Konfigurationsmanagement komponentenorientierter betrieblicher Anwendungen

Fachgebiet Operations Research - TU Darmstadt
Prof. Dr. W. Domschke

Steffen Becker, 1039465

Sommersemester 2003

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit Fragen des Konfigurationsmanagements bei der betrieblichen Anwendungsentwicklung mit Softwarekomponenten. Besondere Schwerpunkte liegen dabei in der Generierung von Konfigurationen und deren systematischer Bewertung im Bezug auf ihre Nutzen und ihre Kosten. Die wichtigsten Elemente bei der Durchführung des Konfigurationsmanagements werden in Form von Prozessen dargestellt, darunter ein Prozess zur Suche nach Konfigurationen sowie deren Nutzenbewertung mit Hilfe des AHP.

Inhaltsverzeichnis

Abbildungsverzeichnis II

Tabellenverzeichnis II

1 Einleitung 1

2 Begriffsdefinitionen 5

3 Bausteine eines Auswahlprozesses 10
3.1 Szenarien . . . 11
3.2 Vorgehensweise . . . 13
3.3 Komponentenspezifikation . . . 16
3.3.1 Eigenschaften . . . 17
3.3.2 CDM . . . 18
3.3.3 KDL . . . 18
3.3.4 Spezifikation nach Ackermann et al. . . . 19
3.4 Erweiterungen der Spezifikation nach Ackerman et al. . . . 22
3.4.1 Parametrisierung . . . 23
3.4.2 GUI Komponenten . . . 25
3.4.3 Qualitätsebene . . . 26
3.5 Knowledge Base . . . 29
3.5.1 Unternehmensinterne Repositorien . . . 31
3.5.2 Komponentenmarktplätze . . . 31
3.6 Speicherstrukturen für die Ergebnisse . . . 32
3.6.1 Baupläne . . . 33
3.6.2 Stücklisten . . . 34
3.7 Entscheidung . . . 35
3.7.1 Alternativen . . . 37
3.7.2 Szenarien . . . 38
3.7.3 Präferenzstruktur . . . 38
3.7.4 Zielsystem . . . 39
3.7.5 Entscheidungstheoretisches Verfahren . . . 40
3.7.6 Sensitivitätsanalyse . . . 41
3.8 Literaturüberblick . . . 42

4 Annahmen 44

5 Suche und Vorauswahl 47
5.1 Problembestimmung . . . 47
5.2 Bestandteile einer Konfiguration . . . 49
5.3 Alternativengenerierung . . . 51
5.3.1 Experience Factory abfragen . . . 52
5.3.2 Ermittlung gedachter Konfigurationen . . . 53
5.3.3 Vorauswahl der gedachten Konfigurationen . . . 53
5.3.4 Suche nach Komponenten . . . 54

5.3.5 Verwendung eines Distanzmaßes . . . 54
5.3.6 Auswertung der Suchergebnisse . . . 55
5.3.7 Ausscheiden einer Konfiguration aus dem Prozess . . . 55
5.3.8 Kritik . . . 56
5.4 Eigenentwicklung . . . 56
5.5 Durchführung . . . 57
5.6 Abschlussprüfung . . . 59

6 Ein Kostenmodell für Konfigurationen 60
6.1 Kosten einer Komponente . . . 61
6.2 Kosten der Konfiguration . . . 62
6.2.1 Suchkosten . . . 63
6.2.2 Bewertungskosten . . . 64
6.2.3 Beschaffung . . . 64
6.2.4 Tailoring . . . 65
6.2.5 System Volatility . . . 65
6.2.6 Adaptorenentwicklung . . . 66
6.2.7 Konnektorenentwicklung . . . 66
6.2.8 Vorhersagekosten . . . 66
6.2.9 Administrative Kosten . . . 67
6.2.10 Formel . . . 67

7 Entscheidungsprozess 68
7.1 Probleme beim Einsatz von Entscheidungsunterstützung . . . 68
7.1.1 Allgemeine Probleme . . . 68
7.1.2 Probleme beim Einsatz von MAUT . . . 70
7.2 Beschreibung des AHP . . . 71
7.2.1 Ermittlung der Präferenzen . . . 72
7.2.2 Erstellung der Vergleichsmatrix . . . 73
7.2.3 Konsistenz . . . 73
7.2.4 Ermittlung der Zielgewichte und Nutzwerte . . . 74
7.2.5 Näherungsverfahren . . . 75
7.2.6 Inkonsistenzindex . . . 75
7.3 Untersuchung des AHP . . . 77
7.3.1 Rangumkehr . . . 77
7.3.2 Skala . . . 78
7.3.3 Anzahl der Paarvergleiche . . . 79
7.3.4 Unvollständige Vergleichsmatrizen . . . 80

8 Auswahl einer Konfiguration 82
8.1 Ableitung einer Zielhierarchie . . . 82
8.1.1 Funktionalität . . . 83
8.1.2 Qualität . . . 85
8.1.3 Benutzeroberfläche . . . 88
8.1.4 Homogenität/Zusammenarbeit . . . 90
8.1.5 Zusammenfassung . . . 91

8.2 Szenarien 92
8.3 Kosten-Nutzen-Diagramm 93

9 Kritik und Ausblick **95**

Literatur **97**

Abbildungsverzeichnis

1 Komponentenmodell . . . 6
2 Bausteine des Konfigurationsmanagements . . . 11
3 Einsatzszenarien . . . 13
4 Vorgehensmodell . . . 14
5 Spezifikationsrahmen nach Turowski et al. . . . 20
6 3 Tier Architekturschema . . . 25
7 Vereinfachtes Grundmodell der Experience Factory . . . 30
8 Verschiedene Arten von Stücklisten . . . 35
9 Variantenstückliste einer Buchhaltungskomponente . . . 36
10 Beispiel einer abhängigen Konfiguration . . . 49
11 Eine einfache Konfiguration einer Buchhaltung . . . 51
12 Iterativer Suchprozess . . . 52
13 Beispiel zur Suche . . . 58
14 Kostenmodell für Konfigurationen - Kostenarten . . . 63
15 Eine einfache Zielhierarchie . . . 71
16 Eine Vergleichsmatrix . . . 73
17 Erste Ebene der Zielhierarchie . . . 83
18 Teilbaum der funktionalen Ziele . . . 84
19 Teilbaum der Qualitätsziele . . . 86
20 Teilbaum der benutzeroberflächenbezogenen Ziele . . . 89
21 Teilbaum der auf die Zusammenarbeit bezogenen Ziele . . . 90
22 Anpassung des AHP auf die Verwendung von Szenarien . . . 93
23 Ein Beispiel eines Kosten/Nutzen Diagramms . . . 93

Tabellenverzeichnis

1 Qualitätsmodell nach Bertoa und Vallecillo (2002) . . . 28
2 Verbalskala nach Saaty (1980) . . . 73
3 Kosten/Nutzen Vergleich . . . 94

1 Einleitung

Seit McIlroy (1968) zum ersten Mal die komponentenorientierte Anwendungsentwicklung als eine neue Methode zur Entwicklung von Software vorgeschlagen hat, haben sich in diesem Gebiet der Informatik einige wichtige Fortschritte ergeben. Insbesondere das Voranschreiten der entsprechenden Technologien in den letzten Jahren hat zu dieser Entwicklung einen entscheidenden Beitrag geliefert. Dabei sind es besonders die großen Softwareunternehmen wie Microsoft oder Sun, die hier durch entsprechende Investitionen die neue Technologie vorantreiben. Neuste Entwicklungen zielen beispielsweise darauf, Komponenten im weltweit verteilten Internet zur Verfügung zu stellen. Initiativen wie Microsoft.NET und Sun ONE belegen dieses Engagement eindrucksvoll.

Jedoch ist festzustellen, dass sich auch mit der Einführung immer besser werdender Basistechnologien zur Erstellung und Bereitstellung von Softwarekomponenten das Komponentenparadigma noch immer nicht auf breiter Front durchgesetzt hat. Um dies besser verstehen zu können, erscheint es sinnvoll, die Vor- und Nachteile des Komponentenansatzes zu beleuchten.

Vorteile der Komponentenorientierung ergeben sich nach Szyperski (1998, S. 3ff) insbesondere durch eine zu erwartende Verkürzung der Entwicklungszeit und damit durch eine insgesamt geringere time-to-market Zeitspanne, sowie durch höhere Qualität, Skalierbarkeit und Wartbarkeit der Anwendung.

Die Entwicklungszeit verkürzt sich durch den Wegfall eines Großteils der Implementierungsarbeiten, da weitgehend vorgefertigte Komponenten eingesetzt werden und der entsprechende Aufwand damit bereits bei der Erstellung der Komponente geleistet wurde. Darüber hinaus wird eine Kostensenkung durch diese Zeitersparnis erwartet, die das große Interesse der betroffenen Unternehmen an der Komponentenorientierung erklärt.

Eine gesteigerte Qualität ergibt sich zum einen dadurch, dass die Komponente unabhängig von ihrem späteren Einsatz getestet werden kann, so dass Testzyklen bereits frühzeitig Schwächen aufdecken können. Zum anderen ergeben sich durch den mehrfachen Einsatz der Komponente eine große Zahl an Erfahrungen mit dieser Komponente, so dass davon auszugehen ist, dass Mängel, die während der Herstellung unentdeckt geblieben sind, früher auffallen und damit beseitigt werden können.

Eine bessere Skalierbarkeit lässt sich durch bewusst redundant gehaltene Komponenten erreichen. Beispielsweise könnte eine Komponente zur Verbuchung einzelner Geschäftsvorfälle auf mehrere Computer verteilt werden, wodurch erreicht werden würde, dass deutlich mehr Vorfälle pro Zeiteinheit bearbeitet werden.

Eine Verbesserung der Wartbarkeit begründet sich in der Idee, fehlerhafte Komponenten einfach gegen fehlerbereinigte Komponenten auszutauschen bzw. fehlende Funktionalitäten in der Anwendung durch die Aufnahme weiterer Komponenten ergänzen zu können.

Es erscheint durchaus realistisch, dass die angesprochenen Vorteile genutzt werden können, wie ein Blick in andere Ingenieurswissenschaften zeigt. Die Fertigung von Maschinen aus einer großen Anzahl von Komponenten im Maschinenbau oder die Erstellung von Schaltungen aus standardisierten Bauteilen in der Elektrotechnik seien hier nur beispielhaft genannt.

Trotz dieser wichtigen Vorteile gibt es jedoch noch immer eine Reihe von Herausforderungen und Risiken bei der Einführung einer komponentenbasierten Softwareentwicklung.

Hierbei kann grundsätzlich zwischen technischen Problemen und organisatorischen Hürden unterschieden werden.

Eher technischer Natur ist beispielsweise die Frage, mittels welcher Attribute Softwarekomponenten charakterisiert werden können. Dabei wird eine Beschreibung benötigt, die den Hersteller in die Lage versetzt, Komponenten so zu spezifizieren, dass mögliche Käufer der Komponente entscheiden können, ob sie diese Komponente benötigen. Phillips und Polen (2002) werfen hierbei mit Recht die Frage auf, wie glaubwürdig solche Spezifikationen eigentlich sind und wie der Kunde die angebotenen Komponenten auf ihre Spezifikationen hin überprüfen kann.

Auch treten fachliche Probleme bei der Spezifikation von Komponenten auf. Ein unterschiedliches Verständnis der verwendeten (Fach-)Wörter innerhalb der Spezifikation kann die Suche nach geeigneten Komponenten massiv erschweren bzw. gänzlich unmöglich machen.

Des Weiteren verhindert auch die Anzahl der verfügbaren Komponenten auf den wenigen existierenden Komponentenmarktplätzen (vgl. Hahn (2002)) den effizienten Einsatz der Komponententechnologie in der Anwendungsentwicklung. Es wird in der aktuell anzutreffenden Situation daher zumeist nicht möglich sein, die gerade benötigten Komponenten zu beschaffen. Über bereits existierende Möglichkeiten, *innerhalb* eines einzelnen Unternehmens Softwarekomponenten wiederverwenden zu können, kann hier nur spekuliert werden.

Noch schwerwiegender sind wohl die Probleme auf der organisatorischen Ebene anzusehen, da die Entwicklung von Anwendungen beim Einsatz von Komponenten andere Abläufe erfordert, als dies in der klassischen Softwareentwicklung bei der Erstellung prozeduraler oder objektorientierter Programme nötig war.

