

Ein Nachdruck aus OBJEKTspektrum 3/97

# Bausteinorientierte Anwendungsentwicklung: Voraussetzungen, Anforderungen und Auswirkungen

Das Interesse an der bausteinorientierten Anwendungsentwicklung hat in den letzten Jahren stark zugenommen. Dieses neue Paradigma zur Erstellung von komplexen Informationssystemen besagt, daß eine Anwendung mehr oder weniger vollständig aus vorgegebenen, ausgetesteten Softwarebausteinen zusammengesetzt wird. Herrscht weitgehend Einigkeit über die fundamentalen Vorteile (etwa die schnellere Anwendungsbereitstellung und Wiederverwendung von Softwarebausteinen), so gibt es unterschiedliche Ansätze und Vorgehensweisen, diese Vision umzusetzen. Um eine erfolgreiche und effektive Anwendungsentwicklung auf der Basis von Bausteinen zu ermöglichen, sind bestimmte Voraussetzungen zu treffen, wie z. B. ein angepaßter Anwendungsentwicklungsprozeß. In diesem Artikel, der auf einem Projekt des „Informatikzentrums der Sparkassenorganisation“ basiert, wird eine Reihe elementarer Voraussetzungen beleuchtet. Insbesondere wird aufgezeigt, wie die ORB-basierten Technologien CORBA und COM/OLE zur bausteinorientierten Anwendungsentwicklung eingesetzt werden können und welche Vorteile sich hieraus ergeben.

Henning Heuer-Hasenpatt ist Mitentwickler von Standardkommunikationstechnologien zwischen OS/2-Entwicklungsumgebungen und einem IMS-Dialogsystem bei der Firma dvg in Hannover.

Dr. Bernhard Hollunder ist Technologieberater bei der Firma Interactive Objects Software GmbH mit Sitz in Freiburg. Seine Interessen liegen im Bereich des „Distributed Object Computing“.

Hans-Bernd Kittlaus ist Bereichsleiter im Modellierungskompetenzzentrum des SIZ in Bonn.

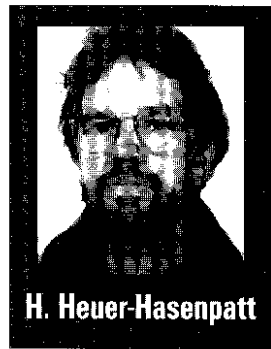
Norbert Schumacher ist im SIZ Bonn im Modellierungskompetenzzentrum für die Themen Softwarebausteine, Anwendungsintegration sowie Produktmanagement für Softwareprodukte zuständig.

\* Im folgenden werden die Begriffe Softwarebaustein, Baustein und Komponente synonym verwendet.

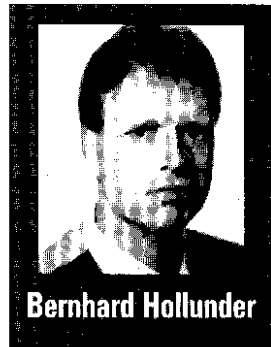
## Bausteinorientierter Anwendungsentwicklungs- prozeß

Bei einer geplanten und konsequenten Umsetzung der bausteinorientierten Vorgehensweise ergeben sich eine Vielzahl von Vorteilen. Die wohl wichtigsten sind:

- **Schnellere Anwendungsentwicklung und -bereitstellung:** Steht eine interessante Menge von eigen- und fremdentwickelten Softwarebausteinen\* zur Verfügung und lassen sich diese einfach zu Anwendungssystemen zusammensetzen, so läßt sich der Programmieraufwand bei der eigentlichen Anwendungsbereitstellung stark reduzieren.
- **Reduzierte Kosten:** Projekte produzieren – z. B. im Rahmen einer zu erstellenden Anwendung – wiederverwend-



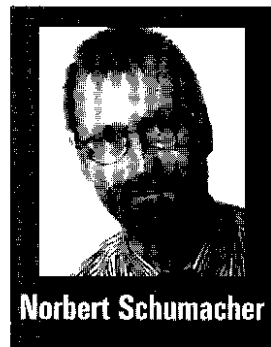
H. Heuer-Hasenpatt



Bernhard Hollunder



Hans-Bernd Kittlaus



Norbert Schumacher

bare Softwarebausteine. Können andere bzw. zukünftige Entwicklungsprojekte diese Bausteine nutzen, ergibt sich eine schnellere Amortisierung der Entwicklungskosten.

- **Qualitätssteigerung:** Der Einsatz ausgetesteter und qualitätsgesicherter Softwarebausteine unterstützt die Entwicklung von qualitativ hochwertigen Anwendungssystemen. Die verwendeten Bausteine werden im Laufe ihres Le-

benszyklus permanent erweitert und verbessert – ein guter Baustein „reift“ über Jahre hinweg und wird ständig wertvoller.

- **Reduktion der Komplexität:** Umfangreiche Anwendungssysteme sind aus einer Menge von strukturell einfacheren Teilen zusammengesetzt, die jeweils eine scharf umrissene Aufgabe im Gesamtsystem übernehmen. Gemäß der Strategie „dividere et imperare“ wird es erst möglich, langlebige und zuverlässige Informationssysteme zu bauen.
- **Flexible Anwendungssysteme und moderne Systemarchitekturen:** Anwendungssysteme, die konsequent auf der Basis von Bausteinen erstellt worden sind, zeichnen sich durch eine hohe Flexibilität aus. Lokale Änderungen (wie z. B. der Austausch eines Bausteins) haben auch nur lokale Auswirkungen im Gesamtsystem, wodurch die einfache Wartbarkeit und Erweiterbarkeit des Systems gewährleistet ist. Hierdurch wird eine wichtige Voraussetzung an eine moderne Systemarchitektur sichergestellt: die permanente Anpaßbarkeit des Systems an neue Aufgaben und Anforderungen, wie z. B. an modifizierte Geschäftsabläufe und veränderte gesetzliche Rahmenbedingungen.

