

Modellierung von Architekturen

Malte Foegen, Patrick Atamaniuk

wibas GmbH, Otto-Hesse-Str. 19 / T5, 64293 Darmstadt
{malte.foegen, patrick.atamaniuk}@wibas.de

Abstract: This paper is based on our experience with medium and large software projects. It defines the term “architecture” and shows its relevance in software engineering. In its main part this paper describes the elements of an architecture and defines the work products produced during a software development project in order to model an architecture.

1 Grundlagen

1.1 Begriff der Architektur

In diesem Beitrag werden unter *Architektur* die Dinge verstanden, welche die (Grund-)Struktur eines Systems definieren. Mit Struktur sind dabei nicht nur die statischen Aspekte eines Systems wie z.B. Komponenten, ihre Schnittstellen und Beziehungen untereinander gemeint, sondern auch dynamische Aspekte wie etwa die Kommunikation zwischen den Komponenten. Die Struktur eines Systems ergibt sich aus einer Reihe von sich ergänzenden und überlagernden Teilstrukturen (beispielsweise die physische Komponentenstruktur, die logische Paketstruktur oder die Verteilungsstruktur). Sie wird folglich aus einer Reihe von Sichten – d.h. einzelnen Arbeitsprodukten – beschrieben, die zusammen die Architektur definieren (vgl. die ähnliche Definition in [BK99], [BCK98], [CN99])

Mit der Definition der Arbeitsprodukte, die zur Darstellung einer DV-Architektur verwendet werden, wird erstmals eine durchgängige, ausreichende und getestete Modellierung für Architekturen vorgeschlagen. Die Arbeitsprodukte werden im 4. Abschnitt näher betrachtet und anschließend in den Phasenprozess eingeordnet.

Eine ausführliche Darstellung der Rolle der Architektur in den Anwendungsentwicklung findet sich in [FB01].

1.2 Ziele einer Architektur

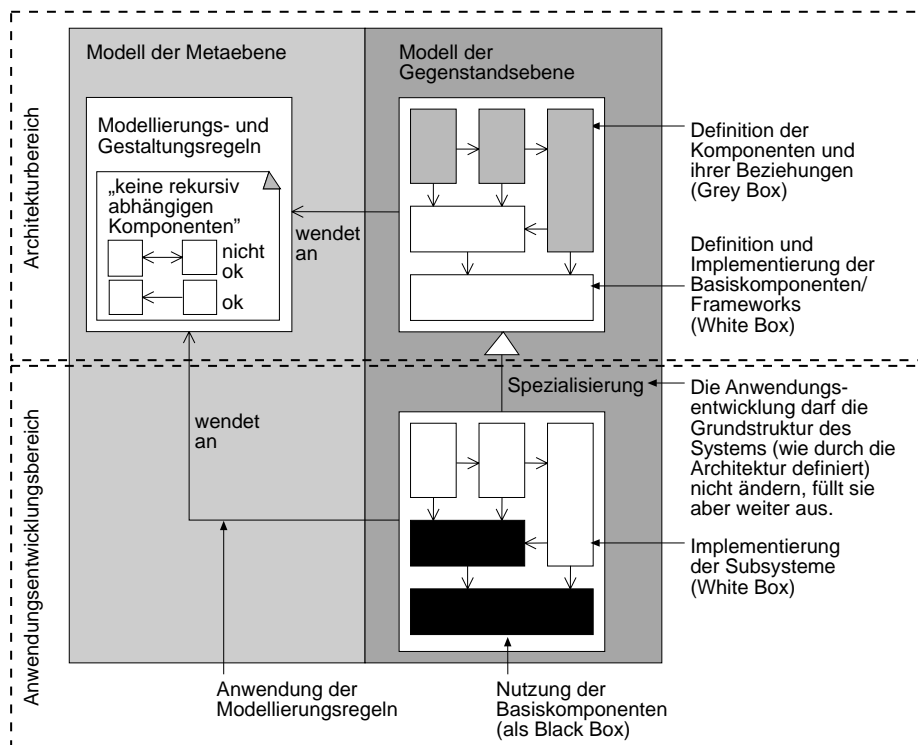
Das Ziel einer Architektur ist es sicherzustellen, dass das spätere System die Anforderungen erfüllt (*utilitas*), robust gegenüber Änderungen ist (*firmitas*) und eine gewisse „Schönheit“ besitzt (*venustas*). Mit Schönheit ist dabei gemeint, dass ein System „seine Funktion durch seine Form kommuniziert“. Diese allgemeine Definition von „Architektur“ in der Encyclopaedia Britannica [EB78] ist auch für die Softwaretechnik unverändert gültig (zur Robustheit siehe z.B. [SD00]). Die Begriffe *utilitas*, *firmitas* und *venustas* fassen die drei wichtigsten Fragen zusammen, der sich eine Architektur täglich stellen muss: Wird das System damit die Anforderungen erfüllen?