Bei diesen klassischen und nach wie vor stark genutzten Vorgehensweisen liegt das Augenmerk in der Implementierungsphase hauptsächlich auf der Umsetzung der geplanten Softwarearchitektur *im Kleinen*, dass heißt, die Realisierung der benötigten Funktionalität steht im Mittelpunkt des Interesses. Diese wird in Form von Prozeduren und Datenstrukturen bzw. Methoden von Objekten realisiert. Die Erstellung einer Anwendung aus Komponenten verlagert diesen Fokus auf die Entwicklung *im Großen*, dass heißt die Zusammenführung existierender Komponenten, die die benötigte Funktionalität beinhalten, in der Art, dass die entstehende Anwendung die an sie gestellten Erwartungen erfüllt (vgl. u.a. Haines et al. (1997)).

Dieser Wandel hat weitreichenden Einfluss auf die Organisation eines mit der Erstellung von Anwendungssoftware beschäftigten Unternehmens. Die Bedeutung der Implementierer nimmt ab und an ihre Stelle tritt zunehmend ein so genannter Konfigurierer (vgl. u.a. Ritter (2000, S. 9)), der die Aufgabe hat, die notwendigen Komponenten zur Endanwendung zu kombinieren. Dabei kommt es entscheidend auf die Auswahl der Komponenten an, denn diese Auswahl birgt unter Umständen ein hohes Risikopotential.

Das auswählende Unternehmen könnte sich beispielsweise durch die Entscheidung für eine bestimmte Komponente langfristig in ein Abhängigkeitsverhältnis zu ihrem Hersteller begeben, da es z.B. bei Fehlerkorrekturen auf den Hersteller angewiesen ist (vgl. Haines et al. (1997)). Darüber hinaus besteht auch eine Abhängigkeit im Bezug neuerer Versionen der Komponente. Falls eine neue Version einer Komponente auf den Markt gebracht wird, müssen die Nutzer der alten Version dieser Komponente entscheiden, ob sie die aktualisierte Version einsetzen wollen oder nicht. Entscheiden sie sich gegen die neue Version, droht

ihnen früher oder später der Ausschluss vom Herstellersupport. Möchten sie dagegen die neue Version beziehen und einsetzen, erfordert dies heutzutage zumeist eine umfangreiche Überprüfung, ob die neue Version der Komponente die Kompatibilität mit der alten Version der Komponente wahrt. Falls sie diese nicht einhält, wird es erforderlich sein, existierende Adaptoren für diese Komponente anzupassen bzw. zu erstellen. Daher stellen Haines et al. (1997) auch fest: „This is why the component-based system approach is sometimes considered a risk transfer and not a risk reduction approach."

Weiterhin kann es aber auch schon bei der ersten Integration einer Komponente in eine Anwendung zu unvorhersehbaren Problemen kommen. Diese können unter anderem durch ungenügende oder fehlerhafte Spezifikation der Komponente entstehen oder durch nicht vorhersehbare Seiteneffekte in anderen Teilen der Anwendung.

Ein weiteres wesentliches Problem bei der Einführung einer komponentenorientierten Anwendungserstellung dürfte derzeit in fehlendem Personal bestehen. So befinden sich z.B. spezielle Vorlesungen, bei denen die Aufgaben während der Konfiguration einer Anwendung aus Komponenten behandelt werden, meistens erst in Vorbereitung. Bereits vorhandenes Personal muss in Schulungen die Unterschiede zum herkömmlichen Vorgehen erlernen.

Ein kritischer Faktor hierbei ist, dass sich kaum Arbeiten zu einer methodischen Unterstützung in diesem Bereich finden lassen. Die meisten Vorarbeiten, die sich mit der Auswahl von COTS (commercial off-the-shelf) Software beschäftigen, fassen den Begriff einer Komponente sehr weit und behandeln folglich die Auswahl einer geeigneten Datenbank oder eines geeigneten Office-Pakets.

Daher soll in dieser Arbeit ein Prozess vorgestellt werden, der auf die besonderen Erfordernisse bei der Erstellung einer Anwendung aus Komponenten eingeht. Dabei sollen vor allem die Aufgaben des Konfigurierers und - falls dies in der entsprechenden Organisation getrennt ist - die des Kompontenbeschaffers im Mittelpunkt der Betrachtung stehen. Ihnen sollen Methoden und Werkzeuge an die Hand gegeben werden, damit sie in der Lage sind, ihre Aufgaben effizient und nachvollziehbar zu erledigen.

Dabei klärt Kapitel 2 zunächst die in dieser Arbeit verwendeten Begriffe, insbesondere den der (Software-)Komponente. Im sich anschließenden Kapitel sollen die elementaren Bausteine eines Prozesses für die betroffenen Organisationseinheiten eingeführt und erläutert werden. Sie stellen die Grundlage für die restlichen Kapitel dieser Arbeit dar.

Diese Grundlagen geben dabei einen sehr ausführlichen Überblick über den aktuellen Stand der Forschung in den für diese Arbeit relevanten Teilgebieten der Komponentenorientierung. Dies ist notwendig, da die meisten Arbeiten zu den entsprechenden Themen noch nicht sehr alt sind. Daher kann nicht davon ausgegangen werden, dass sie schon zum Allgemeinwissen gehören. Außerdem wird durch die Ausführlichkeit der Darstellung der Stand der Forschung, wie er dieser Arbeit zu Grunde liegt, nachvollziehbar dokumentiert.

In Kapitel 4 werden darauf aufbauend die theoretischen Annahmen vorgestellt, auf denen diese Arbeit aufbaut. Das Kapitel 5 widmet sich im Anschluss daran der Frage, wie die für den Einsatz in der zu erstellenden oder zu wartenden Anwendung in Frage kommenden Komponenten auf Komponentenmarktplätzen oder in unternehmensinternen Komponentenrepositorien gefunden werden können. Falls die Treffermenge groß sein sollte, stellt sich ebenfalls die Frage, wie die Anzahl der Suchergebnisse durch geeignete Vorauswahl verkleinert werden kann.

Als Resultat des Suchschritts entstehen dabei Konfigurationen von Komponenten, für die in Kapitel 6 ein Kostenmodell angegeben wird. Das Modell basiert dabei auf vorhandenen Arbeiten zur Schätzung der Kosten einzelner COTS Komponenten.

Kapitel 7 stellt einige Verfahren aus der Entscheidungstheorie vor und prüft ihre Eignung im hier betrachteten Kontext. Insbesondere wird auf den AHP (s. Abschnitt 7.2) eingegangen, wobei einige der im Zusammenhang mit dem AHP häufig genannten Bedenken (s. Abschnitt 7.3) näher betrachtet werden.

Im Kapitel 8 wird darauf aufbauend eine Zielhierarchie angeben, die verwendet werden kann, um den subjektiven Nutzen einer Konfiguration für einen bestimmten Entscheider zu ermitteln. Anschließend werden die ermittelten Nutzenwerte den geschätzten Kosten der einzelnen Konfigurationen gegenübergestellt, so dass eine vernünftig begründete Entscheidung für eine zu erstellende Konfiguration getroffen werden kann.

Ein Ausblick auf weitere Arbeiten in diesem Bereich sowie eine kritische Betrachtung des vorgestellten Prozesses bilden den Abschluss dieser Arbeit.

2 Begriffsdefinitionen

Im Folgenden werden die für diese Arbeit grundlegenden Begriffe eingeführt und definiert. Dies ist insbesondere deshalb notwendig, weil einige dieser Begriffe auch nach über 30 Jahren an Forschungsarbeit noch immer keine einheitliche Bedeutung besitzen.

Zunächst soll der Begriff der Komponente bzw. genauer ausgedrückt der Softwarekomponente geklärt werden.
Szyperski (1998, S. 30) nennt hierzu drei grundlegende Eigenschaften von Komponenten:

- Komponenten sind unabhängig einsetzbare Einheiten
- Komponenten werden von Dritten zusammengesetzt
- Komponenten haben keinen persistenten Zustand

Aus diesen Eigenschaften leitet er die folgende Definition einer Softwarekomponente ab (vgl. Szyperski (1998, S. 34)):

„A software component is a unit of composition with contractually specified interfaces and explicit context dependancies only. A software component can be deployed independently and is subject to composition by third parties.“

In dieser Definition gibt es einige interessante Punkte. Eine Komponente hat gemäß dieser Definition eine oder mehrere Schnittstellen, die mittels Softwarekontrakten (meist in Form von Vor- und Nachbedingungen) beschrieben werden. Sie erfüllt ihren Kontrakt, sobald ein vorher definierter Kontext geschaffen wurde. Ein solcher Kontext kann zum Beispiel die Basistechnologie beinhalten oder weitere Komponenten, deren Dienste von der betrachteten Komponente benutzt werden. Ansonsten kann sie unabhängig vom angedachten Einsatzzweck installiert werden und wird dann von Dritten, also nicht vom Hersteller der Komponente, in neue Anwendungen integriert.

In dieser Arbeit soll allerdings der Definition aus Ackermann et al. (2002) der Vorzug gegeben werden. Die Autoren dieses Memorandums zur „Vereinheitlichten Spezifikation von Fachkomponenten“ haben sich auf die folgende Definition geeinigt:

„Eine Komponente besteht aus verschiedenartigen (Software-)Artefakten. Sie ist wiederverwendbar, abgeschlossen und vermarktbar, stellt Dienste über wohldefinierte Schnittstellen zur Verfügung, verbirgt ihre Realisierung und kann in Kombination mit anderen Komponenten eingesetzt werden, die zur Zeit der Entwicklung nicht unbedingt vorhersehbar ist.“

Interessant gegenüber der Definition von Szyperski ist der Einschluss aller Artefakte, die zu einer Komponente gehören. Hierzu gehören neben der eigentlichen Komponente im Binärformat auch die Dokumentation der Komponente, Grafik- bzw. Textressourcen oder Testfälle. Die Komponenten sollen darüber hinaus vermarktbar sein, so dass sie als unabhängige Güter gehandelt werden können. Dies kann sowohl unternehmensintern als auch extern auf Komponentenmarktplätzen geschehen. Die restlichen Eigenschaften des Komponentenbegriffs decken sich mit Szyperskis Forderungen.

Die zitierte Spezifikation sieht des Weiteren noch die Einführung des Worts „Fachkomponente“ vor. Unter einer Fachkomponente versteht das Memorandum eine Komponente, *„die eine bestimmt Menge an Diensten einer betrieblichen Anwendungsdomäne anbietet“*. Da sich diese Unterscheidung zum einen aktuell in der Diskussion befindet und zum anderen keinen Einfluss auf diese Arbeit hat, soll in dieser Arbeit nur der Begriff der (Software-) Komponente benutzt werden.

Nachdem der Begriff der Komponente definiert ist, ist es notwendig das Komponentenmodell vorzustellen, auf dem die Ausführungen in dieser Arbeit basieren, da dieses Modell unter anderem Auswirkungen darauf hat, wie zusammengesetzte Komponenten zu behandeln sind und damit, wie der Konfigurationsbegriff zu fassen ist.

Diese Arbeit legt das Komponentenmodell von Shaw und Garlan (1996) zugrunde. Abbildung 1 stellt das Modell so dar, wie es von Overhage und Thomas (2003) in leicht modifizierter Form verwendet wurde.

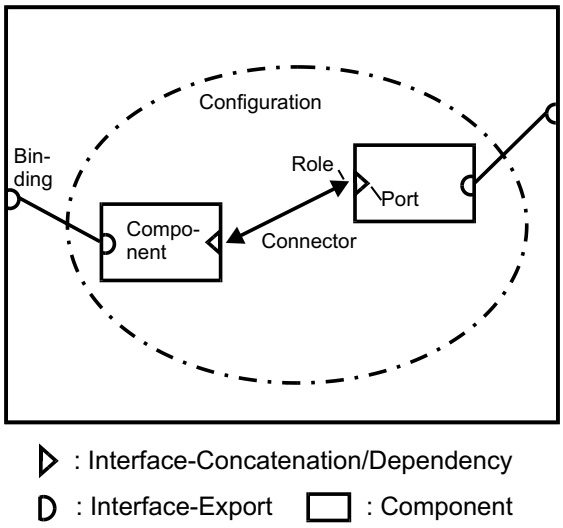

Abbildung 1: Komponentenmodell (aus Overhage und Thomas (2003))

In diesem Komponentenmodell werden die Komponenten zu so genannten Konfigurationen zusammengesetzt, indem Konnektoren benutzt werden, die die Verbindung zwischen zwei Komponenten herstellen. Konnektoren werden in der Literatur häufig auch unter dem Begriff Glue (Kleber) oder Glue-Code eingeführt.

