

PARADIGMENWECHSEL: WAS KOMMT NACH DER OBJEKTORIENTIERUNG?

Der letzte Paradigmenwechsel in der Softwareentwicklung ist 20 Jahre her und führte uns von der strukturierten Programmierung zu objektorientierten Technologien. Wird es langsam Zeit für einen neuen Wechsel? Wohin geht die Reise? Und wie kommen wir dorthin? Diesen und anderen Fragen wird in diesem Artikel nachgegangen.

Historische Betrachtung

Auf die maschinennahe Programmierung folgte bald die strukturierte Programmierung. In den 80er Jahren begann die Entwicklung der Objektorientierung, die ihrem Durchbruch in den 90ern hatte. Jeder dieser Paradigmenwechsel brachte Verbesserungen in der Strukturierbarkeit von Softwaresystemen mit sich. Am Anfang jedes Wechsels standen Weiterentwicklungen im Bereich der Programmiersprachen. Wichtigstes Merkmal dabei: Die Sprachen steigern mit jeder Evolution die Abstraktionsebene des Entwicklers:

- Statt Maschinenregister werden Variablen verwendet.
- Statt konkreten Einsprung- und Rücksprung-Adressen ruft man Methoden auf und verwendet IF-, FOR- und WHILE-Ausdrücke.
- Statt Zeiger auf Adressbereiche verwendet man Datenstrukturen und Objekte.
- Statt expliziter Speicherverwaltung verwendet man heute automatisches *Garbage Collecting*.

Doch nur die Sprache alleine schafft den Paradigmenwechsel nicht. Ausgehend von der Sprache müssen Methoden und Werkzeuge entstehen, die die neue Technologie begreifbar und beherrschbar machen. So entstanden aus der objektorientierten Programmierung objektorientierte Analyse und objektorientiertes Design (OOA und OOD) oder allgemein die objektorientierte

Softwareentwicklung mit fundierten Konzepten. Parallel dazu erleichtern Tools die Arbeit im neuen Paradigma. Seien es integrierte Entwicklungsumgebungen (IDEs) mit ausgefeilten Code-Editoren und Klassen-Browsern oder grafische Werkzeuge wie für Strukturierte Analyse/Design (SA/SD) in den 80ern und die CASE-Tools der 90er.

Die Voraussetzungen für einen Paradigmenwechsel sind:

- bessere Strukturierbarkeit von Softwaresystemen
- höhere Abstraktion der Sprache
- Entwicklung formaler Methoden
- Werkzeugunterstützung

Sprachvielfalt

Da Paradigmenwechsel zunächst auf Sprachebene beginnen, wollen wir zunächst einen Blick auf die aktuelle Situation bei den Programmiersprachen werfen. Nachdem das neue Jahrtausend sehr ereignislos bezüglich Sprachentwicklung begann, gibt es derzeit eine Explosion an Sprachvielfalt. Parallel zum Java- und .NET-Mainstream sind dynamische Sprachen derzeit in aller Munde (z. B. Groovy, Ruby und Python). Insbesondere die Anstrengungen, diese Sprachen in die bestehenden Plattformen zu integrieren, machen sie sehr attraktiv. Aber bringen sie wirklich eine höhere Abstraktion? Kaum. Lässt sich Software mit ihnen besser struk-



Achim Demelt

(E-Mail: a.demelt@excellent.de)

ist Softwarearchitekt bei der eXXcellent solutions gmbh in Ulm. Seine Arbeitsschwerpunkte liegen in der wirtschaftlichen Nutzung aktueller Technologien im Bereich modellgetriebene und aspektorientierte Softwareentwicklung.



Dr. Martina Maier

(E-Mail: m.maier@excellent.de)

ist Geschäftsführerin der eXXcellent solutions gmbh in Ulm. Ihr Schwerpunkt sind effiziente Softwareentwicklungsvorgehen.

turieren? Eher nicht. Methoden? Fehl-anzeige. Werkzeugunterstützung? Eher schlechter, da die (statisch arbeitenden) Tools mit der Dynamik zur Laufzeit nicht umgehen können.

Ein anderer Ansatz ist die funktionale Programmierung. Dieser ist vergleichsweise alt – es gibt ihn bereits seit den 70ern –, durch Sprachen wie *Erlang* und *Haskell* erlangt er aber derzeit wieder mehr Popularität: Statt Strukturierung mit Objekten wird hier Strukturierung mit Funktionen betrieben. Inhärent zustandslos lassen sich funktionale Programme damit problemlos auf mehrere Prozessoren oder Rechner verteilen. Das in manchen Bereichen immer wichtiger werdende Problem der Skalierung von Softwaresystemen wird damit gelöst. Der Entwickler arbeitet auf einer anderen Abstraktionsebene (den Funktionen) und muss sich um Skalierung nicht mehr kümmern. Formale Methoden gibt es mit dem *Lambda Calculus* seit beinahe 30 Jahren. Drei Voraussetzungen für einen Paradigmenwechsel sind also erfüllt: Strukturierung, Abstraktion und formale Methoden, wenn da nicht die Werkzeugunterstützung wäre. Komfort à la Eclipse-



Abb. 1: Welches Paradigma kommt nach OOP?

Entwicklungsumgebung oder grafische Werkzeuge sind nicht vorhanden.