Wird es robust gegenüber möglichen Änderungen sein? Werden die Anwendungsentwickler die Architektur und die Anwender das System intuitiv nutzen können?

1.3 Einfluss der Architektur auf das System

Eine Architektur bestimmt die Struktur eines Systems auf zwei Ebenen. Zum einen legt die Architektur das Modell des späteren Systems in einem gewissen Umfang fest und definiert damit die Gegenstandsebene, u.a. durch die Strukturierung des Systems in bestimmte Subsysteme und durch die Entwicklung von Basiskomponenten wie z.B. einem Persistenzframework. Zum anderen definiert sie Regeln, die bei der Entwicklung des Systems einzuhalten sind. Damit definiert sie die Metaebene der Entwicklung, z.B. durch Programmierrichtlinien oder Muster. Diesen Sachverhalt stellt Bild 1 näher dar.

Bild 1: Einfluss der Architektur auf die Anwendungsentwicklung
(die Pfeile lehnen sich an die UML Notation an)



2 Abgrenzung

Der Begriff Architektur wird für eine Vielzahl von Systemen verwendet. Grundsätzlich können wir zwischen der Architektur von DV-Systemen (mit Hard- und Software) und sozialen Systemen (mit Organisationen, Prozessen und Ressourcen)

unterscheiden. Im Kontext der Entwicklung von DV-Systemen sind vier Architekturen von Bedeutung:

- Die *Geschäftsarchitektur* legt die Grundstrukturen des Geschäfts durch die Definition von Geschäftszielen, Prozessen, Organisationsstrukturen und Ressourcen fest. Für die Erstellung von betrieblichen Informations- und Unterstützungssystemen ist sie eine zentrale Informationsquelle.
- Die *Projektarchitektur* definiert die Prozesse, Organisation und Ressourcen des Projekts. Sie ist Voraussetzung für das reibungslose Funktionieren des Projekts.
- Die *Systemarchitektur* definiert die Hard- und Software des zukünftigen technischen Systems (das zur Unterstützung der Abläufe im Geschäft – beschrieben durch die Geschäftsarchitektur – verwendet wird).
- Die *Entwicklungsarchitektur* definiert die Hard- und Software der Entwicklungsumgebung, also des Systems, mit dem das Zielsystem erstellt wird. Ggf. tritt hierzu noch die Architektur für das Testsystem.

Die im folgenden vorgestellten Arbeitsprodukte dienen der Beschreibung der Architektur eines technischen Systems, d.h. sie können zur Definition der System- und Entwicklungsarchitektur verwendet werden (zur Modellierung der Architektur von sozialen Systemen siehe z.B. [MD99]).

3 Arbeitsprodukte zur Beschreibung der Architektur von DV-Systemen

Die beiden zentralen Teile einer Architektur eines DV-Systems sind die softwaretechnische Architektur und die Infrastrukturarchitektur.

Die *Infrastrukturarchitektur* erfasst die holistische, operationale Sicht auf das System. Hierzu zählen unter anderem das technische System (mit Hardware, Plattformen, Lokationen, Verbindungen), die Platzierung der Softwarekomponenten in Rahmen des technischen Systems, die Konfiguration und das Management des Systems (Kapazitätsplanung, Softwareverteilung, Datensicherung und Wiederanlauf).

Die *softwaretechnische Architektur* erfasst die Softwarekomponenten, die auf den Hardwarekomponenten ausgeführt werden, d.h. die funktionale Sicht auf das System. Hierzu gehören unter anderem die Struktur und Aufteilung der Softwarekomponenten, die Schnittstellen der Komponenten, die Beziehungen zwischen den Komponenten und die Zusammenarbeit der Komponenten miteinander.

Bild 2 stellt die Arbeitsprodukte im Überblick dar, die in den Bereich der Systemarchitektur fallen. Arbeitsprodukte definieren die fassbaren (Teil-)Ergebnisse, d.h. „was“ für eine Architektur getan werden muss – unabhängig vom zeitlichen „wann“ (zur Bedeutung von Arbeitsprodukten siehe [IBM97]). Die Auswahl dieser Arbeitsprodukte zur Definition einer Architektur beruht auf den praktischen Erfahrungen, die in vielen Projekten gemacht wurden und die sich in mehreren Schritten herauskristallisiert haben (siehe z.B. die Entwicklungsmethoden der IBM in [IBM00], [Yo99], [IBM97]). Im Folgenden werden die wichtigsten Pakete von Arbeitsprodukten dargestellt, und anschließend wird jedes einzelne Arbeitsprodukt näher erläutert. Um eine einheitliche Bezeichnung der Arbeitsprodukte zu gewährleisten, wird im folgenden immer auch der englische Name verwendet.