Jede der Komponenten stellt an ihrer Schnittstelle definitionsgemäß ihre Dienste anderen Komponenten zur Verfügung. Komponenten, die diese Dienste nun in Anspruch nehmen wollen, können also die Funktionalität durch einen Aufruf einzelner Funktionen der Schnittstelle der Zielkomponente nutzen. Damit dient dieses Interface gewissermaßen als Anknüpfungspunkt für eingehende Kommunikation mit dieser Komponente. Shaw und Garlan bezeichnen daher diese Kommunikationsstellen auch als Ports (Häfen), an dem Konnektoren angedockt werden können.

Die Kommunikation zweier oder mehrerer Komponenten findet unter bestimmten, festgelegten Abläufen bezüglich der Reihenfolge der einzelnen Aufrufe der verschiedenen Dienste der beteiligten Komponenten statt. Die erlaubten Abläufe werden hierbei als Protokoll

bezeichnet.

Durch die Verwendung eines bestimmten Protokolls werden die beteiligten Kommunikationspartner, in diesem Fall also die beteiligten Komponenten, in bestimmte Rollen gedrängt. So bestimmt beispielsweise die Kommunikation zweier Komponenten, die Geschäftspartner abbilden sollen, wer sich in der Rolle des Anbieters und wer sich in der Rolle des Nachfragers befindet.

Eine komplexe Konfiguration, also der Zusammenschluss mehrerer Komponenten mittels Konnektoren, kann nach außen wieder als Komponente auftreten, indem sie Schnittstellen exportiert, das heißt, vorhandene Funktionalität wird an der Schnittstelle der zusammengesetzten Komponente wieder zur Verfügung gestellt.

Die Konfiguration einer Anwendung kann im Gegensatz zu der vorangegangenen Definition auch als Aufgabe verstanden werden (aus Ritter (2000)). Wird dieser Ansatz verfolgt, dann kann die Konfiguration verstanden werden als die „Aufgabe, bei der

- eine feste Menge vordefinierter Komponenten
- eine Menge von Kompatibilitätsinformationen, die angeben, wie Komponenten verbunden werden dürfen
- eine Menge von anwendungsspezifischen Konfigurationsplänen, die Prozeduren zur Auswahl und Komposition von Komponenten definieren, und
- eine Menge von Anforderungen durch den späteren Nutzer/Anwender eines Systems

vorgegeben werden, und zu deren Zielerreichung entweder

- eine Anwendung der Konfigurationspläne auf Grundlage der vorgegebenen Komponenten gefunden werden muss, die alle Kompatibilitätsrestriktionen und Anforderungen des Nutzers/Anwenders einhält, oder
- gezeigt werden muss, dass es eine solche Anwendung, die die Restriktionen und Anforderungen einhält, nicht gibt."

Andere Autoren fügen dieser Aufzählung noch die Aufgabe hinzu, die Parametrisierung (s. Abschnitt 3.4.1) der Komponenten vorzunehmen.

Ritter (2000) unterscheidet weiterhin zwischen gebundenen und generischen Konfigurationen. Dabei sind gebundene Konfigurationen eindeutige Konfigurationen, die unabhängig vom jeweiligen Kontext sind. Die Konfiguration einer fertiggestellten und ausgelieferten Anwendung ist beispielsweise eine gebundene Konfiguration. Demgegenüber sind generische Konfigurationen zwar ebenfalls eindeutig beschreiben, sie sind jedoch noch abhängig von speziellen Kontexten. Eine Anwendung, die durch eine Konfiguration, die Varianten berücksichtigt, beschrieben wird, ist solange generisch, bis die Variantenentscheidungen getroffen wurden. Sind diese Entscheidungen gefallen, handelt es sich wieder um eine gebundene Konfiguration.

Der Begriff des *Komponentenkonfigurationsmanagements* wird in der Literatur auch nicht immer eindeutig verwendet, obwohl die intuitive Semantik, nämlich der Planung,

Durchführung und Kontrolle der Aufgaben, die bei der Konfiguration anfallen, offensichtlich erscheint.

Szyperski (1998, S. 334) fasst unter diesem Begriff verschiedenste Aufgaben zusammen. Dazu gehören die Beobachtung und aktive Suche nach neuen oder besser geeigneten Komponenten auf verfügbaren Komponentenmarktplätzen, der Einsatz neuer Komponenten in bestehende Anwendungen und die Migration bereits existierender Anwendungen auf einen komponentenorientierten Systementwurf.

Auf die letzte Aufgabe richtet er ein besonderes Augenmerk. Ein besonders großes Problem bei der Umwandlung solcher so genannter Legacy (geerbter) Anwendungen besteht zumeist darin, zu analysieren, auf Basis welcher Annahmen die Altanwendung eigentlich erstellt wurde, da hier häufig mit mangelhafter Dokumentation gearbeitet werden muss. Zumeist kommt erschwerend hinzu, dass die ursprünglichen Entwickler der Programme entweder nicht mehr zur Verfügung stehen oder selbst nicht mehr wissen, wie die Anwendung eigentlich zusammenarbeitet. Als Beispiel soll die Umwandlung des SAP R/3 Systems auf eine komponentenorientierte Basis betrachtet werden. Dieses System besitzt eine sehr große Datenbasis, die innerhalb einer Datenbank abgelegt ist. Allein die Analyse des Datenflusses zwischen den einzelnen Teilen der Anwendung dürfte einige Zeit in Anspruch nehmen. Diese ist jedoch notwendig, um die Anwendung in Komponenten aufzuteilen. Zu weiteren Arbeiten hierzu siehe auch Krammer und Zaha (2003).

Larsson und Crnkovic (2000) befassen sich bei ihrer Definition des Begriffs des Komponentenkonfigurationsmanagements hauptsächlich mit der Analyse der Abhängigkeiten der Komponenten untereinander. Falls eine Komponente aus weiteren Komponenten zusammengesetzt ist, entstehen zwangsweise Abhängigkeiten zwischen diesen Komponenten, die es zu kontrollieren gilt. Einen besonderen Schwerpunkt in ihren Betrachtungen setzen die Autoren auf die Frage, wie verschiedene Versionen von ein und derselben Komponente gehandhabt werden müssen.

Sie identifizieren die Teilaufgaben des Versionsmanagements, des Konfigurations- oder Build-Managements und des Änderungsmanagements als die zentralen Aufgaben im Konfigurationsmanagement. Ersteres beschäftigt sich mit einzelnen Komponenten, aber jeweils in verschiedenen Versionen, das Build-Management befasst sich mit der Erstellung von Anwendungen durch Auswahl und Kombination von Komponenten während sich das Änderungsmanagement mit der Wartung und Pflege bestehender Anwendungen auseinander setzt. Dabei geht es hauptsächlich um die Frage, ob bestehende Komponenten durch aktualisierte oder bessere ersetzt werden können, ohne dass die Funktionalität der Anwendung gestört wird.

Wie bereits eingangs erwähnt wurde, liegt ein besonderer Schwerpunkt der Arbeit von Larsson und Crnkovic auf der Analyse der Abhängigkeiten von Komponenten untereinander. Hierbei untersuchen sie auch einen Aspekt, der vielfach unberücksichtigt bleibt, nämlich den, dass bei der Installation von Anwendungen durchaus auf dem Zielsystem bereits einige der Komponenten der neuen Anwendung installiert sein könnten. Dabei tritt in der Regel der Fall ein, dass die zu installierende Komponente in einer anderen Version vorliegt als die bereits im System verfügbare. Dass dies häufig zu Problemen führt, da verschiedene Komponentenversionen zueinander inkompatibel sein können, ist leicht einzusehen. Dies ist besonders wichtig, weil beispielsweise die weit verbreitete Komponententechnologie COM von Microsoft bisher kein Versionsmanagement unterstützte. In dieser Arbeit soll allerdings

davon ausgegangen werden, dass diese Probleme, die eher technischer Natur sind, gelöst sind. Etwa kann das Problem dadurch umgangen werden, dass es ermöglicht wird, verschiedene Versionen einer Komponente im System zu integrieren, wie es in der neuen Technik Microsoft.NET vorgesehen ist.

In dieser Arbeit soll die Definition von Becker und Overhage (2003) verwendet werden. Die Autoren fassen die Teilaufgaben des Auswahl- sowie des Änderungs- und Versionsmanagements zur Gesamtaufgabe des Konfigurationsmanagements zusammen.

- Auswahlmanagement: Hierbei geht es um die systematische Planung und Entscheidungsfindung bei der Frage, welche der verfügbaren Komponenten in der herzustellenden oder zu erweiternden Anwendung benutzt werden sollen. Dabei wird davon ausgegangen, dass ein Teil (eine Komponente) aus einem Teilelager auszuwählen ist. Das Teilelager enthält dabei mehrere Möglichkeiten für die Auswahl einer Komponente, die sich sinnvollerweise in wichtigen Attributen, beispielsweise Qualitätsaspekten, unterscheiden. Jedoch muss jedes gefundene Teil prinzipiell die an es gestellten Anforderungen erfüllen. Darüber hinaus darf die Auswahl der Gesamtmenge der einzelnen Komponenten evtl. existierende Budgetrestriktionen nicht verletzen. Eine Gegenüberstellung der Kosten und des zu erwartenden Nutzens der wichtigsten Alternativen sollte am Ende des Auswahlmanagements erfolgen.

- Änderungsmanagement/Versionsmanagement: Wie bereits weiter oben erwähnt, fallen Wartungsarbeiten an komponentenbasierten Anwendungen häufig dadurch an, dass neue Versionen bereits eingesetzter Komponenten verfügbar werden oder dass verwendete Komponenten durch geeignete andere Komponenten (evtl. auch eines anderen Anbieters) ersetzt werden sollen oder müssen. Hierbei gilt es zum einen, die von den geplanten Änderungen betroffenen Anwendungen effizient zu ermitteln, und zum anderen gilt es, die Komponenten auch tatsächlich auszutauschen. Dies führt in der Regel zu Anpassungen an den Adaptoren dieser Komponente und zu einer erneuten Durchführung der Testpläne der angepassten Anwendung. Diese Tests dienen dazu, nicht vorhersehbare Kompatibilitätsprobleme, die durch den Einsatz der neuen Komponente auftreten, herauszufinden. Das Änderungs- bzw. Versionsmanagement wird ihm Rahmen des Abschnitts 3.6 über Speicherstrukturen für die Ergebnisse aufgegriffen und behandelt.

Nachdem nun die zentralen Termini, die in dieser Arbeit verwendet werden, geklärt wurden, werden hierauf aufbauend im folgenden Kapitel die Elemente eines Prozesses zum Auswahlmanagement angegeben.

3 Bausteine eines Auswahlprozesses

In der Literatur finden sich bereits einige unterschiedliche Ansätze, um das Auswahlmanagement für Bestandteile von zu realisierenden Systemen zu unterstützen. Diese Verfahren besitzen vielfach einige Gemeinsamkeiten, so dass es sinnvoll erscheint, gemeinsame Elemente zu identifizieren und als Bausteine eines generischen Auswahlprozesses anzusehen.

Wird eine solche Einteilung vorgenommen, ist es einfacher möglich, die Gemeinsamkeiten und Unterschiede in den Prozessen der Vorarbeiten aufzuzeigen. So verwenden beispielsweise einige Autoren ein einfaches phasenorientiertes Vorgehensmodell während andere kompliziertere iterativ vorgehende Modelle vorschlagen. Trotzdem könnten beide Autoren ihre Entscheidungsfindung zum Beispiel auf den AHP aufbauen.

Durch die Einführung von Bausteinen soll erreicht werden, dass der Prozess flexibel bleibt und an den geplanten Einsatzzweck und an entsprechende Zeit- und Budgetvorgaben angepasst werden kann. Durch geeignetes Austauschen einzelner Elemente soll der Prozess unter diesen unterschiedlichen Rahmenbedingungen einsetzbar bleiben.

Dabei wird immer das zentrale Ziel verfolgt, den Gesamtprozess verständlich und nachvollziehbar zu halten, so dass die in der Realität häufig anzutreffende Situation unkontrollierter Vorgehensweisen ausgeschlossen wird. Das Vorgehen soll für Dritte nachvollziehbar bleiben, damit ein effizientes Teamwork auch dann gewährleistet werden kann, wenn einzelne Teammitglieder das Team verlassen oder neue Mitarbeiter aufgenommen werden sollen. Insbesondere zielt diese Arbeit darauf, die Auswahl von Komponenten bzw. Konfigurationen rational zu gestalten, da gerade dieser Teil der Entwicklung einer komponentenorientierten Anwendung in der Praxis häufig auf schlecht reflektierten ad hoc Entscheidungen basiert ist.