Die ursprünglich von vielen vertretene These und Hoffnung, eine umfangreiche Wiederverwendung von Bausteinen würde sich zwangsläufig von selbst ergeben und sei somit zum Nulltarif zu haben, sofern nur genügend viele Bausteine zur Verfügung stehen, hat sich in der Praxis nicht bewährt. Empirische Studien über die Wiederverwendung von Software haben gezeigt, daß sich die größten Erfolge erst dann einstellen, wenn die Entwicklung von Bausteinen geplant und durch den gesamten Entwicklungsprozeß unterstützt wird (siehe z. B. [Fra95]). Mit anderen Worten: Die Entwicklung von Bausteinen darf nicht nur als positiver Seiteneffekt einer traditionellen Vorgehensweise betrachtet werden. Ziel muß es sein, die zufällige Wiederverwendung von Bausteinen – die zweifelsohne heute schon in verschiedenen Bereichen praktiziert wird – hinsichtlich einer geplanten und ingenieurmäßigen Vorgehensweise zu optimieren.

Welche grundsätzlichen Aspekte und Aktivitäten gilt es zu beachten, die eine effektive bausteinorientierte Anwendungsentwicklung erst ermöglichen? Einige Hinweise hierzu finden sich in folgendem Zitat aus [Nie95]: „Present software en-

Das im Jahr 1991 gegründete Informatikzentrum der Sparkassenorganisation GmbH hat den Auftrag, die informationstechnischen Aktivitäten der Sparkassenorganisation zu bündeln, Strategien für die zukünftige technologische Ausrichtung zu entwickeln und die Kooperation der Anwendungsentwicklung zwischen den Großrechenzentren der Landesbanken und Verbandsrechenzentren zu organisieren. Das SIZ führt selbst keine eigenständigen Entwicklungsarbeiten durch, sondern fungiert vor allem als Impulsgeber, Koordinationsstelle sowie als Informations- und Technologielieferant. Die in diesem Artikel vorgestellten Ergebnisse wurden im Rahmen des SIZ-Projekts „Software-Grundbausteine Phase 1“ erarbeitet, an dem Mitarbeiter des SIZ, externe Berater sowie Mitarbeiter der Sparkassenorganisation beteiligt waren.

gineering practice actually discourages component-oriented development by focusing on the individual application rather than viewing it as part of a much broader software process.“

Die Entwicklung wiederverwendbarer Bausteine, die projektübergreifend oder gar unternehmensweit genutzt werden können, ist erst dann möglich, wenn der Fokus nicht ausschließlich auf den projektspezifischen Interessen und Zielen liegt. Während des gesamten Entwicklungszyklus der Bausteine sollte eine von einer konkreten Anwendung abstrahierende Sichtweise eingenommen werden, die es ermöglicht, Bausteine mit „hinreichend interessanter“ Funktionalität zu erstellen. Dies bedeutet, daß ein Baustein einen bestimmten fachlichen oder technischen Kontext vollständig abdeckt und somit eine logisch zusammenhängende Gruppe von Funktionen zur Verfügung stellt, die in unterschiedlichen Anwendungen und Kontexten genutzt werden können.

Die abstrahierende Sichtweise kann im allgemeinen jedoch nur ein Entwickler einnehmen, der über genügend *Domänenwissen* verfügt. Hierzu gehören alle Informationen und Erfahrungswerte, die direkt oder indirekt für die Entwicklung von Anwendungssystemen verwendet werden, wie z. B. Systemarchitekturen, De-

signmuster, verfügbare Klassenbibliotheken etc. Darüber hinaus müssen bei der Entwicklung von Bausteinen laufende und geplante Entwicklungsprojekte berücksichtigt werden, um möglichst viele Gemeinsamkeiten zwischen unterschiedlichen Problemstellungen zu identifizieren. Diese Überlegungen motivieren eine Trennung zwischen Bausteinentwicklern einerseits und Bausteinutzern andererseits.

Die Aufgabe eines *Bausteinentwicklers* (engl. *Component Builder*) besteht darin, bestimmte Funktionalitäten in Form von wiederverwendbaren Bausteinen zu realisieren. Diese umfassen fachliche oder technische Funktionalitäten und sind im allgemeinen keine vollständigen Anwendungen, sondern stellen wiederverwendbare Komponenten dar, die in verschiedenen Kontexten eingesetzt werden können.

Ein *Bausteinutzer* (engl. *Component User* bzw. *Component Assembler*) ist typischerweise ein Anwendungs- oder Bausteinentwickler, der bestimmte Teile eines zu erstellenden Systems mit Hilfe bereits existierender Komponenten realisiert. Er muß über fundiertes Fachwissen verfügen, ist aber entkoppelt von systemnaher Programmierung mit Datenbankzugriffen, Transaktionsverwaltung, Sicherheit, Netzwerkprotokollen usw.

Ausgehend von dieser Trennung läßt sich ein grundsätzliches Modell einer bausteinorientierten Anwendungsentwicklung herausarbeiten. Das in Abbildung 1 dargestellte Modell ist auf einer relativ hohen Abstraktionsebene beschrieben und hat sowohl für eine prozedurale als auch für eine objektorientierte Vorgehensweise Gültigkeit.

In horizontaler Richtung sind die folgenden beiden Prozesse zu sehen:

- die Entwicklung von Anwendungssystemen unter Nutzung vorhandener oder noch zu entwickelnder Bausteine sowie
- die Entwicklung von Bausteinen, die in verschiedenen Kontexten eingesetzt werden können.