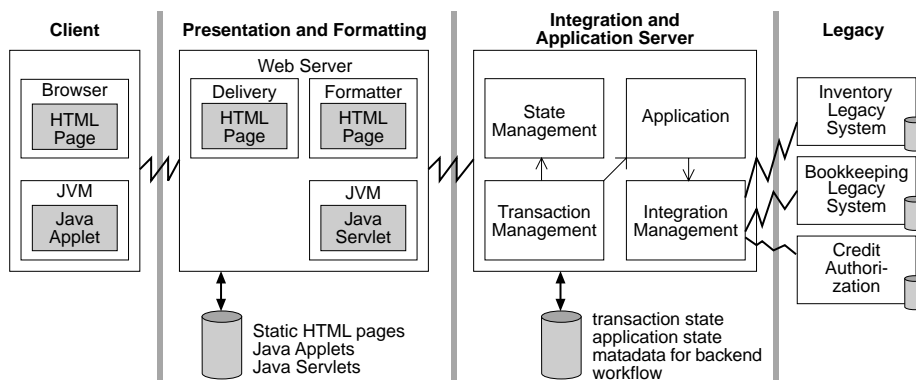
Eine Architektur muss insgesamt die Anforderungen erfüllen, die durch eine Reihe von Arbeitsprodukten im Paket „Anforderungen“ festgehalten werden (*utilitas*).

Bei der Definition einer Architektur ist es sinnvoll, auf bewährte Konzepte zurückzugreifen – die gezielte Auswahl von Referenzarchitekturen ist daher ein eigenes Arbeitsprodukt (*reference architecture fit/gap analysis*).

Schließlich muss eine Architektur überprüft werden. Einzelne Fragestellungen können ggf. durch Prototypen beantwortet werden. Darüber hinaus ist aber eine Überprüfung der Architektur hinsichtlich der qualitativen Anforderungen notwendig (*service level characteristic analysis*), wie auch gegenüber den Anforderungen insgesamt (*viability assessment*).

Die zentralen Arbeitsprodukte werden im folgenden durch ein Beispiel näher erläutert. Dieses basiert auf einem Online-Shop-System. Über dessen Komponenten gibt das Architecture Overview Diagram in Bild 3 eine Übersicht.

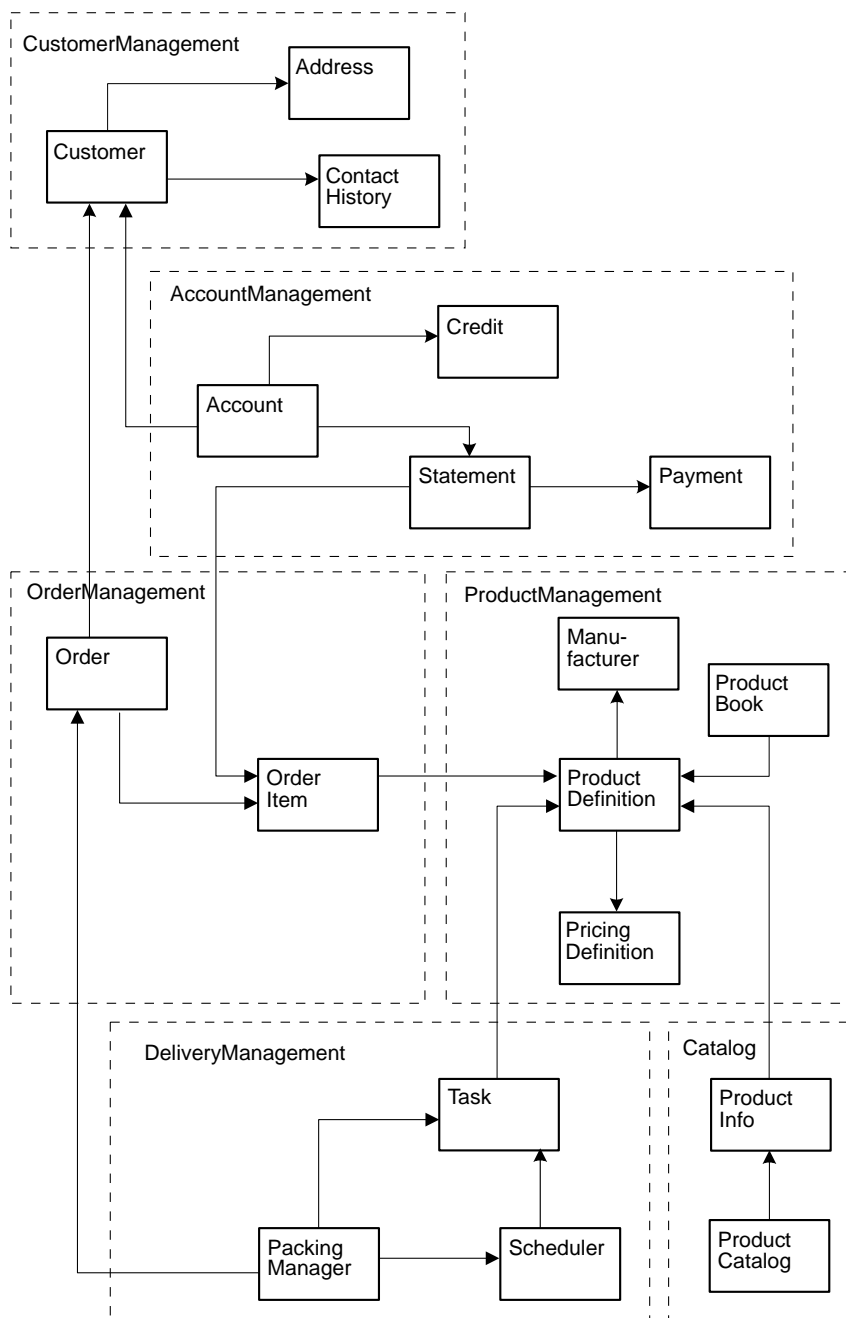
Bild 3: Ein Architecture Overview Diagram für ein Online-Shop-System. Es stellt sowohl operationale Aspekte als auch ausgewählte Komponenten (funktionale Aspekte) dar.



