Entwicklungspraxis unerlässlich ist, sich durch Navigation innerhalb des Quelltextes Informationen zu beschaffen, wurde bereits einiges an Unterstützung hierfür realisiert. Hierunter fällt die Erweiterung des *package explorers* und eine neue Hierarchiesicht, die notwendig wurde, da das Konzept der virtuellen Klassen eine zweite Vererbungsdimension einführt.

Im Hinblick auf die aspektorientierten Anteile der Sprachen ist es insbesondere wichtig, dass sich ein Entwickler einen Überblick verschaffen kann, welche Methoden des Basisprogramms durch Aspekte betroffen sind, was in Object Teams z.B. durch sogen. *callin* Bindungen erreicht wird. Die erweiterte Entwicklungsumgebung kann diese Information bereits durch Markierungen am Rande des Editorfensters darstellen und der Entwickler kann über diese Markierungen von einer Basis-Methode zu den Aspekten navigieren, die hier wirksam sind.

Weitere Features der Entwicklungsumgebung, wie Hervorhebungen der Syntax sowie automatische Vervollständigung unter Berücksichtigung der neuen Strukturen sind in Arbeit.



2.1.3 Modellierungswerkzeug

Ein prototypisches Modellierungswerkzeug wurde entwickelt, das es erlaubt, UML-Klassendiagramme mit einigen Object Teams spezifischen Ergänzungen (UML For Aspects – UFA) zu erstellen (siehe Abbildung 1). Dieses Werkzeug basiert auf dem quelloffenen Programm *umbrello*. Da letzteres auch die Generierung von Quelltext aus dem Modell unterstützt, muss die-

se Funktionalität nur noch um Object Teams spezifische Elemente erweitert werden, so dass auch UFA Diagramme direkt für die Programmentwicklung eingesetzt werden können.

2.2 Schulung

Zur Vorbereitung des Praxiseinsatzes von Object Teams bei der Firma GEBIT Solutions wurde eine umfassende Schulung durchgeführt. Bei der Entwicklung des Schulungskonzeptes wurde zunächst festgelegt, wie die unterschiedlichen Sprachkonzepte in mehrere Lehreinheiten aufgeteilt werden sollten. Diese sollten von den Grundlagen (Grundlagenschulung) bis hin zu komplexeren Anwendungstechniken (Vertiefungsschulung) der Object Teams Konzepte eine grundlegende Einführung in die Sprache gewähren. Das Resultat war eine Einteilung in vier Schulungseinheiten mit einer Dauer von je ca. 4 Stunden:

1. Allgemeine Einführung und Motivation, sowie knappe Darstellung der wichtigsten Kernkonzepte anhand eines einfachen, aber umfassenden Beispiels
2. Aufbau und Verwendung von Kollaborations-Modulen, Besonderheiten bei der Vererbung von Kollaborationen und darin enthaltener Rollen
3. A-Posteriori-Integration von zusätzlicher Funktionalität in eine bestehende Anwendung mit Hilfe flexibler Kopplung von Rollen an bestehende Anwendungsobjekte
4. Object Teams Lösung spezieller Kontrollflussmuster, dynamische Aspekt-Aktivierung zur Flexibilisierung des Einflusses von Adaptionen

Neben einer theoretischen Einführung der Sprachkonzepte bestand ein wesentlicher Teil der Schulung darin, den Teilnehmern (GEBIT-Mitarbeiter u. Studenten) den praktischen Umgang mit Object Teams (inkl. Compiler und Laufzeitsystem) näher zu bringen. Hierzu wurden sowohl Beispiele, als auch Übungsaufgaben verwendet.

Etwa ab der Hälfte der Schulung konnte das Alpha-Release der für Object Teams erweiterten integrierten Entwicklungsumgebung Eclipse eingeführt und verwendet werden.

Die materiellen Ergebnisse der Schulung bestehen aus Vortragsfolien, sowie Unterlagen zu Beispielen (Code, Modelle) und Übungsaufgaben (Aufgabenstellungen, Code-Vorgaben, Musterlösungen), die universell einsetzbar sind.

3 Erfahrungen und Bewertung

Bei der **Schulung** zeigte sich, dass die Konzepte der neuen Sprachen in den Firmen interessiert aufgenommen wurden. Die Aufgaben der Schulung inspirierten die Entwickler zu eleganten Lösungen, wie sie mit traditionellen Techniken nicht möglich sind. Auch wenn der Lernaufwand nicht unterschätzt werden sollte, konnten die Entwickler die zugrundeliegenden Metaphern schon nach wenigen Schulungsterminen eigenständig beim Entwurf anwenden. **Rückmeldungen** aus den Firmen haben andererseits unterstrichen, dass ein innovativer Sprachentwurf allein nicht ausreicht, sondern dass vielmehr eine detaillierte Ausarbeitung der Sprachen notwendig ist, um die neuen Elemente beliebig mit bekannten Techniken von Java einsetzen zu können. Dies erforderte einige pragmatische Kompromisse, wo bestimmte Eigenschaften der Sprache Java aus konzeptueller Sicht kritisiert werden. Da Java zunächst in seiner existierenden Form als gegeben betrachtet werden muss, müssen neue Elemente auch mit unliebsamen Eigenschaften der Basissprache harmonieren. Beispiele hier sind Konstruktoren, Sichtbarkeits-Modifizierer sowie das Schlüsselwort *static*, die alle im wissenschaftlichen Kontext als unzulänglich diskutiert werden.