Es ist darüber hinaus noch anzumerken, dass die meisten der in diesem Kapitel zitierten Vorarbeiten sich zwar prinzipiell mit Softwarekomponenten auseinander setzen. Problematisch ist jedoch die grobe Granularität der Komponenten (COTS, s. Kapitel 1), mit der in der Regel argumentiert wird. So werden meist ganze Anwendungen als Komponenten angesehen. Dies lässt sich zwar mit der hier verwendeten Definition einer Softwarekomponente vereinbaren, allerdings soll hier eher in kleineren Einheiten einer Anwendung gedacht werden. So wäre beispielsweise die Buchhaltung ein Teil einer betrieblichen Anwendung und Teile davon wären wiederum die Bilanzierung und die Gewinn- und Verlustrechnung. Komponenten mit einer Größe, die diesen Aufgabenstellungen angemessen ist, sollen hier im Mittelpunkt der Betrachtungen stehen (vgl. auch Abschnitt 5.1).

In dieser Arbeit werden die folgenden Bausteine als wichtig für den zu bildenden Prozess angesehen:

- Es gibt zwei Arten von Szenarien, die unterschieden werden sollen: Zum einen das Einsatzszenario der entwickelten Anwendung und zum anderen das Einsatzszenario des Prozesses selbst.

- Eine Vorgehensweise, die jederzeit angibt, welche Teilaufgabe innerhalb des geplanten Entwicklungs- bzw. Auswahlprozesses zu erfüllen ist und welche Ergebnisse in den einzelnen Schritten erstellt werden müssen.

- Geeignete Spezifikationssprachen für Komponenten, in denen sowohl die benötigten Komponenten, als auch die während der Suche gefundenen Komponenten spezifiziert

werden können.

- Eine Wissensbasis (Knowledge Base), die sowohl die im Unternehmen verfügbaren Komponenten dokumentiert als auch die gesammelten Erkenntnisse früherer Durchführungen einer komponentenbasierten Anwendungsentwicklung enthält.
- Eine oder mehrere Sprachen, in denen die Ergebnisse der einzelnen Schritte der Vorgehensweise dokumentiert werden können. Die Sprachen sollen dabei natürlich die auszuführenden Teilaufgaben bestmöglich unterstützen.
- Ein Verfahren, mit dem es möglich ist, unter den ermittelten Alternativen eine im gegebenen Kontext möglichst optimale im Bezug auf die Präferenzen des Entscheiders auszuwählen.

Über die hier vorgeschlagenen Bausteine hinaus gibt es in der Literatur weitere, meist nicht einheitliche, Ansichten darüber, welche Anforderungen an Prozesse dieser Art gestellt werden sollen. Daher sei hier auch auf die Arbeiten von Ncube und Maiden (1999), Kunda und Brooks (1999) und Ruhe (2003) verwiesen.

Im Folgenden werden die vorgestellten Bausteine, die in Abbildung 2 noch einmal dargestellt sind, weiter beleuchtet.

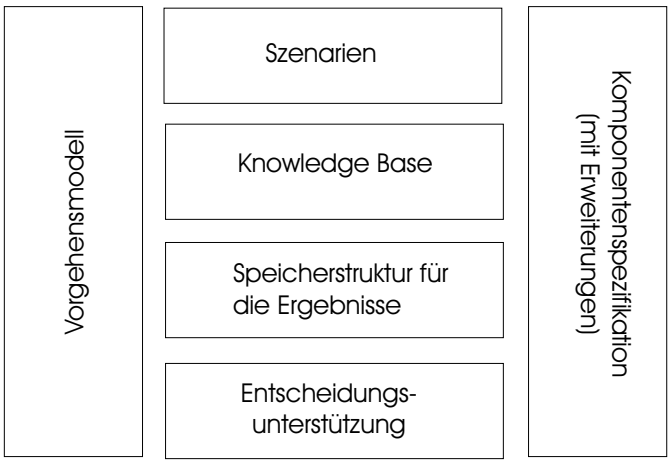

Abbildung 2: Bausteine des Konfigurationsmanagements

3.1 Szenarien

Zwei Arten von Szenarien können betrachtet werden. Zum einen sind Szenarien für die weitere Verwendung der entwickelten Anwendung bzw. die Zeit nach einer durchgeführten Wartung einer bereits existierenden Anwendung zu betrachten. Dabei steht die Frage nach eventuell vorhandenen Möglichkeiten, eine Anwendung in zusätzlichen oder ähnlichen Anwendungsfeldern einzusetzen, im Mittelpunkt. Da diese Szenarien direkten Einfluss auf die zu treffende Entscheidung besitzen können, werden sie in den Abschnitten 3.7.2 bzw. 8.2 behandelt.

In diesem Abschnitt sollen dagegen motivierend für die folgenden Bausteine einige Anwendungsszenarien für den hier vorgestellten Prozess beschrieben werden.

Ziel der hier vorgestellten Methode soll die Unterstützung der Entwickler bei der Entwicklung komponentenbasierter Anwendungen sein. Hierbei kann prinzipiell zwischen Komponentenentwicklern oder Herstellern von Komponenten und Verwendern von Komponenten oder Konfiguratoren unterschieden werden. Dass diese sich durchaus innerhalb des selben Unternehmens befinden oder sogar in Personalunion auftreten können, sollte dabei klar sein.

Bei der Herstellung neuer Komponenten gibt es zwei grundsätzliche Möglichkeiten. Zum einen kann der Fall eintreten, dass eine Elementarkomponente zu erstellen ist. Dabei soll das Wort Elementarkomponente so verstanden werden, dass sie eine Komponente ist, die nicht aus weiteren Komponenten zusammengesetzt wird. Für solche Komponenten fällt nur die Planung und Durchführung eines Entwicklungszyklus einer Entwicklung im Kleinen (s. Abschnitt 3.2) an.

Falls jedoch die zu erstellende Komponente aus weiteren Komponenten aufgebaut werden soll, so ist eine Entwicklung im Großen durchzuführen. Soll bei der entstehenden Komponente über deren inneren Aufbau gesprochen werden, wird bei diesen Komponenten in Anlehnung an die Materialwirtschaft häufig von Baugruppen gesprochen. Wird eine Baugruppe entwickelt, so ergibt sich ein Auswahlproblem (die Bestandteile der Baugruppe müssen ausgewählt werden), und die hier vorgeschlagenen Verfahren können eingesetzt werden.

In beiden Fällen ist zu beachten, dass die erstellten Komponenten und deren Dokumentationen (Spezifikationen, Testpläne, Baupläne, etc.) in die Wissensbasis (s. Abschnitt 3.5) aufgenommen werden müssen.

Aus Sicht der Anwender ergibt sich meist die Situation, dass eine Anwendung aus vorgefertigten Komponenten zusammengesetzt werden muss. Hierbei entstehen prinzipiell die gleichen Probleme wie bei der Herstellung einer Baugruppe. In der Regel ist sogar davon auszugehen, dass die Gesamtanwendung auch aus identifizierbaren Baugruppen besteht (z.B. aus den Baugruppen Buchhaltung, Personalwesen, ...). Sind die benötigten Teile nicht verfügbar, so müssen auch auf der Seite der Anwender erst entsprechende Baugruppen erstellt werden.

Orthogonal zu der Frage, ob Hersteller oder Anwender von Komponenten betrachtet werden, steht die Frage, ob eine Anwendung von Grund auf neu entwickelt wird oder ob Komponenten in einer bestehenden Anwendung ergänzt oder ausgetauscht werden müssen. Vom Fall der Neuentwicklung wurde in den vorangegangenen Absätzen ausgegangen. Bei einer durchzuführenden Wartung müssen meist nur wenige Komponenten gesucht und damit auch ausgewählt werden, da dabei nur einige wenige, spezielle Stellen innerhalb der Anwendung von der entsprechenden Wartungsaufgabe betroffen sind. Dagegen gewinnt in diesem Fall die Problematik der Suche einer Komponente an Brisanz, da es eventuell eine Reihe von Abhängigkeiten zwischen der von der Wartung betroffenen Komponente und dem Restsystem gibt. Eine genaue Analyse dieser Abhängigkeiten muss vor der Entscheidung für eine bestimmte Komponente durchgeführt werden, da die Abhängigkeiten den Entschei-

dungsprozess beeinflussen können. Daher führen diese Abhängigkeiten auch zu einer neuen Betrachtungsweise in der Suche, wie im Abschnitt 5.1 erläutert wird.

In Abbildung 3 sind die unterschiedlichen Einsatzmöglichkeiten noch einmal graphisch dargestellt.

	Erstellung	Wartung
Hersteller	Elementar-komponente Bauteil	Austausch, Erweiterung von Bauteilen
Konsument	Bauteil Anwendung	Austausch, Erweiterung von Anwendungen

Abbildung 3: Einsatzszenarien

3.2 Vorgehensweise

Es gibt zwei Arten von Vorgehensweisen, die für diese Arbeit relevant sind. Die eine Vorgehensweise steuert die Entwicklung oder Wartung der komponentenorientierten Anwendung an sich. Kernaufgabe dieser Vorgehensweise ist es, Teilaufgaben wie die Anforderungsanalyse, den Systementwurf, die Konfiguration oder das Testen zu koordinieren. Die zweite Vorgehensweise, die hier betrachtet werden soll, ist die, die bei der Suche nach Komponenten und bei der Generierung und Bewertung von Konfigurationen durchgeführt wird. Diese Aufgabe wird als eine Teilaufgabe der ersten Vorgehensweise durchzuführen sein, wodurch der Zusammenhang zwischen beiden gegeben ist.

Für Vorgehensweisen des ersten Typs gibt es in der Software Engineering Literatur einige weit verbreitete Ansätze, die vom einfachen Wasserfallmodell, über das V-Modell (siehe Dröschel und Wiemers (1999)) hin zum Unified Process (siehe Jacobson et al. (1999)) reichen. Diese Vorgehensweisen sind allerdings alle so generisch gehalten, dass sie in beliebigen Einsatzszenarien angewendet werden können. Spezielle Vorgehensweisen, die für die komponentenorientierte Anwendungsentwicklung geeignet sind, gibt es bislang wenige. Ein solches Vorgehensmodell mit den entsprechenden Dokumenten einzuhalten, ist jedoch eine der wichtigsten Voraussetzungen einer ISO 9001 Zertifizierung. Dieses Zertifikat bestätigt dem entsprechenden Unternehmen die Einhaltung bestimmter Qualitätsanforderungen, was für den Geschäftserfolg in der Praxis von essentieller Bedeutung sein kann.

Ein Vorgehensmodell, das für die komponentenorientierte Anwendungsentwicklung angepasst wurde, findet sich in Becker und Overhage (2003) (vgl. Abbildung 4). Dieses Modell basiert auf Vorarbeiten in Overhage (2002a) und Ortner (1998).

Neben den eher klassischen Phasen Voruntersuchung, Fachentwurf, Integration und Einführung, unterteilt dieses Modell in die in der komponentenorientierten Anwendungsentwicklung wichtigen Stufen der Entwicklung im Großen und der Entwicklung im Kleinen.

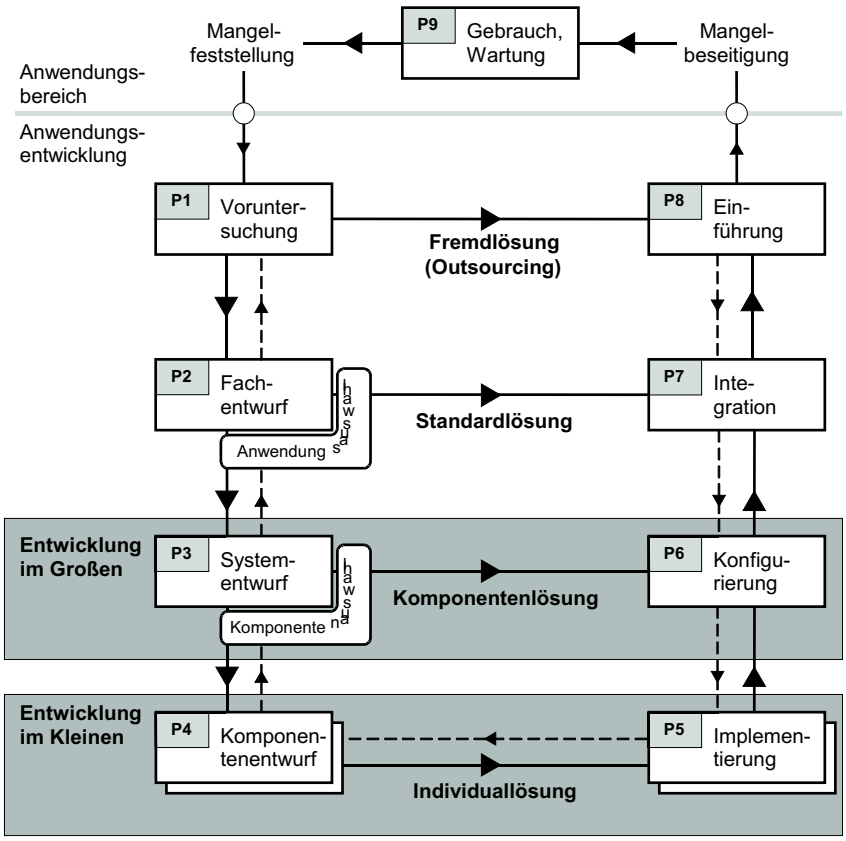

Abbildung 4: Vorgehensmodell (aus Becker und Overhage (2003))

In der Entwicklung im Großen findet die Konfiguration der Anwendung aus ihren Einzelteilen statt. Wie die Abbildung zeigt, ist eine der wichtigsten Aufgaben hier die Komponentenauswahl. In ihr entscheidet sich auch, ob eine benötigte Komponente eventuell zuerst im eigenen Unternehmen hergestellt werden muss. Dies ist dann der Fall, wenn keine geeignete Komponente während des Suchprozesses gefunden wird.