3.1 Softwaretechnische Architektur

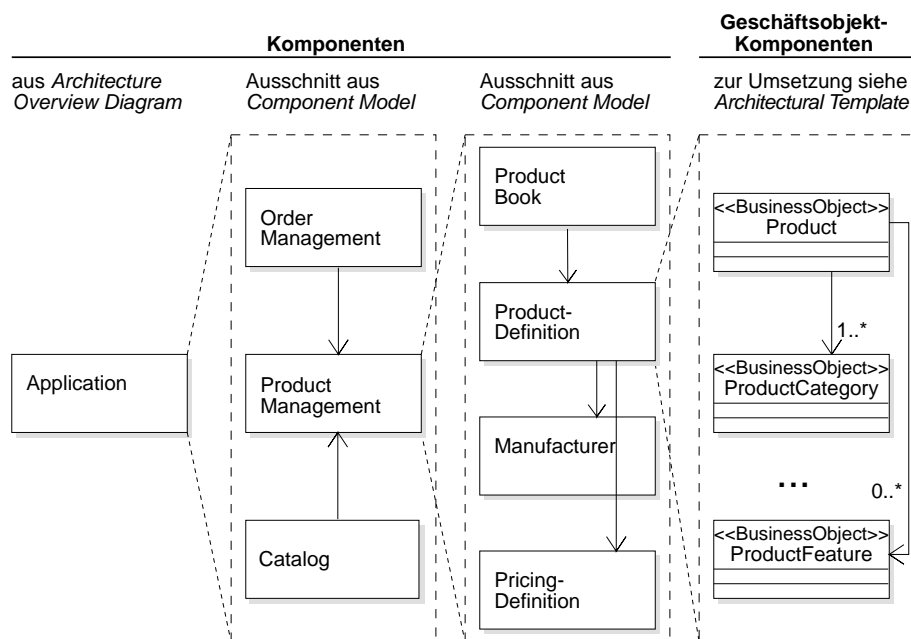
Die softwaretechnische Architektur, die den funktionalen Aspekt erfasst, gliedert das System in Komponenten. Dies sind funktionale Einheiten, die eine Schnittstelle, ein Verhalten und einen internen Zustand haben. Komponenten können miteinander in Beziehung stehen; solche Beziehungen sind die Nutzung oder Spezialisierung anderer Komponenten. Komponenten können wiederum andere Komponenten enthalten. Diese Definition lehnt sich an die Definition eines Objekts an, und Objekte sind die Blätter einer solchen Komponentenstruktur. Damit können zur Beschreibung der softwaretechnischen Architektur in Form eines Komponentenmodell (*component model*) die bekannten Diagramme der UML verwendet werden (Klassendiagramm und Kollaborations- bzw. Interaktionsdiagramme – siehe Bilder 4 und 6). Der Vorteil dieser Definition liegt darin, dass Komponenten eine klare Semantik haben und dass bestehende Notationen zur Darstellung eines Komponentenmodells verwendet werden können.

Bild 4: Komponentenmodell eines Online-Shop-Systems. Dieses Komponentenmodell ist ein Ausschnitt der Verfeinerung der „Application“ Komponente aus dem Architecture Overview Diagram. Die Pfeile stellen „nutzt“ Beziehungen dar (UML Klassendiagramm-Notation); sie sind für die Anwendungsentwickler verbindlich.