Als Bindeglied zwischen diesen Prozessen fungieren die Blöcke *Baustein-Komitee* und *Baustein-Repository*. Im Rahmen einer Anwendungsentwicklung werden bestimmte fachliche bzw. technische Funktionalitäten benötigt. Werden diese Funktionalitäten bereits durch vorhandene, in einem Repository abgelegte Bausteine realisiert, so erfolgt eine Wiederverwendung. Anderenfalls werden aus der Anwendungsentwicklung heraus Anforderungen

an zu realisierende Bausteine abgesetzt. Solche Anforderungen werden von verschiedenen Anwendungsentwicklungsprojekten gestellt. Nach einem Abgleich und Verdichten dieser Anforderungen durch ein Baustein-Komitee werden von einem oder mehreren Bausteinentwicklern die geforderten Funktionalitäten umgesetzt. Die resultierenden Bausteine werden in einem Repository abgelegt und stehen somit zur allgemeinen Nutzung zur Verfügung.

Hinsichtlich einer mittelfristigen Umsetzung kann die im folgenden Abschnitt beschriebene dreistufige Vorgehensweise gewählt werden.

Hauptaugenmerk der ersten Migrationsphase besteht darin, den Anwendungsentwickler für die Bausteinthematik zu sensibilisieren und ihn langsam an die damit verbundene Arbeitsweise heranzuführen.

In der zweiten Phase findet eine Spezialisierung der Programmierer – je nach Qualifikation und Interessenlage – als Bausteinentwickler oder Bausteinnutzer statt. Ein Bausteinnutzer sollte (entsprechend seiner Funktion als Anwendungsentwickler) die zu lösende fachliche Problemstellung erfassen können und erkennen, inwieweit Teile des Zielsystems durch vorhandene Bausteine abzudecken sind. Sind die Bausteine so konzipiert, daß sie einfach kombiniert werden können, braucht ein Bausteinnutzer sich nicht mit technischen und

kommunikationsspezifischen Problemen zu befassen. Hingegen sollte ein Bausteinentwickler exzellente Programmierfähigkeiten besitzen und in der Lage sein, qualitativ hochwertige Softwareartefakte zu produzieren. In dieser zweiten Phase kann ein Bausteinentwickler kommissarisch die Aufgaben des Baustein-Komitees und des Repository-Komitees übernehmen.

Je größer die Anzahl der Bereiche und Projekte einer Organisation ist, die an der bausteinorientierten Vorgehensweise partizipieren wollen, desto notwendiger sind die Einführung eines Baustein- und eines Repository-Komitees als organisatorisch eigenständige Bereiche. Diese Aktivität ist Bestandteil der dritten Migrationsphase. Die Hauptaufgabe der beiden Komitees besteht darin, zwischen den verschiedenen Projekten zu vermitteln. Eine weitere Aufgabe ist die Definition von Richtlinien für die Entwicklung von Bausteinen und für deren Aufnahme in das Repository; ebenso muß die Umsetzung und Einhaltung der Richtlinien garantiert werden.

## Dokumentations-, Repräsentations- und Suchmethoden

### Dokumentation von Bausteinen

Für die Wiederverwendung von Baustei-

nen ist deren adäquate Dokumentation von besonderer Bedeutung. Ein Baustein muß derart dokumentiert sein, daß ein potentieller Bausteinnutzer schnell das Einsatzgebiet, den Funktionsumfang und die Abhängigkeiten zu anderen Bausteinen erkennen kann. Das sogenannte *3C-Modell* (vgl. [Wei91]) spielt inzwischen eine große Rolle im Bereich der Softwarewiederverwendung und unterstützt die Dokumentation der Dimensionen *Concept*, *Content* und *Context*.

■ **Concept:** Das *Concept* dokumentiert die Funktionalität eines Bausteins und legt somit fest, welche Dienste erbracht werden. Dies umfaßt insbesondere eine Beschreibung der Schnittstelle, der Vor- und Nachbedingungen sowie von Anwendungsfeldern, in denen der Baustein eingesetzt werden kann.

■ **Content:** Diese Dimension zielt auf die Implementierung eines Bausteins ab und dokumentiert, wie das *Concept* des Bausteins realisiert ist. Neben Aussagen zur Testhistorie, den Testverfahren und Testergebnissen beinhaltet diese Dimension auch die Änderungshistorie.

■ **Context:** Der *Context* beschreibt zum einen die Umgebung, in der der Baustein implementiert ist (Entwicklungsumgebung), und zum anderen die Umgebung, für die der Baustein ausgelegt ist (Produktionsumgebung). Darüber