Tritt dieser Fall ein, so wird für die fehlende Komponente eine Iteration der Entwicklung im Kleinen durchgeführt, bei der die Komponente fachlich und technisch spezifiziert, implementiert und getestet wird. Dies kann z.B. wieder nach dem unmodifizierten Multipfadvorgehensmodell aus Ortner (1998) geschehen.

Für das Vorgehensmodell zur Auswahl der Komponenten gibt es einige Ansätze in den in der Literatur veröffentlichten Modellen. Je nach Schwerpunkt der verschiedenen Arbeiten sind die Vorgehensweisen mehr oder weniger detailliert beschrieben. Eine gute und aktuelle Übersicht über die wenigen Arbeiten in diesem Bereich gibt Ruhe (2003). Im Abschnitt 3.8 werden einige dieser Arbeiten kurz vorgestellt.

Auf zwei interessante Arbeiten soll dabei hier näher eingegangen werden. Kommt es zu der Frage, wie Komponenten in einem zu erstellenden System ausgewählt werden sollen, so werden in der Literatur sehr häufig die Arbeiten von Kontio (1995) erwähnt, die durch zwei Fallstudien (s. Kontio (1996) und Kontio et al. (1996)) erprobt worden sind. Die andere Arbeit beschäftigt sich primär mit einem Prozess zur Bewertung von Komponenten nach

Qualitätsmerkmalen.

In der Arbeit von Kontio wird in sehr ausführlicher Art und Weise angegeben, wie eine COTS Auswahl nach dem präsentierten Prozess, der OTSO (Off-The-Shelf Option) genannt wird, aussehen soll. Dabei gibt der Autor sowohl eine Vorgehensweise an, die er in das Informationsmanagement der Unternehmung einordnet, als auch einen sehr ausführlichen Katalog an Qualitätskriterien, die in einer der Fallstudien verwendet wurden.

In OTSO wird eine Vorgehensweise verwendet, die davon ausgeht, dass Anforderungen aus einer bereits vorgenommenen Anforderungsanalyse existieren, die nach Abschluss dieser Analyse als unveränderlich angesehen werden. Auf Basis dieser Anforderungen an das herzustellende System wird in OTSO zuerst eine Menge von Kriterien zur Evaluation der in Frage kommenden Alternativen erstellt. Die Ergebnisse dieses Schritts sind Such- und Vorauswahlkriterien sowie genaue Beschreibungen der zu evaluierenden Merkmale der später genauer zu untersuchenden Komponenten. An diesen Schritt schließt sich die Suche nach geeigneten Kandidaten für die benötigten Komponenten an. Diese Suche soll dabei noch ein relativ breites Spektrum an Alternativen liefern. Diese werden dann in einem nächsten Schritt vorausgewählt, so dass nur die vielversprechendsten Kandidaten in der Menge der Suchergebnisse übrig bleiben. Die Menge muss dabei soweit dezimiert werden, dass eine genaue Untersuchung der Restkomponenten nicht den finanziellen Rahmen sprengt. Dies geschieht anhand der im ersten Schritt ermittelten Vorauswahlkriterien.

Die verbliebenen Alternativen werden anschließend anhand der definierten Bewertungskriterien genau untersucht. Die Untersuchungsergebnisse werden für jede Alternative gemäß der Bestimmungen des ersten Schritts dokumentiert. Sind alle Alternativen auf diese Art und Weise bewertet worden, folgt im letzten Schritt die so genannte Analyse der Ergebnisse. Hierbei wird im OTSO Prozess der AHP zur Ermittlung der Präferenzen der Entscheider und zu deren Aggregation eingesetzt. Diesen Ergebnissen sollen abgeschätzte Kosten bei der Auswahl einzelner Alternativen gegenübergestellt werden, die nach einem speziellen Schema (s. Kontio (1995, S. 18)) ermittelt werden können. Auf Basis dieser Kosten-Nutzen-Informationen sollen die Entscheidungsträger die geeignete Komponente auswählen.

Die Arbeit von Hansen (2001) beschäftigt sich dagegen mit der Frage, wie der Prozess der Bewertung, also hauptsächlich der vorletzte Schritt des OTSO Verfahrens, im Bezug auf Qualitätsaspekte durchzuführen ist. Besonders interessant hieran ist, dass es eine ISO Norm - die Norm ISO 9126 - gibt, die einen Prozess für ein solches Vorhaben beschreibt. Die Bewertung findet allerdings anhand von Metriken statt, so dass sich das entscheidende Problem darin stellt, für qualitative Merkmale geeignete Metriken finden zu müssen.

Die ISO Norm 9126 gliedert den Prozess grob in drei Phasen: Anforderungsanalyse, Vorbereitungsphase und Bewertungsphase. In der ersten Phase werden die Qualitätsanforderungen an die zu erstellende Software gesammelt. In der zweiten Phase werden geeignete Metriken ausgesucht, Bewertungsverfahren spezifiziert und Bewertungsmerkmale festgelegt. In der dritten Phase werden dann die Alternativen untersucht („vermessen“) und die Messergebnisse werden anhand der Bewertungsmerkmale und -verfahren bewertet.

Angelehnt an diesen Prozess beschreibt die Arbeit von Hansen einen Prozess, der QUESTA genannt wird. Dieser Prozess soll an dieser Stelle jedoch nicht wiedergegeben werden, da er sich nicht in wesentlichen Punkten von demjenigen der ISO Norm 9126 unterscheidet.

Die weiteren der in Ruhe (2003) angegebenen Arbeiten benutzen meist Variationen eines oder mehrerer der in diesem Abschnitt angesprochenen grundlegenden Vorgehensweisen. Die in der Arbeit von Ruhe angegebenen Prozesse könnten daher mittels der hier vorgestellten Bausteine kategorisiert werden.

3.3 Komponentenspezifikation

Wie der Titel eines Artikels von Overhage (2002a) schon aussagt, ist die Spezifikation der Komponenten ein kritischer Erfolgsfaktor für den Durchbruch der komponentenorientierten Anwendungsentwicklung insgesamt. Da die Grundidee dieses Paradigmas darauf beruht, einmal erstellte Komponenten in späteren Projekten wiederzuverwenden bzw. entsprechende Komponenten von externen Quellen zu beziehen, ist es besonders wichtig, eine umfassende und möglichst standardisierte Komponentenspezifikation zu besitzen. Die Spezifikation bildet etwas ähnliches wie eine vertragliche Grundlage zwischen dem Komponentenanbieter und dem Nachfrager der Komponente. Der Anbieter sichert mit ihr gewisse Eigenschaften der Komponente zu, auf die sich der Nachfrager verlassen können muss. Diese sollten so beschaffen sein, dass sie durch den Konsumenten der Komponente nachprüfbar sind. In der Realität ist dies bisher jedoch aufgrund der hohen Komplexität von Komponenten bzw. deren Spezifikationen nicht für alle Attribute möglich.

Dass es enorme Probleme gibt, eine geeignete Komponentenspezifikation aufzustellen, zeigt sich an den Eigenschaften, die eine solche Spezifikation idealerweise haben müsste. Sie sollte die Komponente vollständig und nachprüfbar beschreiben, soll dabei aber auch nicht zu umfangreich sein, da sonst kein Hersteller seine Komponenten mittels dieser Spezifikation beschreibt. Für die Hersteller stellt sich sowieso die Frage, inwiefern sie bereit sind, ihre Komponenten technisch und fachlich exakt zu spezifizieren. Viele der im Marketing benutzten Methoden beruhen gerade darauf, beim Konsument psychologische Vorgänge auszulösen, die in der Regel nicht auf Fakten beruhen. Und selbst wenn dieser Faktor außer Acht gelassen wird, ist es fraglich, ob die Hersteller in den für den Konsumenten verbesserten Möglichkeiten zum Vergleich von Komponenten nicht ebenfalls mehr Risiken als Chancen sehen werden.

Werden all diese Probleme außer Acht gelassen, ergeben sich noch weitere Anforderungen an Spezifikationen. Sie sollten sowohl für Maschinen als auch für Menschen lesbar sein. Die Maschinenlesbarkeit wird dabei für Werkzeuge benötigt, die die an der Arbeit mit Komponenten beteiligten Personen unterstützen. Ein Werkzeug bzw. eine Sammlung von Werkzeugen, die die hier angegebene Methode unterstützen, wäre ein Beispiel hierzu.

So hat die eingangs zitierte Arbeit von Overhage auch die Forderung nach einem standardisierten Spezifikationsrahmen für Komponenten aufgestellt und die Konsequenzen der Existenz eines solchen Rahmens projiziert. Neben dem bereits in Abschnitt 3.2 angesprochenen Vorgehensmodell aus der Arbeit von Overhage ergeben sich Möglichkeiten für weitere Werkzeuge. Hierunter fallen auch einige der später noch anzusprechenden Elemente dieses Prozesses wie ein Komponentenrepository und ein entsprechender Komponentenmarktplatz. Auch die Frage, wie die für die Auswahl von Komponenten essentielle Suche auf eine einheitliche Basis gestellt werden kann, könnte angegangen werden. Erste Denkansätze in diese Richtung sollen in Kapitel 5 dargestellt werden.

Diese Beispiele zeigen die zentrale Bedeutung der Komponentenspezifikation sowie ihrer Exaktheit auch für diese Arbeit. Daher wird an einigen Stellen in dieser Arbeit auf die

Kernfrage nach der Spezifikation der Komponenten zurückzukommen sein. Damit jedoch eine einheitliche Grundlage für diese Diskussionen besteht, sollen hier einige bekanntere Spezifikationssprachen vorgestellt und kritisch beurteilt werden.

3.3.1 Eigenschaften

Bevor jedoch auf die Beispiele zu Komponentenspezifikationen eingegangen wird, werden einige Eigenschaften vorgestellt, die Spezifikationen erfüllen können. Anhand der erfüllten Eigenschaften ergibt sich ein besseres Bild über die Leistungsfähigkeit einer Spezifikation. Da jedoch - bezogen auf den Speicherverbrauch und die Bearbeitungsgeschwindigkeit - effektive Mechanismen zum Wiederauffinden einer spezifizierten Komponente umso leichter zu implementieren sind, je einfacher die Spezifikation gehalten ist, muss bei der Beurteilung einer Spezifikation auch dieser Aspekt berücksichtigt werden. Das effiziente Suchverhalten beim Auffinden auf der einen Seite und die vollständige und exakte Spezifikation einer Komponente auf der anderen stellen in diesem Sinne konkurrierende Ziele dar.

Daher ist die wichtigste Eigenschaft einer Spezifikation der *Umfang der spezifizierten Daten* über eine Komponente. Hier soll auf den drei Ebenen Syntax, Semantik und Pragmatik unterschieden werden[1]. Der Umfang einer Spezifikation soll nur die Syntax abdecken, wenn die Schnittstellen und die einzelnen Signaturen spezifiziert werden. Er sei darüber hinaus Semantisch, wenn die verwendeten Begriffe (z.B. Variablentypen, Methodennamen, ...) mit einer Bedeutung in Form von Begriffsdefinitionen (Lexika, Ontologien, ...) versehen sind. Pragmatisch ist sie, wenn zusätzlich spezifiziert wird, was die Dienste der Komponente *tun*, z.B. in Form der umgesetzten Geschäftsprozesse.

Orthogonal zu dieser Einteilung können Spezifikationen danach klassifiziert werden, welchen *Ausschnitt der Komponente* sie beschreiben. Ortner (1997) unterteilt hierbei in fachliche, logische und physische Beschreibungen. Fachliche Angaben beziehen sich auf das Anwendungsgebiet und modellieren die (betriebliche) Realität. Angaben zur logischen Arbeitsweise sind Informationen über die internen Abläufe der Komponente aus einer konzeptionellen Sicht heraus. Auf der physischen Ebene werden die Abläufe auf einer rechnernahen Art und Weise charakterisiert.