Die Definition ermöglicht es, Komponenten auf unterschiedlichen Abstraktionsebenen zu handhaben (Bild 5). Jede Abstraktionsebene ist für sich lesbar und verbirgt die darunter liegenden Details. So kennt z.B. das Modell auf der Entwurfsebene Geschäftsobjekte, die als Klassen modelliert werden. Auf der Implementierungsebene wird die Funktionalität eines solchen Geschäftsobjekts dann durch eine Menge von Klassen, d.h. durch eine Komponente realisiert. Dadurch besitzen die Modelle einerseits einen sinnvollen Abstraktionsgrad und bleiben aussagefähig, indem sie nicht durch (immer wiederkehrende) Implementierungsdetails überfrachtet werden. Andererseits existiert eine klare Umsetzung des Modells auf eine Implementierung, und die Modelle haben eine klare Bedeutung. Insbesondere bei großen Projekten bzw. bei umfangreichen Systemen ist die durchgängige Erstellung eines Modells auf der Implementierungsebene zu umfangreich, zu detailliert und nicht lesbar. Ohne die Nutzung des obigen Komponentenkonzepts und der Verwendung von unterschiedlichen Abstraktionsebenen in der Modellierung besteht die Gefahr, dass entweder die Modelle zu detailliert und umfangreich sind (wenn jede Klasse im Modell mit einer Klasse im Code korrespondiert) oder dass die Modelle keinen Bezug zur Implementierung haben und damit eine rein akademische Übung bleiben.

Bild 5: Unterschiedliche Abstraktionsebenen von Komponenten



Eine solche Umsetzung von Objekten der Modellierungsebene auf Komponenten der Implementierungsebene setzt voraus, dass klare Umsetzungsregeln definiert sind. Die Definition solcher Umsetzungs- und Modellierungsregeln ist eine Aufgabe der Architektur. Die Regeln werden in der Form von Architekturschablonen bzw. Architekturmustern (*architectural templates*) formuliert. Sie stellen Anleitungen für den Anwendungsentwickler dar, wie bestimmte Aspekte zu lösen bzw. zu implementieren

sind (Metaebene, vgl. Bild 1). Die Notation der Architekturmuster erfolgt analog zu Patterns (wie z.B. in den Büchern [Ga96], [Bu96], [Co00]).

Bild 6: Beispiel für ein Interaktionsdiagramm zwischen Komponenten. Die Interaktionen im Komponentenmodell stellen nicht eine Nachricht, sondern ein Bündel von Nachrichten dar.

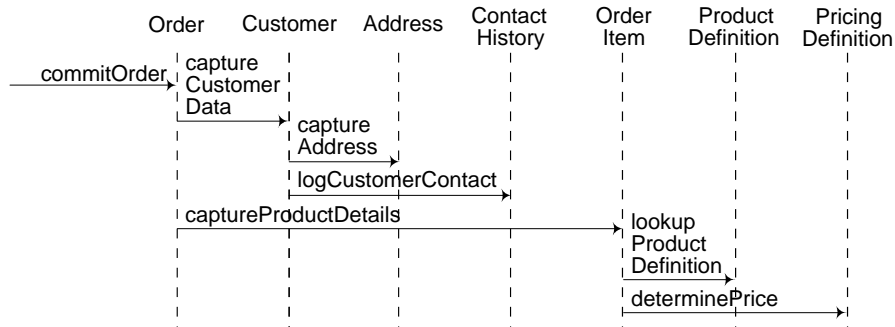
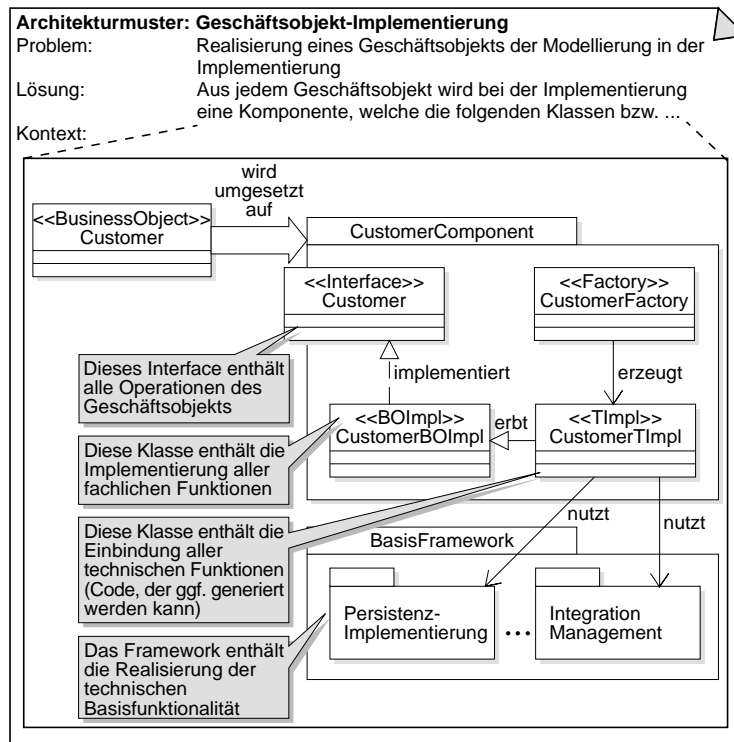