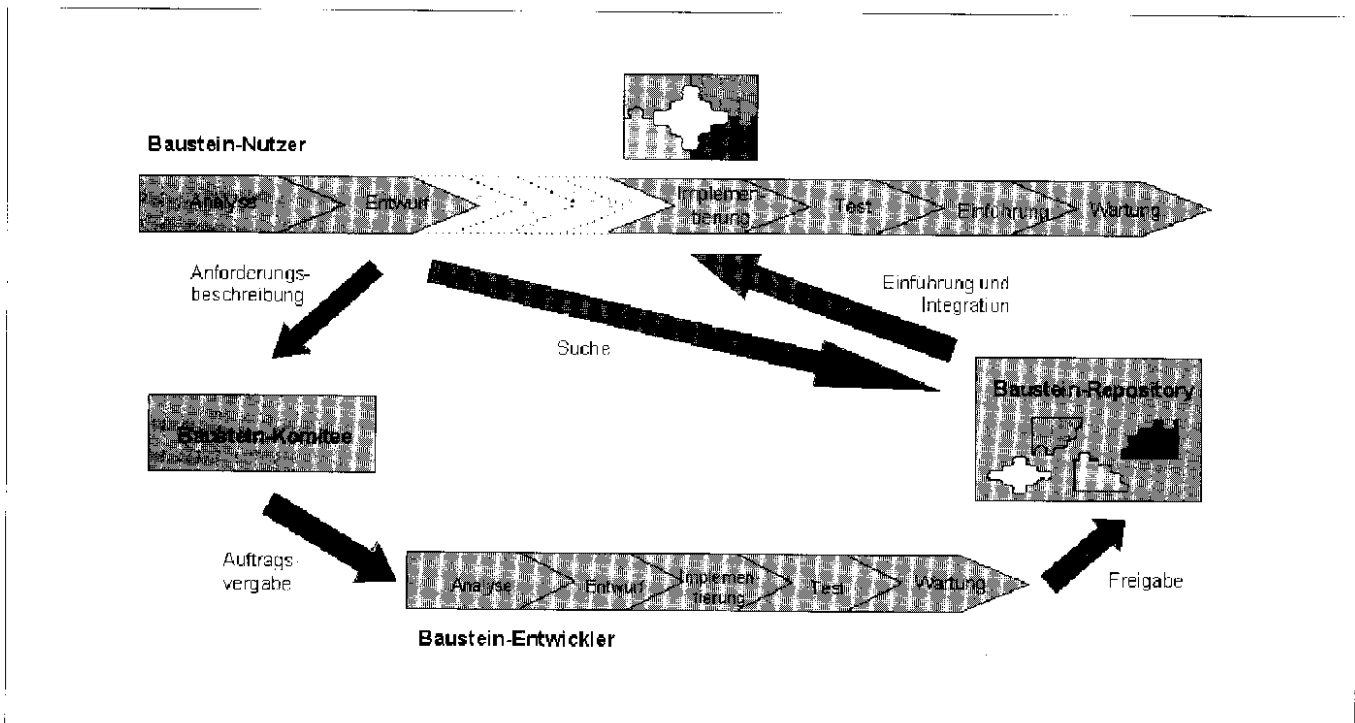


Abb. 1: Bausteinorientiertes Anwendungsentwicklungsmodell

hinaus umfaßt der *Context* eine Beschreibung der Abhängigkeiten zu anderen Bausteinen, zu Compilern, zu Speicherressourcen usw.

## Repräsentationsmethoden

Repräsentationsmethoden werden benötigt, um Bausteine geeignet in Repositories ablegen zu können. Im folgenden werden die grundlegenden Konzepte der drei bekanntesten Ansätze erläutert. Weitere Ansätze, wie etwa wissens- oder hypertextbasierte Methoden, sind in [Zen95] beschrieben.

Bei der *attributbasierten Repräsentation* werden Bausteine unter Verwendung von Attributen und entsprechenden Attributwerten beschrieben. Als Attribute kommen grundsätzlich die in den Dimensionen des 3C-Modells beschriebenen Aspekte (Beschreibung der Schnittstelle, Änderungshistorie etc.) in Frage. Weitere Attribute können eingeführt werden, um z. B. über den Typ eines Bausteins (Quellcode, Übersetzer Code etc.) oder den Status des Bausteins (geplant, abgenommen etc.) Auskunft zu geben. Bei einer Suche können solche Bausteine aufgefunden werden, die bestimmte Attributwerte bzw. Kombinationen von Attributwerten aufweisen.

Grundlage der *thesaurusbasierten Repräsentation* ist eine wohldefinierte Menge von Begriffen – der sogenannte Thesaurus. Indexterme sind Begriffe bzw. Kombinationen von Begriffen, die zur Indexierung von Bausteinen herangezogen werden können. Einem Baustein wird eine Menge von Indextermen zugeordnet, die ihn am genauesten beschreibt. Die Suche und das Auffinden von Bausteinen erfolgen über die dem Baustein assoziierten Indexterme.

Die aus der Bibliothekswissenschaft stammende *Facettenklassifikation* ist durch Prieto-Díaz auf den Bereich der Softwarewiederverwendung übertragen worden (vgl. [Pri91]). Die Facettenklassifikation setzt voraus, daß für eine bestimmte Domäne die relevanten inhaltlichen Aussagen herausgearbeitet wurden, die die sogenannten *Items* definieren. Die nachfolgende Gruppierung der *Items* zu *Facetten*, die von Prieto-Díaz als „*Perspectives, Viewpoints, or Dimensions of a Particular Domain*“ verstanden werden, ermöglicht eine weitere Strukturierung der Domäne. So können beispielsweise bei einer einfachen Facettenklassifikation die *Items* „Sort“, „Compress“ und „Append“ zu einer Facette „Function“ gruppiert werden, und die *Items* „Ar-

rays“, „Strings“ und „Trees“ zu einer Facette „Types“. Die Repräsentation eines konkreten Bausteins bezüglich einer bestimmten Facettenklassifikation erfolgt durch die Auswahl von *Items* für die eingeführten Facetten.

Wiederverwendbare Softwarebausteine sollten mit verschiedenen Methoden repräsentiert werden, um einem Bausteinutzer unterschiedliche Sichten auf ein Baustein-Repository zu geben. Während die attribut- wie auch die thesaurusbasierte Repräsentation eine allgemeine Sicht auf die Menge der verfügbaren Bausteine ermöglichen, erlaubt es die Facettenklassifikationen, verschiedene anwendungsbereichsspezifische Sichten abzubilden. Unterschiedliche Repräsentationsmethoden ermöglichen darüber hinaus das Wiederauffinden von Bausteinen mit mehreren Suchverfahren. So kann das unterschiedliche Benutzerverhalten bei der Suche unterstützt und das Wiederauffinden verbessert werden (vgl. auch [Fra94]).

## Suchverfahren

Damit sich ein potentieller Bausteinutzer schnell informieren kann, ob eine bestimmte Funktionalität bereits als verfügbarer Baustein in einem Repository zur Verfügung steht, werden bestimmte Suchverfahren benötigt. Prinzipiell können zwei Klassen von Suchverfahren identifiziert werden: deskriptive und navigierende.

Bei der *deskriptiven Suche* unterscheidet man

- die konventionelle Suche, bei der Softwarebausteine in einem Schritt wiederaufgefunden werden,
- die Umgebungssuche, welche die Suche nach ähnlichen Softwarebausteinen berücksichtigt, und
- die dynamische Suche, bei der bestimmte Interaktionen mit dem Benutzer vorgesehen sind.