In Anlehnung an Frakes und Pole (1994) kann die Art und Weise, wie die Spezifikation *klassifiziert* wird, als weitere Eigenschaft aufgefasst werden. Dabei sind Attribut-Wert-Kombinationen (Key-Value-Pairs), die Erfassung von Schlüsselwörtern, die beliebig wählbar oder aus einem kontrollierten Vokabular stammen können, die Einteilung anhand eines Klassifikationsbaums oder die Verwendung von Facetten (s. Prieto-Diaz (1991)) zu betrachten.

Die Art und Weise, in der die Spezifikation *dargestellt* wird, kann als weitere Eigenschaft angegeben werden. Es wird z.B. zwischen Klartextspezifikationen (in natürlichen Sprachen), formalen bzw. halbformalen Spezifikationen oder - die für den Austausch von Spezifikationen zwischen verschiedenen Systemen wichtige - Spezifikation in XML unterschieden. Eine XML Spezifikation legt zwar die Syntax und beim Einsatz von XSchema auch die erlaubten Datentypen fest, die Semantik der Sprachen unterliegt jedoch der Interpretation des Empfängers eines solchen Dokuments.

[1] In der Informatik wird diese Unterteilung auch in Form von Syntax, statischer Semantik und dynamischer Semantik vorgenommen.

Erweiterbar soll eine Spezifikation genannt werden, wenn durch den Anwender weitere Attribute aufgenommen werden können. Die Attribute *Lesbarkeit* durch Menschen und Maschinen/Programme sind selbsterklärend.

3.3.2 CDM

Als ein erstes Beispiel einer Beschreibungssprache soll hier CDM vorgestellt werden. CDM speichert Informationen über Komponenten in einer ganz einfachen Form als besonders strukturierte Attribut-Wert-Paare.

Die Abkürzung CDM steht für Component Description Manager, ein von Meling et al. (2000) entwickeltes Rahmenwerkzeug zur Beschreibung von Software Komponenten. Näher betrachtet und analysiert wurde dieses Werkzeug in einer Seminararbeit an der Technischen Universität Darmstadt (s. Eschinger et al. (2001)).

Die Spezifikation der Komponenten erfolgt in diesem Schema mittels Klassifikationsbäumen. Dabei werden die zu spezifizierenden Attribute hierarchisch aufgebaut. An den Enden der Hierarchie, dass heißt in den Blättern des aufgespannten Baums befinden sich die Informationen in Form von Attribut-Wert-Paaren.

Prinzipiell ist der Klassifikationsbaum erweiterbar, da das Aufnehmen weiterer Attribute vorgesehen ist. Die erfassten Attribut-Wert-Paare enthalten als Attribute den Pfad entlang der Hierarchie bis zum spezifizierten Wert. Eine solche Angabe sieht beispielsweise wie folgt aus:

```
characteristics.required.name = "Buchhaltungskomponente für 5 Benutzer"
```

Vorgegeben in der CDM Spezifikation ist die erste Ebene des Klassifikationsbaums. Sie besteht aus den vier Teilbäumen „characteristics“, „grammar“, „components“ und „standards“. In Anlehnung an Eschinger et al. (2001) soll kurz erläutert werden, was innerhalb dieser einzelnen Teilbäume zu spezifizieren ist.

Auf der „characteristics“-Ebene wird das „Vokabular“ für die Komponentenbeschreibung gespeichert. Es handelt sich um die eigentlichen Attribute, die die Komponente charakterisieren. Auf der „grammar“-Ebene werden die Vokabeln der „characteristics“-Ebene in Beziehung gesetzt. So werden beispielsweise abstraktive oder kompositiven Beziehungen festgehalten. Auf der „components“-Ebene wird die Komponente mittels einer eindeutigen Klassifikation, die sich an den Namespace-Konventionen der Programmiersprache Java orientiert, eindeutig identifiziert. In der „standards“-Ebene werden zu guter Letzt Standards gespeichert, die bei der Erstellung oder dem Einsatz der Komponente wichtig sind, wie beispielsweise andere (Standard-)Komponenten oder unterstützte Datenformate.

Die Spezifikation gemäß CDM speichert die Semantik einer Komponente, ist erweiterbar und weitgehend menschenlesbar. Die Lesbarkeit durch Maschinen ist nur eingeschränkt gegeben. Zwar können die Attribut-Wert-Paare effizient verarbeitet werden, jedoch sind die Werteinträge untypisiert. Hier würde sich der Einsatz von XML in Kombination mit XSchema lohnen. Ob dies allerdings in der Zwischenzeit nicht bereits eingesetzt wird, ist nicht bekannt.

Eschinger et al. (2001) haben weiterhin angedacht, die Felder des Spezifikationsrahmens nach Ackermann et al. (s. Abschnitt 3.3.4) auf die Elemente der Beschreibung gemäß CDM zu übertragen.

3.3.3 KDL

Als zweites Beispiel soll hier die KDL (Kind Description Language), die in Kiniry (1999) dargestellt wird, beschrieben werden. Sie steht beispielhaft für eine Klasse von Sprachen, die noch ausreichend formal sind, um mit ihrer Hilfe Aussagen über das Gesamtsystem durch Werkzeuge herleiten zu können, und gleichzeitig den Versuch unternehmen, die Semantik zu berücksichtigen. Andere Ansätze in dieser Richtung benutzen beispielsweise Petri-Netze oder das Pi-Kalkül, um Komponenten so zu beschreiben, dass aus diesen Angaben Vorhersagen über das Gesamtsystem getroffen werden können.

Außerdem ist diese Sprache ein Beispiel für eine Gruppe von Sprachen, die existierende andere Sprachen modifizieren und auf den Einsatzfall der Spezifikation von Komponenten anpassen. Besonders die im Software Engineering häufig eingesetzten Sprachen aus der UML oder die von der OMG entwickelten Sprachen werden hierbei in angepasster Form wiederverwendet.

KDL verwendet eine erweiterte Form der OCL, der Object Constraint Language (siehe OMG (2001b)), um die Komponenten zu beschreiben. Die OCL ist in unmodifizierter Form in der Lage, die Vor- und Nachbedingungen der einzelnen Dienste sowie die Invarianten einer Komponente zu beschreiben.

In KDL wurden hierzu zusätzlich temporale Operatoren eingeführt, mit denen es möglich ist, Abläufe und Reihenfolgen zu spezifizieren. Weiterhin wurden eine Reihe von Konstrukten definiert, mit denen es möglich ist, Relationen zwischen Komponenten zu spezifizieren. Abstraktive und kompositive Beziehungen können damit ebenfalls angegeben werden. Dabei sind abstraktive Beziehungen diejenigen, bei denen konkretere Komponenten zu abstrakteren Komponenten zugeordnet werden, z.B. Vertriebsleiter zu Personal oder Autos zu Fahrzeugen. Kompositive Beziehungen beschreiben, aus welchen anderen Komponenten eine Komponente zusammengesetzt ist, wie z.B. die Aussage, dass ein Auto vier Räder besitzt. Des Weiteren wurden zusätzlich Konstrukte aufgenommen, die es ermöglichen, Termini in Form von Ontologien in die Spezifikation aufzunehmen.

Kiniry (1999) gibt darüber hinaus weitere Beispiele für Spezifikationssprachen, die hier nicht betrachtet wurden.

3.3.4 Spezifikation nach Ackermann et al.

Der Spezifikationsrahmen nach Ackermann et al. (2002) soll als Beispiel einer Beschreibungssprache dienen, deren oberstes Ziel die *vollständige* Beschreibung der Komponente ist. Diese Kategorie von Sprachen beschreibt die Komponenten zwar besonders gut, allerdings ist die Verarbeitung durch Werkzeuge sowie das Vorhersagen des Verhaltens eines aus Komponenten zusammengesetzten Gesamtsystems bei solchen Spezifikationssprachen besonders schwierig.

Im Arbeitskreis 5.10.3 der Gesellschaft für Informatik (GI), der sich mit der „komponentenorientierten Gestaltung unternehmensindividueller oder branchenspezifischer betrieblicher Anwendungssysteme“ beschäftigt, wurde dazu ein bereits im Kapitel 2 angesprochenes Memorandum zur Vereinheitlichung der Spezifikation von Fachkomponenten entwickelt (siehe Ackermann et al. (2002)). Um eine breite Zustimmung für dieses Memorandum zu erreichen, wurde das Memorandum von verschiedensten Vertretern aus der Industrie und der Wissenschaft getragen.

Der in diesem Memorandum publizierte Spezifikationsrahmen für (Fach-)Komponenten soll hier näher erläutert werden. Innerhalb dieses Spezifikationsrahmens wird die Beschreibung einer Komponente anhand von so genannten Beschreibungsebenen postuliert. Auf diesen einzelnen Ebenen kommen dann für die jeweilige Ebene speziell angepasste Notationen zum Einsatz. In Abbildung 5 sind die sieben Ebenen, die der Spezifikationsrahmen in seiner aktuellen Fassung vorsieht, zu erkennen.

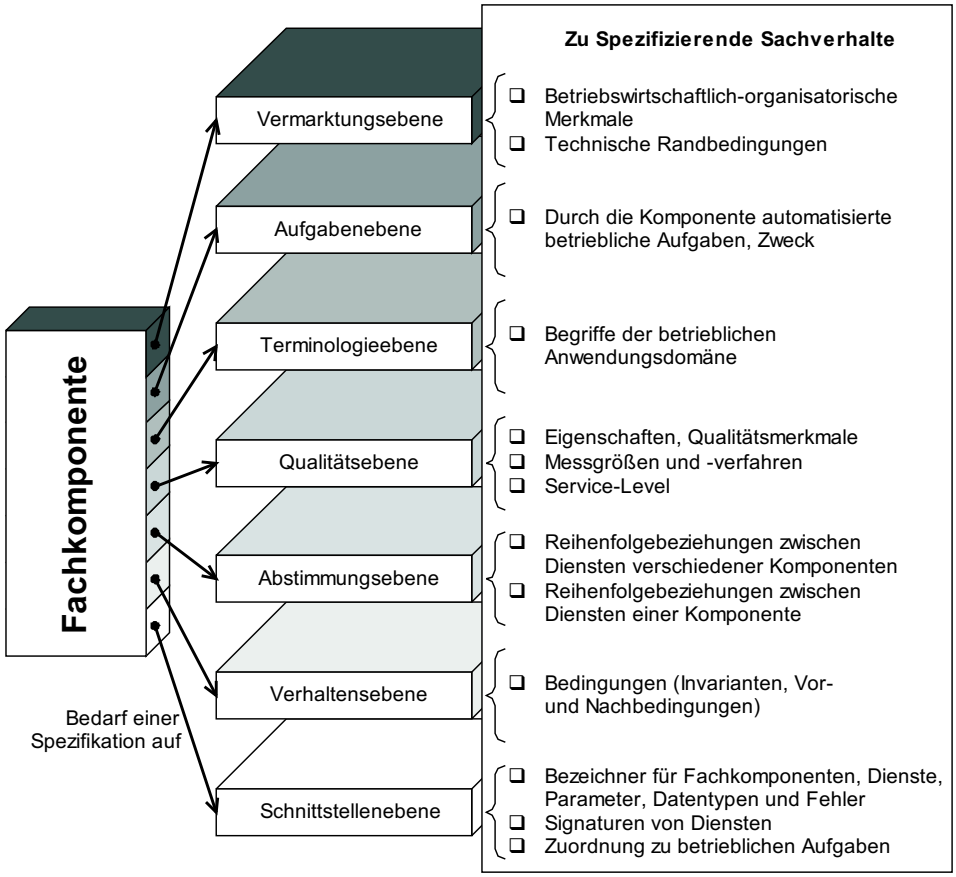

Abbildung 5: Spezifikationsrahmen für Komponenten (aus Ackermann et al. (2002))

Im Folgenden sollen die einzelnen Ebenen kurz vorgestellt und die zu spezifizierenden Attribute und ihre primäre Notationssprache beispielhaft angegeben werden.

Auf der Schnittstellenebene, die hier zuerst betrachtet werden soll, erfolgt unter anderem die Spezifikation der öffentlich angebotenen Dienste sowie deren Signaturen, der öffentlichen Attribute und Parameter, der bei Aufrufen möglichen Fehlersituationen sowie der exportierte Datentypen. Dabei wird als mögliche Notation für diese Ebene die von der OMG (2001a) entwickelte IDL (Interface Definiton Language) vorgeschlagen, die entwickelt wurde, um die plattformunabhängige Spezifikation von Schnittstellen zu unterstützen. Darüber hinaus werden in dieser Ebene die Verbindungen zwischen den definierten Wörtern aus der Terminologieebene, den Schnittstellendiensten und den durch diese realisierten Aufgaben gezogen bzw. zwischen den Wörtern und den korrespondierenden Datentypen.