Die detaillierten Festlegungen im Sprachentwurf werden kontinuierlich in präzise **Sprachdefinitionen** eingepflegt, die in dieser Form ein wichtiges Produkt des Projektes sind, das u.a. die Schnittstelle zwischen der Werkzeugimplementierung und den Anwendern der Sprache bildet.

3.1 Integration mit existierenden Frameworks

Aus den ersten Entwürfen zu den Fallbeispielen wird bereits deutlich, dass neben Sprachen, Modellen und Werkzeugen noch ein weiterer Bereich betrachtet werden muss: beide Firmen setzen bei der Entwicklung verschiedene Komponenten, Frameworks und Generatoren ein, die sich um Aspekte wie Persistenz der Daten kümmern. Grundsätzlich ist es natürlich möglich, Programme in ObjectTeams/Java und CaesarJ mit beliebigen Java-Bibliotheken zu binden. Allerdings wurde dabei übersehen, dass einige dieser Frameworks tatsächlich die neuen Programmelemente aktiv unterstützen müssen. Beispielsweise sollte ein Persistenzframework für Object Teams tatsächlich Teaminstanzen und Rolleninstanzen gesondert behandeln, was u.U. eine enge Integration mit der Laufzeitumgebung bzw. internen Struktur von ObjectTeams/Java-Programmen erfordert. Eine Erweiterung eines Persistenzframeworks für Object Teams ist derzeit in Planung.

4 Ausblick

Zum aktuellen Zeitpunkt sind alle wesentlichen Vorarbeiten geleistet und der Einsatz der neuen Techniken in den Fallstudien beginnt. Die Werkzeugumgebungen bieten Unterstützung bei der Entwicklung, die über den Stand üblicher Forschungsprototypen weit hinausgeht. Während auf der einen Seite Werkzeuge und Konzepte kontinuierlich vervollständigt werden, beginnt nun als zentrale Aufgabe des Projektes die Erarbeitung einer Methode, die zukünftigen Entwicklern als Wegweiser für den Einsatz der neuen Techniken dienen soll. Am Ende des Projektes wird bewertet werden, ob Sprache, Werkzeuge und Methode zusammen einen spürbaren, ökonomischen Vorteil bei der Softwareentwicklung und -wartung versprechen.

Literatur

- Buhr,Casselmann 96** Buhr, Casselman: *Use Case Maps for Object-Oriented Systems*, Prentice Hall, 1996.
- Clarke 02** Siobhán Clarke: *Extending standard UML with model composition semantics*, Journal Science of Computer Programming, Elsevier Science, 2002.
- Ernst 99** Erik Ernst: *gbeta - a Language with Virtual Attributes, Block Structure, and Propagating, Dynamic Inheritance*, Dissertation Universität Aarhus, 1999.
- DeSouza,Wills 98** D. De Souza, A. Wills: *Objects, Components, and Frameworks with UML -- The Catalysis Approach*, Addison Wesley, 1998.
- Ernst,Lorenz 03** E. Ernst, D. Lorenz: *Aspects and polymorphism in AspectJ*, Proc. AOSD, 2003.
- Herrmann 02** Stephan Herrmann: *Object Teams: Improving Modularity for Crosscutting Collaborations*, Proc. Net.ObjectDays, Erfurt, 2002.
- Herrmann 02 a** Stephan Herrmann: *Composable Designs with UFA*, Proc. Workshop on Aspect-Oriented Modeling with UML, AOSD, 2002.
- Hirschfeld 02** Robert Hirschfeld: *Aspect Oriented Programming with AspectS*, Proc. Net.ObjectDays, 2002.
- Kiczales+ 01** G. Kiczales, E. Hisdale, J. Hugunin, M. Kersten, J. Palm: *An Overview of AspectJ*, Proc. ECOOP, 2001.
- Kiczales+ 97** G. Kiczales, J.Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J.-M. Loingtier, J. and Irwin, *Aspect-Oriented Programming*, Proc. ECOOP, 1997.
- Kniesel 99** Günter Kniesel: *Type-Safe Delegation for Run-Time Component Adaptation*, Proc. ECOOP, 1999.
- Mezini,Ostermann 03** M. Mezini, K. Ostermann: *Conquering Aspects with Caesar*, Proc. AOSD, 2003.
- Ossher,Tarr 01** H. Ossher, P. Tarr: *Multi-Dimensional Separation of Concerns and The Hyperspace Approach*, in: M. Aksit (Ed.): *Software Architecture and Component Technology: State of the Art in Research and Practice*, Kluwer, 2001.
- Ungar,Smith 87** D. Ungar, R. Smith: *Self: the power of simplicity*, Proc. OOPSLA, 1987.