Buchbesprechung „Object-Oriented Software-Composition“ von Oscar Nierstrasz und Dennis Tsichritzis

Alexander Fronk
Lehrstuhl für Software-Technologie
Universität Dortmund
e-mail: fronk@ls10.informatik.uni-dortmund.de

23. Mai 1997

Neben einer Einleitung in die Thematik besteht das Buch aus vier Teilen, die sich jeweils verschiedenen Anwendungsbereichen widmen: *Nebenläufigkeit und Verteilung, Spezifikation und Komposition, Software-Informationsmanagement, Frameworks und Applikationen.* Jeder Teil besteht seinerseits aus jeweils zwei bis drei Artikeln. Der Leser erwartet – glaubt er dem Titel – über Komponenten und ihre Komposition informiert zu werden und wie sich diese Begriffe in den genannten Bereichen umsetzen lassen. Die Artikel sind Forschungsberichte, die schnell in die entsprechenden Spezialgebiete abdriften. Das zeigt zwar die Tragweite der Komponenten-orientierten Software-Komposition, der Leser verliert jedoch schnell den Zusammenhang zum postulierten Thema.

Betrachtet man die Artikel näher, so stellt man fest, daß sie eher die Notwendigkeit motivieren, vom (engen) objekt-orientierten Begriffsdenken auf einen flexibleren und skalierbaren Komponentenbegriff zu kommen, um insb. die Wiederverwendung von Software zu verbessern. Der Leser stellt ebenfalls fest – und die Autoren weisen in ihrer Einleitung darauf hin –, daß der Komponentenbegriff je nach Anwendungsgebiet definiert werden (können) muß. Gleichzeitig geben sie aber eine Definition an („static abstraction with plugs“), die lediglich eine Abstraktion des Komponentenbegriffs mit Eigenschaften bedeutet und erst mit Leben gefüllt werden muß. Es macht wenig Sinn, eine solche generische Begriffsbildung vorzunehmen. Die technologisch und methodisch schwierige Umsetzung des Komponentenbegriffs wird durch diese Verallgemeinerung keineswegs vereinfacht. Vorschläge zur Komponentenbildung, die auf Erfahrungen beruhen (wie sie Vicki de Mey in ihrem Artikel gibt), sind sicherlich hilfreicher.

Die Artikel im einzelnen:

Die beiden Artikel im Teil II, *Nebenläufigkeit und Verteilung*, beschreiben bestehende Probleme und Lösungsansätze des objekt-orientierten Paradigmas mit dieser Thematik.

In seinem Beitrag *Nebenläufigkeit in objekt-orientierten Programmiersprachen* zeigt Michael Papathomas, daß verschiedene Ansätze existieren, Nebenläufigkeit in OO-Sprachen zu integrieren. Er weißt darauf hin, daß die Entwicklung solcher Sprachen zwar der Wiederverwendung von Software(-Komponenten) in größerem Stil dient und daß dieses Ziel in die Sprachentwicklung einfließen muß. Dabei spricht er jedoch stets von Objekten. Komponente also gleich Objekt?

Mit *Interpretation of Object-Oriented Applications* beschäftigt sich Dimitri Konstantas. Er stellt den wichtigen Vorteil der Wiederverwendung von Software-Moduln heraus, den objekt-orientiertes Design und entsprechende Entwicklungsmethoden bieten. Wegen ihrer Vielzahl und der damit verbundenen Orthogonalität werden Möglichkeiten gesucht, Interoperabilität von Modulen aus verschiedenen Sprachen zu unterstützen. Modul gleich Objekt gleich Komponente?

Im Teil III, *Spezifikation und Komposition*, werden theoretische Aspekte der Thematik beleuchtet.

Oscar Nierstrasz zeigt in *Regular Types for Active Objects* einen Forschungsaspekt im Bereich der Typtheorie objekt-orientierter Programmiersprachen auf, der im Gegensatz zu anderen Ansätzen Nebenläufigkeit und Aspekte der Aufgabendelegation und Kommunikation zwischen zustandsbasierten Objekten behandelt. Dabei geht er beispielsweise der Frage nach, welche Dienste ein Objekt in welchem Zustand anbietet.

Der Artikel *A Temporal Perspective of Composite Objects* beschäftigt Constantin Arapis mit der Frage, welche Möglichkeiten für das System-Design existieren, die zeitliche Reihenfolge von Nachrichten von und an Objekte zu beschreibe; den Leser hingegen beschäftigt der Artikel mit der Suche nach der Auswirkung auf Komponenten-orientiertes Denken. Da hilft auch sein Ansatz kaum weiter, die Problematik für kooperierende Objekte in „part-of“-Beziehung mit temporaler Aussagenlogik in den Griff zu bekommen. Arapis definiert die „temporale Komponente“ einer oo-

Design-Methode als die Sammlung von Begriffen und Konzepten zur Beschreibung temporaler Aspekte des Objektverhaltens. Komponenten einer Klasse sind Instanzen von Klassen, die in der umgebenden Klasse verwendet werden. Die Komposition betrachtet das Zusammenspiel, die Kommunikation dieser Objekte.