Bei der konventionellen Suche kann mit Booleschen Operatoren gearbeitet werden, um komplexere Anfragen zu formulieren. Neben der Suche nach bestimmten Kombinationen von Attributwerten, Indextermen bzw. *Items* kann in vielen Fällen auch eine Volltext-Suche hilfreich sein. Die Umgebungssuche kann als Generalisierung der konventionellen Suche betrachtet werden und ermöglicht es, Ähnlichkeiten zwischen Bausteinen (z. B. auf der Basis von Beziehungen und Ähnlichkeiten zwischen Indextermen bzw. *Items*) zu berücksichtigen. Bei der dynamischen Suche schließlich wird

eine Verbesserung des Suchergebnisses auf Basis einer Benutzerinteraktion erreicht: Das Wiederauffinden von Softwarebausteinen erfolgt iterativ, wobei mit einer näherungsweise Suchanfrage begonnen wird; über eine schrittweise Spezialisierung der Anfrage wird das Suchergebnis verbessert.

Bei der *navigierenden Suche* wird vorausgesetzt, daß Bausteine über semantische Beziehungen (z. B. „A ruft B auf“) miteinander verknüpft sind. Ein Benutzer kann an diesen Beziehungen „entlanglaufen“ und weitere Bausteine „entdecken“. Beim ungerichteten Browsing erfolgt die Suche nach Bausteinen durch zielloses Suchen unter Ausnutzung der semantischen Beziehungen. Die Suche nach wiederverwendbaren Softwarebausteinen wird praktisch ausschließlich durch Auswertung lokaler Pfade aus der momentanen Suchposition heraus gestartet. Im Gegensatz dazu erfolgt beim zielgerichteten Browsing die Suche nach Bausteinen systematisch. Gesuchte Bausteine werden durch globale Übersichtskarten, die jeweils erreichte Suchposition und relative Bewegungen von der Suchposition angesteuert.

## Anforderungen an Softwarebausteine

Werden bei Entwurf, Implementierung und Dokumentation von Bausteinen bestimmte Richtlinien berücksichtigt, so ergeben sich Vorteile bei ihrer Wiederverwendung, wie z. B. die einfache Integration einer Komponente in das Zielsystem. Da jeder Baustein letztendlich ein Codefragment einer Programmiersprache darstellt, sollten grundsätzlich bei der Entwicklung von Bausteinen bewährte Prinzipien des Software-Engineerings (z. B. Trennung von Schnittstelle und Implementierung) angewendet werden. Bausteine sollten einen „Black-Box“-Charakter haben, d. h. die Interna eines Bausteins sind nach außen nicht sichtbar. Um darüber hinaus ihre effektive Wiederverwendung zu unterstützen, sollten Bausteine

- hinreichend gut dokumentiert sein,
- eine hinreichend interessante Funktionalität anbieten,
- einfach benutzt werden können und
- qualitativ hochwertige Artefakte sein.

## Verständlichkeit und Dokumentation

Damit ein Anwendungsentwickler schnell beurteilen kann, ob zur Lösung eines Problems ein bestimmter Baustein (oder eine Menge von Bausteinen) herangezogen werden kann, muß die Funktionalität eines Bausteins leicht zu identifizieren sein. Bausteine müssen daher geeignet dokumentiert sein, und Fragen bezüglich des funktionalen Umfangs, der Abhängigkeit zu anderen Bausteinen etc. müssen schnell beantwortet werden können. Zu diesem Zweck kann beispielsweise das oben beschriebene 3C-Modell verwendet werden.

Geht man davon aus, daß die zur Verfügung stehenden Bausteine in unterschiedlichen Programmiersprachen entwickelt wurden, so kommt der Beschreibung der Schnittstelle besondere Bedeutung zu. Diese umfaßt all die Informationen, die ein Bausteinnutzer benötigt, um einen syntaktisch korrekten Aufruf einer Schnittstellenfunktion zu formulieren. Die Schnittstellenbeschreibung muß somit die Namen der Schnittstellenfunktionen, die Typen der Ein- bzw. Ausgabewerte sowie auftretende Ausnahmesituationen (*Exceptions*) festlegen. Traditionell hängt die Beschreibung der Schnittstelle von der verwendeten Programmiersprache ab.

Möchte ein Anwendungsentwickler Bausteine, die in verschiedenen Programmiersprachen geschrieben sind, einsetzen, muß er unterschiedliche Arten von Schnittstellenbeschreibungen erlernen und verstehen. Um die damit verbundenen Probleme zu vermeiden (wie z. B. unterschiedliche Repräsentationen von Datentypen), ist die Verwendung einer *programmiersprachenunabhängigen* Beschreibungssprache sinnvoll. Idealerweise werden die Schnittstellen von Bausteinen einheitlich beschrieben – unabhängig von der konkreten Implementierungssprache. Hiermit wird es ermöglicht, sich von den Charakteristika einer bestimmten Programmiersprache bzw. Technologie zu lösen und eine abstraktere Beschreibungsform zu nutzen. Die Anbindung der Schnittstellenbeschreibungssprache an konkrete Programmiersprachen legt fest, auf welche Art und Weise die Konstrukte dieser Beschreibungssprache abgebildet werden. Die Implementierungssprache eines Bausteins kann somit geändert werden, ohne daß die Schnittstellenbeschreibung modifiziert werden muß.

## Geeignete Abstraktion und Einfachheit der Schnittstelle

Eine Aktivität, deren große Bedeutung häufig unterschätzt wird, ist die Identifikation einer geeigneten Abstraktion. Krueger stellt in einem umfangreichen Artikel über Softwarewiederverwendung ([Krue92]) fest: „Abstraction plays a central role in software reuse. Concise and expressive abstractions are essential if software artifacts are to be effectively reused.“

Die Schnittstelle eines Bausteins sollte einfach sein und die Abstraktion eines interessanten Problembereiches repräsentieren. D. h. ein Baustein deckt einen bestimmten fachlichen oder technischen Kontext *vollständig* ab und beschreibt somit eine logisch zusammenhängende Gruppe von Funktionen. Die Umsetzung dieser Anforderung ist in der Regel keine triviale Aufgabe und erfordert umfangreiches Domänenwissen sowie Erfahrungen, die erst gesammelt werden müssen. Da Bausteine selbst Änderungen unterliegen, sollten sie so implementiert werden, daß sie neuen Anforderungen angepaßt werden können. Ein guter Baustein „reift“ im Laufe der Zeit; er trägt damit zur konzeptuellen Vereinfachung einer Problemstellung bei und wird ein immer wertvolleres Artefakt des Unternehmens.