Bild 7: Ausschnitt aus einem Architekturmuster, das festlegt, wie Geschäftsobjekte implementiert werden. Dieser Anleitung muss ein Entwickler bei der Umsetzung der Klasse folgen. Architekturmuster werden wie „normale“ Muster aufgeschrieben (Name, Problem, Kontext, verwandte Muster, Lösung).



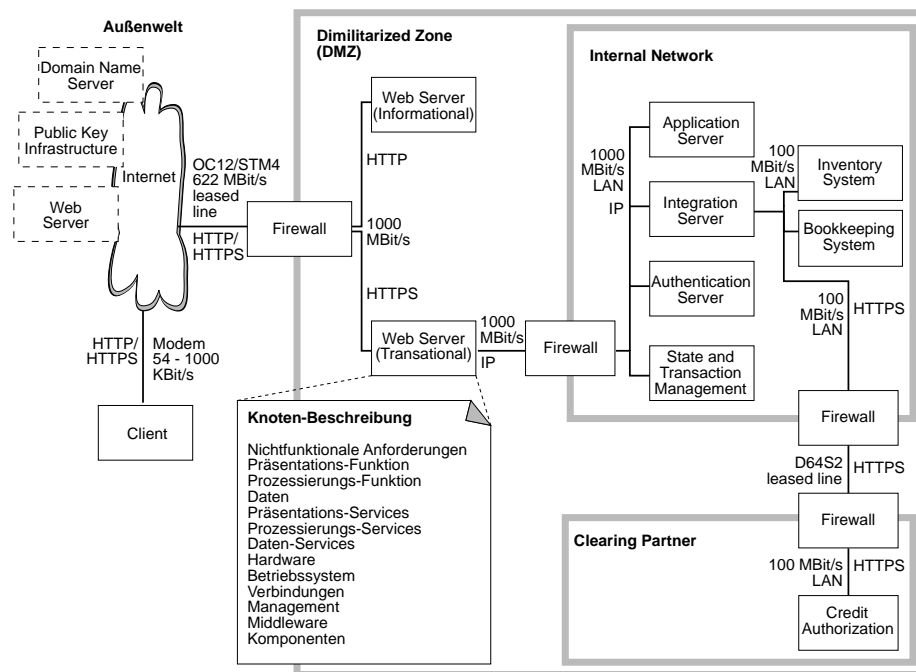
Zusätzlich zu den Architekturmustern müssen ggf. technische Basiskomponenten bzw. Frameworks (z.B. solche für die Persistenz oder Verteilung), auf denen die Um-

setzung beruht und deren Schnittstellen von den Anwendungskomponenten genutzt werden, definiert und implementiert werden (Gegenstandsebene). Bild 7 stellt dies ausschnittsweise dar. Die Umsetzung der Architekturmuster kann ggf. durch Werkzeuge unterstützt werden (Prüfung auf Einhaltung der Regeln, automatische Umsetzung der Objekte der höheren Abstraktionsebene auf Objekte der Implementierungsebene). Diese Werkzeuge müssen dann im Rahmen der Architekturarbeit erstellt bzw. gekauft werden.

3.2 Infrastrukturarchitektur

Das Operationale Modell (*operational model*) bildet den Kern der Infrastrukturarchitektur. Zum Operationalen Modell gehören Diagramme, welche die Topologie und geographische Verteilung des Systems darstellen (siehe Bild 8). Außerdem werden die Knoten, Netzwerkverbindungen und Anwender des Systems spezifiziert, indem die Funktionalität (und später die Services) des Knotens beschrieben werden (z.B. in Form einer Tabelle). Später wird das Operationale Modell um eine Festlegung konkreter (Middleware-)Produkte erweitert, und bei den Netzverbindungen sind die entsprechenden Protokolle anzugeben. Zusätzlich kann die Vollständigkeit des Operationalen Modells durch Szenarien (Walkthroughs) überprüft werden. Solche Szenarien (in Textform oder als Interaktionsdiagramm) zeigen den Fluss einer Aktion eines UseCases vom Anwender durch das System zum Anwender zurück.