Auf der Verhaltensebene wird das Verhalten einer Komponente beschrieben, indem Invarianten der Komponente bzw. Vor- und Nachbedingungen einzelner Dienste spezifiziert

werden. Beispielsweise hätte eine Buchhaltungskomponente die Invariante, dass die Summe aller Soll- und aller Habenpositionen stets identisch sein müssen. Zur Beschreibung dieser Attribute schlägt der Spezifikationsrahmen vor, eine modifizierte Variante der OCL zu verwenden. Damit wird ein ähnlicher Ansatz wie bei der KDL verfolgt, abgesehen davon, dass KDL seine gesamte Spezifikation auf der OCL aufbaut, während hier nur die Verhaltens- und die Abstimmungsebene betroffen sind. Auf der Verhaltensebene wird außerdem noch spezifiziert, welche Dienste von anderen (dritten) Komponenten benötigt werden und in welcher Art und Weise (Reihenfolge, Vorbedingungen, ...) diese aufrufbar sein müssen.

Die Abstimmungsebene spezifiziert die Abhängigkeiten beim Aufruf der einzelnen Dienste untereinander. So muss z.B. zuerst etwas auf ein Bankkonto gebucht werden, bevor dieses wieder belastet werden kann. Außerdem können Synchronisationsanforderungen zwischen den Dienstaufrufen angegeben werden, z.B. könnte es verboten sein, zwei Buchungen zeitgleich durchzuführen. Als Sprache, in der diese Restriktionen darzustellen sind, wird wieder die Variante der OCL (s. Verhaltensebene) vorgeschlagen. Dabei wird die OCL um Konstrukte erweitert, in denen temporale Abhängigkeiten dargestellt werden können.

Auf der Qualitätsebene sollen gemäß der Idee des Spezifikationsrahmens qualitätsbezogene Eigenschaften einer Komponente abgelegt werden. In der Literatur wird diese Eigenschaft den nicht-funktionalen Kriterien einer Software zugeordnet. Da es einige Kritik, auch innerhalb des Arbeitskreises, an der aktuellen Fassung dieser Ebene gibt, wurde ihr ein eigener Abschnitt (s. Abschnitt 3.4.3) gewidmet, so dass hier nicht weiter darauf eingegangen wird.

Die Terminologieebene dient dazu, die von der Komponente verwendeten Bezeichner eindeutig zu klären. Beispielsweise könnte eine Komponente unter dem Begriff „Preis“ den Bruttopreis verstehen, während eine andere den Nettopreis meint. Durch dieses Vorgehen enthält jede Komponentenspezifikation die Definitionen der in ihr verwendeten Wörter. Als Spezifikationsmethode für die Terminologieebene schlägt der Rahmen die Verwendung einer kontrollierten und rekonstruierten Normsprache vor (vgl. Ortner (1997)).

Die (betrieblichen) Aufgaben, die eine (Fach-)Komponente übernehmen kann, werden auf der Aufgabenebene spezifiziert. Dabei werden normierte Aussagen über die in den Diensten realisierten fachlichen Aufgaben angegeben. Das entsprechende Vorgehen ist in Ortner (1997) beschrieben. Aufgaben können weiterhin auch noch in Unteraufgaben unterteilt sein. Durch die Verbindung von Diensten an der Schnittstelle der Komponente zu den auf der Aufgabenebene erfassten Aussagen soll die vollständige fachliche Abgeschlossenheit der Spezifikation der Komponente gesichert werden.

Auf der letzten verbleibenden Ebene, der Vermarktungsebene, werden administrative Aspekte der Komponente in Form von Attribut-Wert-Paaren erfasst. Die Attribute enthalten unter anderem Angaben über den Hersteller, die Anwendungsdomäne, den Preis, die benötigte Basistechnologie und die Systemanforderungen.

Der vorgestellte Spezifikationsrahmen bietet einige Vorteile, besitzt aber bisher auch noch Nachteile. Vorteilhaft ist sicher die große Beteiligung der unterschiedlichen Organisationen an diesem Spezifikationsrahmen, der eine gewisse - zumindest nationale - Akzeptanz dokumentiert. Außerdem ist der Ansatz, Komponenten möglichst vollständig zu spezifizieren, zu begrüßen. Bisherige Spezifikationsstandards legen primär Wert auf technische Faktoren wie z.B. die Schnittstelle. Mit diesen Informationen alleine ist jedoch ein effekti-

ves Auswahlmanagement nahezu unmöglich, da höchstens ermittelt werden kann, ob eine bestimmte Komponente auf rein syntaktische Art und Weise kompatibel zu einer gegebenen Schnittstelle ist. Der Spezifikationsrahmen spezifiziert dagegen sowohl syntaktische als auch semantische und pragmatische Aspekte, die bei der Suche benötigt werden.

Erweiterbar ist der Spezifikationsrahmen durch das Konzept der Aufteilung der Spezifikationsaufgabe auf verschiedene Ebenen auch relativ einfach. Hierzu müssen nur weitere Ebenen eingefügt werden. Eine solche Erweiterung sollte allerdings nur durch den Arbeitskreis vorgenommen werden, da ansonsten eine unüberschaubare Menge an verschiedenen Spezifikationen entstehen würde. Damit wäre der Vorteil, den die Standardisierung mit sich bringt, wieder verloren.

Allerdings gibt es auch noch zu lösende Probleme im Zusammenhang mit diesem Spezifikationsrahmen. Beispielsweise sind die Beschreibungssprachen auf den einzelnen Ebenen nicht standardisiert. Dies führt dazu, dass zwei Beschreibungen, die sich zwar an die Ebenenarchitektur halten, mitunter trotzdem nur schwer verglichen werden können. Aus diesem Grund gibt es bisher auch keine maschinenlesbare Form des Spezifikationsrahmens, z.B. in Form eines XML Dokuments. Erste Ansätze hierzu sind jedoch bereits im Mittelpunkt des Interesses (s. Overhage (2002a)).

Da jedoch die Beschreibung der Komponenten mittels dieser Spezifikationsmethode am vielversprechendsten erscheint, soll in dieser Arbeit hauptsächlich an Komponentenspezifikationen gemäß dieses Rahmens gedacht werden. Weitere Annahmen, die zusätzlich in dieser Arbeit auch im Bezug auf diesen Spezifikationsrahmen getroffen werden, finden sich in Abschnitt 4.

3.4 Erweiterungen der Spezifikation nach Ackerman et al.

Auf dem diesjährigen 5. Workshop für komponentenorientierte Anwendungsentwicklung (WKBA 5), in dem der Spezifikationsrahmen des vorangegangenen Kapitels diskutiert wurde, setzte sich die Erkenntnis durch, dass es noch einige fehlende Spezifikationsebenen oder zumindest -attribute gibt. Unter diesen sind die beiden folgenden besonders wichtig. Eine Spezifikation der Parameter einer Komponente fehlt momentan genauso wie die Möglichkeit, Komponenten, die eine graphische Bedienoberfläche bieten, spezifizieren zu können. Ein weiterer Schwachpunkt in der aktuellen Version des Spezifikationsrahmens sind die Angaben zur Qualitätsebene.

Da die Absicht dieser Arbeit ist, einen entscheidungstheoretisch unterstützten Auswahlprozess zu konstruieren, sind diese Angaben aber von hoher Wichtigkeit. Parametrisierung kann entscheidenden Einfluss auf die Frage haben, ob eine Komponente in einem gegebenen Einsatzszenario verwendet werden kann oder nicht. Die graphische Benutzeroberfläche einer Komponente kann der entscheidende Faktor sein, eine Komponente einer anderen vorzuziehen. Die Qualitätssicherungsmaßnahmen sowie Faktoren wie die Performanz oder das Antwortverhalten sind wichtige Kriterien bei der Auswahl einer Komponente aus einer Menge von Alternativen.

Damit diese Arbeit nicht darunter leiden muss, dass diese Gebiete noch nicht vollständig erforscht sind, werden hier einige mögliche Erweiterungen des Spezifikationsrahmens angedacht, die die Möglichkeit bieten sollen, den Auswahlprozess auch zum gegenwärtigen Zeitpunkt auf die Basis des Spezifikationsrahmens von Ackermann et al. zu stellen.

Dabei soll hier keinesfalls aktuellen Forschungen oder Arbeiten auf diesen Gebieten vorgegriffen werden. Sollten entsprechende Forschungsergebnisse zu den einzelnen Schwachstellen verfügbar werden, wird es ohne Probleme möglich sein, diese in die vorliegende Arbeit zu integrieren.

3.4.1 Parametrisierung

Zur Berücksichtigung von Parametrisierungsmöglichkeiten von Komponenten bei deren Spezifikation gibt es bislang in der Literatur nur wenige Ansätze. In der Arbeit von Ackermann (2002) wird eine Übersicht über die auf diesem Gebiet bereits vorliegenden Arbeiten gegeben. Darüber hinaus beschäftigt sich die Arbeit mit der Frage, wie Parametrisierungsspielräume in die Komponentenspezifikation aus Abschnitt 3.3.4 zu integrieren sein könnten. Diese Arbeit zur Parametrisierung wurde in Ackermann (2003) fortgeführt.

An dieser Stelle soll jedoch zunächst erläutert werden, wie Parametrisierung bisher verstanden wird (aus Ackermann (2002)):

„Parametrisierung heißt eine Anpassungsstrategie, die sich durch folgende Merkmale auszeichnet:

- Die Anpassungsmöglichkeiten werden vom Hersteller der Software vorgedacht.
- Der Hersteller definiert Parameter inklusive deren Bedeutung und deren Auswirkungen auf die Funktionsweise der Software.
- Der Verwender prägt die Parameter aus, indem er diese mit den Werten belegt, die den von ihm gewünschten Verhalten entsprechen.
- Die Parameterbelegungen sind persistent und werden zur Laufzeit ausgewertet."

Ackermann (2002) berücksichtigt dabei weiterhin die Möglichkeit, dass die Parametrisierung der Komponente zu bestimmten Zeitpunkten während der Laufzeit geändert werden kann. Da dieser Fall im Rahmen dieser Arbeit schwer zu berücksichtigen ist, wird er durch die folgende Transformation beseitigt. Sollte eine Komponente existieren, deren Parameter sich zur Laufzeit ändern können, so wird dies als zwei verschiedene Komponenten angesehen, die gleichzeitig im System arbeiten. Vom Zeitpunkt der Änderung der Parametrisierung wird anstelle der ersten Komponente dann die zweite aufgerufen.

Damit etwas klarer wird, wie das Verständnis der Parametrisierung in dieser Arbeit ist, soll sie hier etwas formaler gefasst werden. Gegeben sei eine parametrisierbare Komponente bzw. deren Spezifikation C. Die Komponente besitze n mögliche Parameter, um ihr Verhalten zu steuern. Jeder dieser n Parameter stammt aus einer Menge von möglichen Parameterwerten P_i.

Es sei eine Abbildung definiert, die die parametrisierbare Komponente mit einem Vektor von möglichen Ausprägungen der einzelnen Parameter ausfüllt. Hieraus ergibt sich nach Einsetzen der Parameterwerte eine nicht-parametrisierte Komponentenbeschreibung $C_{\vec{p}}$, wie sie bisher in dieser Arbeit betrachtet wurde.

$$\mathrm{par}(C, \vec{p}) \longrightarrow C_{\vec{p}}$$

$$\vec{p} \in P_1 \times P_2 \times \ldots \times P_n$$

Damit ergibt sich die Menge parall(C) der durch die parametrisierte Komponente C beschriebenen Einzelkomponenten als

$$\text{parall}(C) = \{\text{par}(C, \vec{p}) \mid \forall \vec{p} \in P_1 \times P_2 \times \ldots \times P_n\}$$

Die Mächtigkeit dieser Menge ist

$$|parall(C)| = |P_1| * |P_2| * \ldots * |P_n|$$

falls nicht zwei unterschiedliche Parametervektoren zwei identische Komponenten erzeugen, also falls gilt:

$$\text{par}(C, \vec{p}_1) \neq \text{par}(C, \vec{p}_2) \text{ für alle } \vec{p}_1, \vec{p}_2 \text{ mit } \vec{p}_1 \neq \vec{p}_2$$

Diese Annahme ist in der Praxis durchaus als realistisch anzusehen, da einzelne Parameter in der Regel unterschiedliche Eigenschaften einer Komponente beeinflussen.