Man beachte, daß die Einfachheit eines Bausteins nicht gleichzusetzen ist mit restringierter Funktionalität. Realisieren zwei oder mehr Schnittstellenfunktionen strukturell ähnliche Probleme, so kann durch eine Generalisierung – und damit Abstraktion von spezifischeren Lösungen – eine Schnittstellenfunktion identifiziert werden, die die anderen Funktionen subsumiert.

## Unabhängigkeit

Die Unabhängigkeit eines Bausteins umfaßt verschiedene Facetten:

- Änderungen innerhalb eines Bausteins (d. h. Modifikationen der Implementierung, aber nicht der Schnittstelle) sollten keine Auswirkungen auf die Art der Bausteinnutzung haben.
- Bausteine sollten minimale Abhängigkeiten nach außen, wie z. B. zu globalen Variablen, besitzen.
- Bausteine sollten eine hohe Portabilität aufweisen und können somit von verschiedenen Compilern auf unterschiedlichen Plattformen übersetzt werden.

Durch die Trennung von Schnittstelle und Implementierung bleibt dem Bausteinnutzer die konkrete Realisierung verborgen – somit sind keine (direkten) Zu-

griffe auf intern verwendete Datenstrukturen, Variablen usw. möglich. Der Implementierungsteil eines Bausteins kann demzufolge modifiziert oder ausgetauscht werden, ohne daß das Anwendungsprogramm angepaßt werden muß – sofern sich seine Schnittstelle nicht ändert. Eine noch weitergehende Unabhängigkeit kann erreicht werden, wenn nicht nur die Implementierung eines Bausteins, sondern auch seine technische Umgebung (wie z. B. Programmiersprache, Betriebssystem oder Plattform) gewechselt werden kann, ohne daß ein Anwendungsprogramm von diesen Änderungen beeinflusst wird.

## Qualität

Die Qualität eines Bausteins umfaßt eine Reihe von Aspekten, wie z. B. Korrektheit, Effizienz, Robustheit, Stabilität, Konfigurierbarkeit, Wartbarkeit und Dokumentation. Die Notwendigkeit einer umfangreichen und geeigneten Dokumentation wurde bereits erörtert.

Eine unabdingbare Voraussetzung für die Wiederverwendung eines Bausteins besteht darin, daß die ihm aufgetragene Funktionalität korrekt und vollständig implementiert ist. Bestehen Zweifel an der Korrektheit eines Bausteins, so sinkt die Motivation eines potentiellen Nutzers, diesen wiederzuverwenden. Auch kann der Einsatz eines fehlerhaften Bausteins kontraproduktiv sein: Der Aufwand, der mit dem Finden und der Realisierung eines *Workarounds* verbunden ist, mag den ultimativen Nutzen des Bausteins nicht aufwiegen. Die Korrektheit sollte daher durch umfangreiche Testfälle verifiziert worden sein; an Hand der dokumentierten Testfälle kann sich ein Bausteinnutzer von den Testergebnissen überzeugen.

Robustheit und Stabilität bedeuten, daß bei Nutzung eines Bausteins keine „unvorherschaubaren“ Probleme, wie z. B. Laufzeitfehler, auftreten können. Als wichtige Hilfsmittel zur Realisierung dieser Anforderung sind zu nennen: die Behandlung von Ausnahmesituationen (*Exception Handling*) sowie das statische bzw. dynamische Überprüfen von Typen und Argumentwerten.

Greift der Baustein selbst auf weitere Ressourcen zu, sollte ein geeigneter Mechanismus zur Behandlung von Ausnahmen angewendet werden. Konnte beispielsweise der Aufruf einer bestimmten externen Funktion nicht erfolgreich durchgeführt werden, so kann der aufrufende Baustein auf definierte Art und Weise dar-

auf reagieren und entsprechende Aktionen starten. Je nach verwendeter Programmiersprache bzw. Kommunikationstechnologie stehen unterschiedliche Mechanismen zur Verfügung, wie z. B. die Verwendung von „Return Codes“ (in C) oder von „Exception Objects“ (in C++ und Java).

## Bausteinorientierte Anwendungsentwicklung mittels ORB-basierter Ansätze

Bislang wurden grundlegende Aspekte und Voraussetzungen einer bausteinorientierten Anwendungsentwicklung betrachtet. Grundsätzlich kann festgehalten werden, daß eine bausteinorientierte Vorgehensweise nicht an bestimmte Technologien, Systeme und Werkzeuge gebunden ist. Durch die Verwendung moderner Ansätze im Bereich der *Object Request Broker (ORB)* kann allerdings eine effektive Umsetzung erreicht werden.

Aufgrund der von einem ORB vorgegebenen Funktionalität kann eine effektive Integration und Interoperabilität von Bausteinen erreicht werden. Dies bedeutet, daß Bausteine auf einfache und homogene Art und Weise von einem Bausteinnutzer zu komplexen Anwendungen zusammengesetzt werden können – unabhängig davon, in welcher technischen Umgebung (Programmiersprache, Betriebssystem, Rechnerarchitektur etc.) sie zur Verfügung stehen. Hierdurch wird garantiert, daß sich ein Anwendungsentwickler nicht mit technischen oder kommunikationsspezifischen Fragestellungen auseinandersetzen muß, sondern sich ausschließlich auf das fachliche Anwendungsproblem konzentrieren kann.