Bild 8: Ein operationales Modell auf der logischen Ebene. Eingezeichnete logische Knoten können auf einer physischen Ebene durch Bündel (vertikale Skalierung) realisiert und ggf. logische Komponenten auf einem physischen Rechner platziert werden. Jeder der eingezeichneten Knoten wird durch eine detaillierte Beschreibung näher spezifiziert.



Indem die Komponenten auf den Knoten des Operationalen Modells platziert werden, wird die Verbindung zwischen dem Komponentenmodell und dem Operationalen Modell hergestellt. Die auf einem Knoten platzierten Komponenten erfüllen dann mit ihren Leistungen die Services, wie sie im operationalen Modell definiert wurden. Die Angabe, welche Komponenten auf welchen Knoten liegen, erfolgt im operationalen Modell. Komponentepakete (*deployment units*) gruppieren dabei die Komponenten, die zusammen ausgeliefert und auf einem Knoten platziert werden.

Das Operationale Modell wird durch einen Plan zur Softwareverteilung (*software distribution plan*) ergänzt. Dieser beschreibt, wie die Deployment Units auf die Knoten verteilt werden. Ebenfalls eng mit dem Operationalen Modell verbunden ist der Plan zum System Management. Er stellt dar, welche Knoten verwaltet werden müssen und durch welche Prinzipien, Prozesse, Werkzeuge und Rollen dies bewerkstelligt wird.

3.3 Überprüfung der Architektur

Da die Architektur eine zentrale Rolle spielt, sollte in mehreren Schritten überprüft werden, ob sie die Anforderungen erfüllen kann. Während der Architekturentwicklung können Prototypen (*technical prototypes*) zur Klärung einzelner technischer Aspekte genutzt werden. Später muss gezielt kontrolliert werden, ob die in den Nicht-Funktionalen Anforderungen festgelegten Qualitätsanforderungen durch eine Anwendung auf Basis der konzipierten Architektur erreicht werden können (*service level characteristic analysis*). Schließlich ist zum Ende hin die Tragfähigkeit der ganzen Architektur bezüglich der gesamten Anforderungen zu überprüfen (*viability assessment*). Neben statischen Überprüfungen wie z.B. Reviews kommen hierfür wiederum Prototypen in Frage. Durch eine prototypische Implementierung einer Anwendungskomponente (oder bei größeren Systemen durch eine Anwendung) können sowohl einzelne Qualitätsmerkmale (Durchsatz) als auch die Tragfähigkeit des gesamten Architekturkonzepts überprüft werden (vgl. [Ba98], [BCK98]).

3.4 Eingehende Arbeitsprodukte

Neben den funktionalen und nicht-funktionalen Anforderungen (*use cases, usability requirements, nonfunctional requirements*) sowie der Definition der Systemgrenze (*system context*) stellen insbesondere die derzeitige Infrastruktur (*current IT environment*) und die im Unternehmen (oder bei angebotenen Partnern) geltenden Standards (*current IT standards*) wichtige Anforderungen dar, die im Rahmen der Architektur berücksichtigt werden müssen. Standards und aktuelle Infrastruktur sind oftmals bereits beschrieben, so dass diese Arbeitsprodukte nicht im Projekt erstellt, wohl aber als Anforderungen betrachtet werden müssen.

Obwohl mögliche zukünftige Änderungen, die explizit nicht Gegenstand des aktuellen Projektauftrags sind, ihrer Definition nach keine Anforderungen darstellen, sollten diese potentiellen Änderungen dennoch festgehalten werden (*change cases*). Bei der Entwicklung der Architektur sind häufig Entscheidungen zu treffen, die vom Implementierungsaufwand bzw. von den Kosten her identisch sind. In solchen Fällen kann die Kenntnis solcher möglichen zukünftigen Änderungen als Entscheidungshilfe dienen.

3.5 Referenzarchitektur

Eine Referenzarchitektur ist ein Muster für eine Architektur und umfasst die funktionalen wie operationalen Aspekte einer Architektur für eine bestimmte Klasse von Anwendungen. Konkrete Ausprägungen der Referenzarchitektur sind in einer Reihe von Projekten erfolgreich eingesetzt worden. Die Referenzarchitektur ist dann als eine generische Zusammenfassung der projektspezifischen Lösungen entstanden. Eine solche Referenzarchitektur (beispielsweise für e-Shop-Anwendungen) besitzt einen hohen Wert für ein Projekt: sie ist ein Muster für eine gesamte Architektur, die sich in konkreten Projekten bewährt hat. Die explizite Suche nach Referenzarchitekturen und deren Evaluierung (*reference architecture fit/gap analysis*) sollte daher am Anfang einer Architekturentwicklung stehen.