Und last, but not least: Die domänenspezifischen Sprachen (*Domain Specific Language, DSLs*) nehmen von vornherein einen anderen Weg: Ausgehend von der Problemstellung definiert man sich seine eigene Sprache mit der genau für dieses Problem passenden Abstraktionsebene. Ob sich die Software dann auch noch gut strukturieren lässt, hängt ein bisschen von der Ausprägung der DSL ab: Bei internen DSLs (beliebt in *Ruby*) ist man im Wesentlichen an die bestehende Sprachplattform gebunden. Bei externen DSLs hat man freie Hand, wie man diese strukturiert und verwendet. Natürlich müssen noch passende Generatoren und/oder Interpreter für die DSL geschrieben werden, um sie auf bestehende Sprachplattformen abzubilden. Der Entwickler, der mit der DSL arbeitet, merkt davon aber nichts. Er hat eine schöne, abstrakte und strukturierte Sicht der Dinge vor sich. Apropos Sicht: DSLs eignen sich ebenso gut für grafische Visualisierung und für textliche Darstellung. Entsprechende Werkzeuge und Bibliotheken, wie *EMF*, *GMF* und *xText*, machen diesen Ansatz bereits jetzt praxisreif. Der einzige Nachteil von DSLs ist, dass jede Sprache anders ist. Formale Methoden können bestenfalls für der Erstellung der DSL existieren. Für die Methodik der Verwendung der Sprache ist man auf den Sprachdesigner angewiesen.

Quereinsteiger

Ein Kandidat, der bereits im Namen seinen Anspruch auf einen Paradigmenwechsel kundtut, ist die *Aspektorientierung (AO)*. Sie erlaubt es, querschnittshafte und übergreifende Teile eines Softwaresystems (*Crosscutting Concerns*) zu erfassen und zu modularisieren. Dieser modularisierte, so genannte „Aspekt“ wird dann mit dem Rest des Systems zusammengebracht („verwoben“), um so ein laufendes Gesamtsystem zu erzeugen. Teile, deren Code mit herkömmlichen Mitteln über das gesamte System verteilt wären, werden mit der

Aspektorientierung einzeln und modular entwickelt (*Separation of Concerns*). Die Software wird in Summe strukturierter und wartbarer.

- *Dynamische Sprachen*: Hype mit wenig Potenzial für substanziellen Wandel
- *Funktionale Programmierung*: viel versprechender Ansatz mit mangelnder Werkzeugunterstützung
- *Domänenspezifische Sprachen (DSLs)*: Abstraktion nach Maß mit babylonischer Sprach- und Methodenvielfalt
- *Aspektorientierte Programmierung (AOP)*: besser strukturierte Software, aber ohne inhärenten Abstraktionsgewinn

Kasten 1: Kandidaten für einen Paradigmenwechsel.

Bekanntester Vertreter der aspektorientierten Programmierung ist sicherlich *AspectJ*, eine AO-Spracherweiterung für Java. Entstanden am Xerox PARC und mittlerweile auf Eclipse.org weiterentwickelt gibt es nun auch bereits recht gute Werkzeugunterstützung. Obwohl ausgereift und außerordentlich vollständig, führt *AspectJ* noch ein Nischendasein, wobei der Ansatz, neue Konzepte als Erweiterung einer bekannten Mainstream-Sprache zu platzieren, sicherlich kein schlechter ist, wie der Erfolg von C++ auf Basis von C gezeigt hat.

Bessere Strukturierung und gute Tool-Unterstützung allein genügen aber scheinbar

nicht, um den mangelnden Abstraktionsgewinn wettzumachen. Die aspektorientierte Programmierung bewegt sich im Wesentlichen noch auf derselben Abstraktionsebene wie die objektorientierte Programmierung. Lediglich die Möglichkeiten zur Komposition von Softwaresystemen werden verbessert. Außerdem kommt die Definition formaler Methoden der Aspektorientierung erst langsam in Gang. Publikationen außerhalb von *AspectJ*-Büchern findet man kaum und die universitäre Forschung ist noch nicht weit fortgeschritten.

Fazit

Keines der vorgestellten Konzepte ist zum jetzigen Zeitpunkt bereit für einen Paradigmenwechsel. Dem einen fehlt Struktur, dem anderen Abstraktion, dem dritten Methoden oder Werkzeuge. Vielversprechender könnte jedoch bereits heute die Kombination mehrere Konzepte sein. Warum nicht den Strukturgewinn von AO mit der Abstraktion von DSLs verbinden? Erste Ansätze hierfür existieren bereits, beispielsweise beim *MDSD-Framework (Model-Driven Software Development)* von openArchitectureWare. Hier können beispielsweise Generator-Templates für die Generierung von Code aus formalen Modellen durch Aspekte strukturiert und erweitert werden. Genauso können Werkzeuge Integratoren zwischen abstrakter grafischer Beschreibung und klassischer Codierung werden. Es gilt, diese Ideen in wirtschaftlich nutzbare Innovationen umzusetzen – die Zeit hierfür ist reif. ■

OBJEKTSPEKTRUM ist eine Fachpublikation des Verlags:

SIGS DATACOM GmbH · Lindlaustraße 2c · 53842 Troisdorf
 Tel.: 02241/2341-100 · Fax: 02241/2341-199
 E-mail: info@sig-datacom.de
 www.objektspektrum.de
 www.sigs.de/publications/aboservice.htm

SIGS DATACOM
 FACHINFORMATIONEN FÜR IT-PROFESSIONALS