In den vergangenen Jahren wurden verschiedene ORB-Technologien entwickelt, deren Aufbau und Funktionsweise jedoch zum Teil sehr unterschiedlich sind. Heutzutage spielen – aufgrund ihrer hohen Standardisierung und Verbreitung – die ORB-Technologien CORBA von der „Object Management Group“ (OMG) und COM/OLE bzw. DCOM von Microsoft eine dominante Rolle. \*\* Beide Technologien eignen sich auf-

grund der im folgenden näher ausgeführten Aspekte für eine baustein-basierte Anwendungsentwicklung.

### Trennung von Schnittstelle und Implementierung

Sowohl CORBA als auch COM bzw. DCOM erlauben die Spezifikation von Schnittstellen schon während der Entwurfsphase eines Systems in einer implementierungsunabhängigen Schnittstellenbeschreibungssprache (engl. „Interface Definition Language“, Abk. IDL). Die Verwendung einer IDL erfordert die exakte Trennung zwischen der Implementierung eines Dienstes einerseits und seiner Nutzung andererseits. Die oben eingeführten Rollen „Bausteinentwickler“ und „Bausteinnutzer“ können somit auf naheliegende Weise umgesetzt werden. Ein Bausteinnutzer benötigt keinerlei Informationen über die Implementierung eines Bausteins. Die Kapselung der Implementierung verschiedener Bausteine hinter IDL-Schnittstellen verbietet darüber hinaus den unkontrollierten Zugriff eines Bausteins auf (interne) Daten eines anderen Bausteins; auch lassen sich globale Daten nur noch mittels kontrollierbarer Zugriffe über IDL-Schnittstellen einsetzen.

### Ortstransparente Kommunikation

Die Architektur von CORBA ist originär auf den Entwurf und die Umsetzung von verteilten Systemen ausgerichtet. Für die Implementierung von Bausteinen mit Hilfe von CORBA ergibt sich dadurch sofort die Möglichkeit zu ihrer Verteilung auf unterschiedliche Prozesse und Rechner. Inzwischen stehen für alle relevanten Plattformen CORBA-Implementierungen zur Verfügung, so daß in der Tat heterogene Systemlandschaften integriert werden können. DCOM ist für Windows NT 4.0 und Windows 95 verfügbar. Derzeit erfolgt durch die Software AG eine Portierung auf weitere Plattformen, wie z. B. Solaris und MVS.

Der Zusammenbau von Bausteinen erfolgt mittels CORBA und COM/DCOM dynamisch zur Laufzeit des Systems. Unabhängig davon, in welcher Umgebung ein Baustein zur Verfügung steht, bleibt das Aufrufmuster für einen Dienstenutzer gleich. Dadurch lassen sich einzelne Bausteine zu jedem Zeitpunkt verbessern, erweitern oder verschieben. Die Suche nach verfügbaren Bausteinen bzw. Diensten kann über bestimmte Lokalisierungsdienste erfolgen.

### Portabilität

Durch die strikte Trennung zwischen Schnittstelle und Implementierung ist die Portierung eines Bausteins von einer auf eine andere Plattform unter Beibehaltung derselben Schnittstelle möglich – vorausgesetzt, auf der Zielplattform existiert eine Implementierung von CORBA bzw. DCOM. Hält man sich bei der Entwicklung eines CORBA-Bausteins an die von der OMG definierten Mindesteigenschaften von ORB-Implementierungen, oder ist die verwendete ORB-Implementierung auch auf der Zielplattform erhältlich, so läßt sich die Kommunikationsschicht des Bausteins dort ohne Änderungen neu übersetzen. Nutzt man dagegen über die Spezifikationen der OMG hinausgehende Eigenschaften eines ORB, der auf der Zielplattform nicht verfügbar ist, so muß eine geeignete Abstraktionsschicht verwendet werden. Laut Aussage von Microsoft bzw. der Software AG wird die API von DCOM für die unterstützten Plattformen identisch sein.

### Ausnahmebehandlung

Für die Realisierung von Anwendungssystemen – insbesondere von verteilten Systemen – ist ein einheitliches Konzept zur Behandlung von Ausnahme- bzw. Fehlersituationen von großer Bedeutung. Zum einen muß ein Baustein, der bestimmte Dienste realisiert, so auf auftretende Fehler eingehen, daß ein Bausteinnutzer eine wohldefinierte Rückgabe erhält. Zum anderen sollte der Bausteinnutzer abfragen können, was für eine Ausnahmesituation aufgetreten ist, ohne zu wissen, in welcher Umgebung der angesprochene Baustein abläuft.

CORBA spezifiziert ein elaboriertes Konzept zur Ausnahmebehandlung in einer verteilten Umgebung. Durch die Deklaration von *Exceptions* innerhalb der IDL lassen sich Ausnahmen in C++ und Java durch das *Structured Exception Handling (Try-Throw-Catch-Mechanismus)* und in anderen Implementierungssprachen über – von den IDL-Stubs und Skeletons unterstützte – Mechanismen netzwerkweit generieren, abfangen und bearbeiten.

In COM werden Statusmeldungen über die erfolgreiche Abarbeitung eines Funktionsaufrufs durch einen ausgezeichneten Rückgabewert an den Aufrufer übergeben. Damit ist der Rückgabewert für alle COM-Methoden vorbelegt

\*\* Man beachte, daß beide Ansätze eine unterschiedliche Definition von „Objekt“ voraussetzen. Dies ist jedoch irrelevant für die folgenden Ausführungen.

und kann nicht für sonstige Rückgabewerte verwendet werden. Microsoft gibt allerdings an, in zukünftigen Versionen ebenfalls das *Structured Exception Handling* unterstützen zu wollen.

### Bereitstellung höherwertiger Dienste

Zusätzlich zu den reinen ORB-Funktionalitäten bieten sowohl CORBA als auch COM weitere Dienste an, die die Implementierung und Nutzung von Bausteinen in einer verteilten Umgebung erleichtern. Im Bereich CORBA werden durch die *Object Services* und *Common Facilities* die Schnittstellen von anwendungs-unabhängigen Diensten für eine Vielzahl von Bereichen, wie z. B. Transaktion, Sicherheit und Persistenz, spezifiziert. Seitens COM werden durch die Komponenten *Monikers*, *Structured Storage* und *Uniform Data Transfer* verschiedene Basisdienste abgedeckt.