Das Kompatibilitätsproblem versucht Laurent Dami in *The λ_N-Calculus* durch ein Interaktionsprotokoll zwischen Software-Komponenten zu lösen . Durch die Erweiterung des λ-Kalküls tragen Subtypisierungen dazu bei, Kompatibilität modellieren zu können. Anbei wird eine Erweiterung zur Erfassung nebenläufiger Konstrukte vorgeschlagen. Die erstgenannte Erweiterung ermöglicht es, Objekte im λ-Kalkül codieren zu können. Auch hier wird der Leser im Unklaren darüber gelesen, wie eine Komponente zu erkennen und ihre Komposition zu verstehen ist.

Teil IV, *Software Information Management*, gibt Einblick in die Problematik der Konzepte und Tools zur Wiederverwendung von Software.

Software-Repositories als Schlüsselkomponente bei der Erstellung von Software durch Wiederverwendung werden von Panos Constantopoulos und Martin Dörr in ihrem Artikel *Component Classification in the Software Information Base* diskutiert. Ihrer Meinung nach unterstützen objekt-orientierte Sprachen die Entwicklung wiederverwendbarer Klassen, die hier als Komponenten bezeichnet werden. Die Autoren stellen ein Repositiory-System vor, mit dessen Hilfe entstehende Probleme wie Repräsentation der Klassen, Darstellung struktureller und beschreibender Information, der Beziehung zwischen den Klassen, die Auswahl und das Verständnis der Klassen sowie ihre Evolution gelöst werden können. Komposition wird hier als Akt der Wiederverwendung verstanden.

Die Handhabung der Klassenevolution in objektorientierten Systemen betrachtet Eduardo Casais etwas genauer und schlägt eine adäquate Methode vor, die gängigen Probleme der objekt-orientierten Entwicklung zu lösen. Auch hier nimmt das Design und die Entwicklung von Software-Komponenten (aber was sind solche) eine wichtige Rolle für das Ziel der Wiederverwendung ein.

Ein Tool zur grafischen Repräsentation von Objektbeziehungen sowie die dazu notwendigen Voraussetzungen beschreibt Xavier Pintado in seinem Artikel über den *Affinity Browser*. Damit soll ein Beitrag zur Visualierung, also auch zum besseren Verständnis von Klassen, ihrer Hierarchien, Interaktionen von Komponenten etc. geleistet werden. Die Wiederverwendung, Fehlerbeseitigung, das Tuning etc. für große objektorientierte Systeme soll damit unterstützt werden.

Das Buch schließt in Teil V, *Frameworks and Applikations*, mit Beispielen zur Umsetzung der Erkenntnisse der ersten vier Teile und bildet mit Teil I einen Rahmen um einige wesentliche Aspekte der Software-Technologie.

Mit visueller Komposition von Software-Applikationen führt Vicki de Mey in den letzten Teil des Buches ein. Sie stellt ein Framework und einen darauf aufbauenden Prototypen zur visuellen Komposition offener Applikationen aus schnittstellenkompatiblen Software-Komponenten vor. Dabei werden interaktiv grafisch repräsentierte Software-Komponenten zu lauffähigen Anwendungen zusammengestellt. de Mey zeigt hier eine konkrete Anwendung im Multimedia-Bereich auf, die die Ideen der komponenten-orientierten Software-Entwicklung umsetzt. Abschließend gibt sie Vorschläge zur Umsetzung der Begriffe Komponente, Komposition und Visualisierung in konkrete software-technische Einheiten.

Simon Gibbs betrachtet den Nutzen der objektorientierten Technologie im Bereich der Multimedia-Programmierung. Er betrachtet die Begriffe der Komponente und ihrer Komposition für diesen Bereich und stellt ein Multimedia-Framework sowie eine darauf basierende Anwendung vor, die diese Begriffe umsetzt. Eine Komponente ist für Gibbs die Kapselung von Soft- und Hardwareprozessen, die Medienströme produzieren, konsumieren oder transformieren. Ihre Schnittstellen dienen auch hier der Kommunikation. Komposition bedeutet für ihn das Zusammensetzen schnittstellen-kompatibler Komponenten, sodaß ein Netzwerk aus erzeugenden, transformierenden und verbrauchenden Komponenten eine Anwendung bildet.

Im letzten Artikel *Gluons and the Cooperation between Software Components* zeigt Xavier Pintado einen Ansatz zur Standardisierung von Regeln bzgl. der Interaktion bzw. dem Erkennen von Interaktionsmustern von Komponenten, sodaß die Interaktionsprotokolle, die in speziellen, Gluonen genannte Objekte gekapselt werden, wiederverwendet werden können und die Kooperation von Software-Komponenten dadurch standardisiert werden. Das vorgestellte Entwicklungs-Framework für den speziellen Bereich der Finanzsoftware ermöglicht die dynamische Komposition, d.h. neue Komponenten können in ein bestehendes System integriert und zur Laufzeit komponiert werden.

Fazit: Die Artikel verdeutlichen einige Probleme des objekt-orientierten Paradigmas in verschiedenen Bereichen. Ein je nach Kontext – wie Teil V zeigt – definierter Komponentenbegriff ist nur thematisch sehr begrenzt für Wiederverwendung geeignet. Das Lesen des Buches gibt aber einen guten Einblick in den Lösungsversuch, einen kontextbezogenen Komponentenbegriff zur Software-Komposition heranzuziehen.

Oscar Nierstrasz, Dennis Tsichritzis
Object-Oriented Software-Composition
Prentice Hall, 1995