3.6 Dokumentation von Architekturentscheidungen

Häufig vernachlässigt aber wichtig ist die Dokumentation zentraler Architekturentscheidungen (*architectural decisions*). Dies kann eine Datenbank oder ein Textdokument sein, in dem Entscheidungen und Prinzipien in Form eines Formulars abgelegt werden (für jede Entscheidung werden u.a. Problem, Entscheidung, Alternativen mit ihren Konsequenzen und Entscheidungshintergründe erfasst). Dieses Dokument ist nicht nur für die Dokumentation der Entscheidungen wichtig, sondern es hilft auch, die Architektur später zu verstehen und zu vertreten. Es stellt den Leitfaden für das Architekturteam und die Anwendungsentwickler dar. Nicht zuletzt kann dadurch vermieden werden, dass schon adressierte Probleme immer wieder diskutiert werden. Wichtig ist es jedoch, nicht einfach alle, sondern nur die wenigen zentralen (Grundsatz-)Entscheidungen zu dokumentieren.

4 Fazit

Eine Architektur bestimmt die Struktur eines Systems. Die explizite Definition einer Architektur ist unerlässlich, um die Systemstruktur nicht dem Zufall zu überlassen, sondern um sie gezielt zu gestalten. Aufgabe einer Architektur ist es, die Grundlagen für ein System (bzw. für eine unternehmensweite Systemfamilie) so zu legen, dass es die Anforderungen erfüllt, robust gegenüber Änderungen und verständlich ist. Die in diesem Artikel vorgestellten Arbeitsprodukte sind geeignet, eine Systemarchitektur umfassend bis hinunter auf Detailebene zu modellieren. Diese Arbeitsprodukte werden von Architekten erstellt und müssen von den Anwendungsentwicklern als Nutzer der Architektur verstanden werden. Für ein konkretes Projekt ist es meist nicht erforderlich, alle vorgestellten Arbeitsprodukte als jeweils einzelne Dokumente zu erstellen, aber das Komponentenmodell, das Operationale Modell und die Architekturmuster sollten auf keinen Fall fehlen

Literaturverzeichnis

- [Ba98] M. R. Barbacci et. al: Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis, Software Engineering Institute, Technical Report CMU/SEI-97-TR-029/ESC-TR-97-029, Pittsburgh, 1998

- [BCK98] L. Bass, P. Clements, R. Kazman: Software Architecture in Practice, Addison-Wesley Publishing Company, Reading, 1998
- [BK99] L. Bass, R. Kazman: Architecture-Based Development, Carnegie Mellon University, Software Engineering Institute, Technical Report CMU/SEI-99-TR-007/ESC-TR-99-007, Pittsburgh, 1999
- [Bu96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: Pattern Oriented Software Architecture, Wiley & Sons, Chichester, 1996
- [CN99] P. C. Clements, L. M. Northrop: Software Architecture: An Executive Overview, Carnegie Mellon University, Software Engineering Institute, Technical Report CMU/SEI-96-TR-003/ESC-TR-96-003, Pittsburgh, 1999
- [Co00] J. Coplien, J. Vlissides, R. Martin, N. Harrison: Pattern Languages of Program Design, Volume 1-4, Addison-Wesley, Reading, 1995-2000
- [EB78] Encyclopaedia Britannica Inc. (Hrsg.): The New Encyclopaedia Britannica, Macropaedia, Volume 1, Stichwort Architecture, Seiten: 1088-1115, Encyclopaedia Britannica Inc, Chicago, 1978
- [FB01] M. Foegen, J. Battenfeld: Die Rolle der Architektur in der Anwendungsentwicklung, Informatik Spektrum, Band 24, Seiten 290-301, Springer Verlag, Heidelberg, 2001
- [Ga96] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Addison-Wesley, Reading, 1996
- [MD99] D. W. McDavid: A standard for business architecture description, IBM Systems Journal, Vol. 38, No. 1, Seiten 12ff, IBM, 1999
- [IBM97] IBM, Developing Object Oriented Software, An Experience-Based Approach, Prentice Hall, Upper Saddle River, 1997
- [IBM00] IBM: IBM Global Services Method 3.0, Familie von Vorgehensweisen für Serviceprojekte, IBM internes Dokument, Dokument-Nummer ZZ81-0045-00, 2000
- [SD00] J. Siedersleben, E. Denert: Wie baut man Informationssysteme? Überlegungen zur Standardarchitektur, Informatik Spektrum, Band 23, Seiten 247ff, Springer Verlag, Heidelberg, 2000
- [Yo99] R. Youngs, D. Redmond-Pyle, P. Spaas, and E. Kahan: A standard for architecture description, IBM Systems Journal, Vol. 38, No. 1, Seiten 32ff, IBM, 1999