### Was bleibt zu tun?

Wird durch den Einsatz moderner ORB-Technologien eine hohe Verteilungs- und Kommunikationstransparenz erreicht, so bedeutet dies noch lange nicht, daß damit das Problem der Verteilung von Bausteinen bzw. Objekten gelöst sei. Ganz im Gegenteil: Es bleibt eminent wichtig, beim Entwurf verteilter Systeme auf Basis eines ORB Verteilungsaspekte explizit zu berücksichtigen, um performante Systeme zu erhalten. So stellt sich die Frage, wie verschiedene Bausteine zu Bausteingruppen zusammengefaßt und auf Prozesse abgebildet werden können. Gleichfalls muß in Abhängigkeit von der Funktionalität eines Bausteins entschieden werden, ob eine synchrone bzw. asynchrone Kommunikation gewählt werden soll. Da – im Sinne des Investitionsschutzes – häufig Alt- bzw. Fremdanwendungen in eine auf CORBA bzw. DCOM basierende Anwendungsarchitektur integriert werden müssen, stellt sich die Frage, wie sogenannte „Legacy-Software“ als CORBA- bzw. DCOM-Objekte zu kapseln ist und wie deren Rolle im Gesamtsystem aussieht.

Diese und weitere wichtige Fragestellungen, die es beim Entwurf von verteilten Systemen zu berücksichtigen gilt, sollen hier nicht weiter diskutiert werden. Der interessierte Leser kann auf die entsprechende Fachliteratur zurückgreifen.

## Risiken

Bei der Umsetzung der bausteinorientierten Vorgehensweise sind bestimmte Risikofaktoren zu beachten. Die Erstellung von wiederverwendbaren Softwarebausteinen, die bestimmten Anforderungen genügen, ist mit einem höheren Verbrauch von Ressourcen (Kosten, Zeit etc.) verbunden. Werden bei der Planung eines Entwicklungsprojekts zu enge Abgabetermine gesetzt, besteht die Gefahr, daß die Softwarebausteine ausschließlich für die speziellen Bedürfnisse des Projekts entwickelt werden. Ist keine Zeit vorgesehen, geeignete Generalisierungen der Schnittstellen durchzuführen, läßt sich das Wiederverwendungspotential nicht ausschöpfen.

Der volle Nutzen einer bausteinorientierten Vorgehensweise ist kurzfristig nicht sichtbar. Zunächst sind sogar bestimmte Investitionen und organisatorische Maßnahmen erforderlich, die erhöhte Kosten verursachen. Die umfangreiche Wiederverwendung von Bausteinen kann erst mittelfristig erfolgen, wenn im Laufe der Zeit immer mehr „interessante“ Bausteine zur Verfügung stehen und ein Anwendungsentwickler die Einsatzmöglichkeiten und Funktionalität der Bausteine mehr oder weniger vollständig exploriert hat.

Die Entwickler müssen sich mit der bausteinorientierten Anwendungsentwicklung und der ORB-Technologie vertraut machen. Dies kann zum einen in Form von Schulungen und Seminaren erfolgen. Zum anderen können die Mitarbeiter über die Durchführung von nicht-kritischen Projekten sowohl an Wiederverwendungstechniken als auch an den Umgang mit ORB-basierten Systemen herangeführt werden.

Zusammenfassend läßt sich sagen, daß eine kurzfristig auf Kostenreduzierung ausgelegte IT-Strategie den vollen Umstieg auf die Anwendungsbereitstellung auf der Basis von reifen, stabilen und wartbaren Bausteinen erschwert, wenn nicht gar unmöglich macht. Der mittel- bis langfristige Nutzen gleicht diese erhöhten Startinvestitionen jedoch mehr als aus.

## Danksagung

Dieser Artikel faßt grundlegende Ergebnisse des Projekts „Software-Grundbausteine Phase 1“ zusammen, das in der Zeit von Februar bis Juli 1996 bei der dvg Hannover durchgeführt wurde. Die Autoren bedanken sich bei Holger Feske (Rösch Consulting), Stefan Gastinger (FAST), Lewis Gardner (Rösch Consulting), Peter Holzwarth (Interactive Objects) und Dr. Andreas Zandler (FAST), die maßgeblich an der Erarbeitung der hier zusammengefaßten Projektergebnisse mitgewirkt haben. Des Weiteren möchte sich das Projektteam an dieser Stelle bei Matthias Bendzulla (SIZ), Dr. Marjan Jurcic (SIZ) und Dariusch Soltani (dvg Hannover) sowie dem Review-Team für die anregenden Diskussionen bedanken.

## Literatur

- [Fra95] W.B. Frakes, C.F. Fox, Sixteen Questions about Software Reuse, Comm. of the ACM, Juni 95, Vol 38, No 6, 1995
- [Fra94] W.B. Frakes, T.P. Pole, An empirical study of representation methods for reusable software components, IEEE Transactions on Software Engineering 20, 8, 617-630, 1994
- [Krue92] C. Krueger, Software Reuse, ACM Computing Surveys, Vol. 24, No. 2, 1992
- [Nie95] O. Nierstrasz, D. Tsichritzis, Object-oriented software composition, Prentice Hall, London, 1995
- [Pri91] R. Prieto-Díaz, Implementing faceted classification for Software Reuse, Comm. of the ACM 34, 5, 88-97, 1991
- [Wei91] B.W. Weide, W.F. Ogden, S.H. Zweben, Reusable software components, in: Advances in Computers Vol. 33, M.C. Yovits (Ed.), Academic Press, Boston, 1-65, 1995
- [Zen95] A. Zandler, Konzepte, Erfahrungen und Werkzeuge zur Software-Wiederverwendung, Tectum, Marburg, 1995