Es sollen hier noch kurz die Auswirkungen der Parametrisierung beschrieben werden. Eine parametrisierte Komponente hängt in jedem ihrer Spezifikationsbereichen vom Parametervektor ab. Sei also ω_i ein Attribut einer Komponentenspezifikation. Wird die Beschreibung einer Komponente als (strukturierte) Menge ihrer Attribute aufgefasst, es gelte also

$$C = (\omega_1, \omega_2, \ldots, \omega_m)$$

so gilt bei parametrisierten Komponenten

$$\text{par}(C, \vec{p}) = (\omega_1(\vec{p}), \omega_2(\vec{p}), \ldots, \omega_m(\vec{p}))$$

Ein kurzes Beispiel soll den dargestellten Formalismus verständlicher machen. Es sei C eine Komponente zur Führung eines Girokontos einer Bank. Neben vielen anderen Attributen dieser Komponente gibt es das Attribut ω_K, das eine Bedingung für das Girokonto angibt, um wie viele Geldeinheiten das Konto maximal überzogen werden darf. Da es verschiedene Kunden gibt, die unterschiedliche Bonitäten aufweisen, wurde das Attribut ω_K beim Entwurf der Komponente parametrisierbar gehalten.

Sei also beispielsweise ω_K diejenige Komponenteninvariante, die folgenden logischen Ausdruck darstellt: $Kontostand >= max.Kreditlinie$. Da die maximale Kreditlinie variabel ist im Bezug auf die Parameter, ändert sich dieser Ausdruck bei einer parametrisierten Komponente zu $Kontostand >= max.Kreditlinie(\vec{p})$, dass heißt die maximale Kreditlinie stellt sich nun als eine Funktion der Parameterwerte, hier des Parameters Bonität oder Kundenstatus, dar.

In Becker und Overhage (2003) wurde das Problem der Parametrisierung umgangen, indem ein Bezug zur Variantenfertigung in anderen Ingenieursdisziplinen hergestellt wurde. Dieser Bezug kann mit dem hier angegebenen Formalismus nun etwas genauer gefasst werden.

In anderen Ingenieursdisziplinen werden häufig Produktvarianten verwendet, wenn sich die Erzeugnisse in ihrem Aufbau ähneln. Daher können parametrisierbare Komponenten

gewissermaßen als parametrisierbare Varianten aufgefasst werden, die beim Kunden auf ihren Einsatzzweck hin angepasst werden.

Solche Varianten werden beispielsweise von Zimmermann (1988, Abschnitt 1.2.3) beschrieben. Es gibt zwei Kategorien solcher Varianten.

- Diskrete Varianten liegen dann vor, wenn nur eine endliche Anzahl von verschiedenen Varianten möglich ist. Im hier eingeführten Formalismus bedeutet dies, dass die Menge parall(C) beschränkt ist bzw. dass $|\text{parall}(C)| < \infty$ gilt. Dies ist nur dann möglich, wenn für jede der Mengen der möglichen Ausprägungen der Parameterwerte $|P_i| < \infty$ gilt.

- Stetige Varianten liegen dann vor, wenn die Ausprägungen eines der Parameter unendlich viele Werte annehmen können. Dies ist z.B. der Fall, wenn einer der Parameter kontinuierliche Werte annehmen kann. Beispielsweise wäre ein Parameter für die Belastungsobergrenze eines Kontos sicher ein kontinuierlicher Wert. Im eingeführten Formalismus ausgedrückt, bedeutet dies, dass $|\text{parall}(C)| = \infty$ gilt. Dazu muss für mindestens eine der Parametermengen gelten: $|P_i| = \infty$.

Während der Suche könnten die diskreten Varianten implizit berücksichtigt werden, indem die Parametrisierung vor der Suche aufgelöst wird. Dabei würde die parametrisierbare Komponente ersetzt durch die Menge der einzelnen Variantenausprägungen parall(C). Dass diese Menge unter Umständen sehr groß werden kann, wirft dabei enorme Probleme auf.

Diese Idee wird jedoch unpraktikabel, wenn es stetige Varianten gibt. Dazu müsste in die Suche eine unendliche Anzahl von Kandidaten aufgenommen werden. Daher werden beim Vorliegen von stetigen Varianten neuartige Suchmethoden notwendig sein.

3.4.2 GUI Komponenten

Den meisten Komponentenbeschreibungssprachen liegt die Idee zugrunde, Komponenten zu spezifizieren, die gewisse Dienste einer Anwendungsdomäne erfüllen. So erfüllt zum Beispiel eine Buchhaltungskomponente Aufgaben des betrieblichen Rechnungswesens.

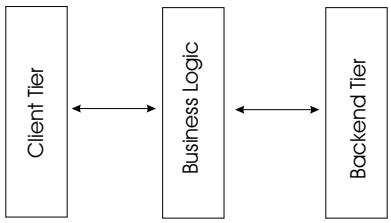

Abbildung 6: 3 Tier Architekturschema

In der Informatik wird häufig das Modell einer so genannten Three Tier Architektur (3-Ebenen-Architektur) beim Entwurf von Anwendungen benutzt (s. Abbildung 6). Hierbei wird eine Anwendung in die Ebenen Frontend, Middle Tier oder Business Logic und Backend aufgeteilt. Das Frontend dient dabei der Präsentation der Berechnungen oder der ausgeführten Datenabfragen und dient der Eingabe der zu erfassenden Informationen. Ist

die Architektur dabei z.B. in einem Netzwerk verteilt, so wird die Anwendung auf dieser Ebene auch Client genannt.

In der mittleren Ebene wird die Geschäftslogik (Business Logic) abgebildet. Sie realisiert die Geschäftsprozesse und bearbeitet die Geschäftsvorfälle. Dabei bildet diese Ebene die Schnittstelle zum Backend. Daher befinden sich auf der Middle Tier Ebene für gewöhnlich die Art von Komponenten, an die bei den allermeisten Spezifikationsvorhaben gedacht wurde.

Diese Komponenten kommunizieren mit dem Backend, das meist durch eine Art von Datenbankserver realisiert wird. Auch Großrechenanlagen bei Banken und Versicherungen wären typische Vertreter von Backend-Servern.

Wie bereits in einigen Beispielen dargestellt wurde, finden Komponenten also direkten Einsatz auf der mittleren Ebene einer solchen Architektur. Es gibt jedoch in den meisten Komponententechnologien darüber hinaus die Möglichkeit, Komponenten mit graphischer Benutzeroberfläche zu erstellen. Diese sind für den Einsatz auf der Seite des Clients gedacht. Beispielsweise ist heutzutage nahezu jedes Office Paket komponentenorientiert aufgebaut, damit es möglich ist, in ein Textdokument Tabellen einer Tabellenkalkulation einzusetzen.

Auch falls eine Konzentration auf Fachkomponenten postuliert wird, wie dies der Spezifikationsrahmen nach Ackermann et al. tut, ist nicht zu vernachlässigen, dass auch betriebliche Anwendungen komponentenorientierte Client-Anwendungen besitzen. Beispielsweise könnte es durchaus eine ganze Sammlung an Komponenten geben, die die Ergebnisse der betrieblichen Kosten- und Leistungsrechnung präsentieren können. Daher muss auch ein Spezifikationsrahmen GUI Komponenten berücksichtigen.

Da jede dieser Komponenten ihre Vor- und Nachteile besitzt, kann mit Recht gefordert werden, dass die Bewertung der GUI (Graphical User Interface, Benutzerschnittstelle) einer Komponente bei der Auswahl von Komponenten für den Einsatz auf der Seite der Client-Anwendung bedeutenden Einfluss besitzt.

Diese Einsicht wurde auf dem 5. Workshop des Arbeitskreises, der den Spezifikationsrahmen aus Abschnitt 3.3.4 entworfen hat, in ersten Ansätzen diskutiert. Dabei ist klar, dass es notwendig ist, GUI Komponenten angemessen zu spezifizieren, damit mit ihnen umgegangen werden kann. Ein Auswahlprozess ist beispielsweise auf eine bestimmte Menge von Attributen aufgebaut, die bei allen verfügbaren Alternativen bestimmt werden.

Leider sind keine Vorarbeiten in diesem Gebiet bekannt, die sich speziell mit der Spezifikation von GUI Komponenten beschäftigen, so dass bei der Erstellung einer Zielhierarchie in Abschnitt 8.1 nicht auf theoretische Grundlagen zurückgegriffen werden kann.

Es gibt jedoch Arbeiten im Bereich der COTS Evaluierung, die als Basis für die Bewertung von Benutzerschnittstellen auch bei feingranulareren Komponenten, wie sie hier im Zentrum des Interesses stehen, herangezogen werden können. So untersuchen beispielsweise Kolisch und Hempel (1996) Standardsoftware zur Projektplanung und analysieren unter anderem auch die Benutzerschnittstelle. Auch Kontio et al. (1996) berücksichtigen in ihrer Zielhierarchie Aspekte, die im Zusammenhang mit der Benutzerschnittstelle stehen. Einige Beispiele sind das Look and Feel, die (Software-)Ergonomie oder die Benutzbarkeit.

3.4.3 Qualitätsebene

Im Abschnitt 3.3.4 wurde bereits erwähnt, dass die aktuell vorliegende Version des Spezifikationsrahmens für Komponenten nach Ackermann et al. ihre Schwächen auf der Quali-

tätsebene besitzt. Das Problem ist nicht einfach zu lösen, da die Spezifikation von Qualitätskriterien auch in der Literatur bisher nur wenig behandelt wurde. Forschungsprojekte sind in diesem Bereich im Gange, deren Ergebnisse die Situation hoffentlich verbessern.

Es kann allerdings die Frage behandelt werden, wieso es so schwierig ist, eine geeignete Spezifikation für die Qualität von Software im Allgemeinen und Softwarekomponenten im Besonderen anzugeben.

Wenn die Qualität eines beliebigen Guts bestimmt werden soll, so stellt sich immer unmittelbar die Frage, wie soll eigentlich etwas abstraktes wie Qualität gemessen werden. Es müssen also Qualitätsmaße bestimmt werden, deren Summe die Vorstellung des Qualitätsbegriffs im Ganzen abdeckt. Geeignete Skalen zu finden ist hierbei schon das erste Problem. Wird eine Verhältnisskala benutzt, müsste beispielsweise die Frage, wann ein Gut die doppelte Qualität - gemessen an einem anderen Gut - hat beantwortet werden. Dass dies nicht ohne weiteres möglich ist, ist leicht einzusehen.

Weiterhin gibt es das Problem, das viele der interessanten Qualitätskriterien wie beispielsweise Performanz oder Antwortzeit für bestimmte Dienstaufrufe zwar messbar wären, jedoch hängen die Ergebnisse davon ab, auf welchem System die Messung durchgeführt wird. Ist die Komponente gut programmiert, so wird sie z.B. auf einem Mehrprozessorsystem eine deutlich bessere Performanz erreichen als auf einem Einprozessorsystem.

Zur Lösung dieses Problems wurde angedacht, eine plattformunabhängige Notation einzuführen. Dabei wurde die aus der theoretischen Informatik bekannte O-Notation vorgeschlagen. Diese Notation ist jedoch auch ungeeignet, das hier angestrebte Ziel, die Komponentenauswahl, zu unterstützen. Der Grund hierfür kann leicht eingesehen werden. Die O-Notation erreicht ihre Plattformunabhängigkeit durch Vernachlässigen der konstanten Faktoren. Hat ein Methodenaufruf nun beispielsweise ein lineares Laufzeitverhalten, so würde er in die Klasse $O(n)$ fallen. Für den Konfigurierer ist es unter Umständen wichtig, ob sich hinter diesem Ausdruck $1 * n$ oder $100 * n$ verbirgt. Denn ob der Aufruf einer Methode eine Sekunde oder einhundert Sekunden benötigt, kann entscheiden, ob die Anwendung ausreichend schnell arbeitet oder nicht.

Weiterhin kommt hinzu, dass die meisten Arbeiten, die sich mit Softwarequalität und deren Qualitätsmaßen mit allgemeinen Qualitätskriterien für die Softwareentwicklung beschäftigen. Dieses allgemeine Herangehen ist aber für den Spezialfall der komponentenorientierten Softwareentwicklung oft nicht angemessen (vgl. Bertoa und Vallecillo (2002)). Der entscheidende Unterschied liegt in der Tatsache, dass Komponenten nach dem Black-Box-Prinzip funktionieren, sprich, ohne dass der Anwender die Innensicht der Komponente kennt.

Daher unterscheiden Bertoa und Vallecillo (2002) in Analogie zur ISO Norm 9126 die Metriken zur Messung der Qualität einer Softwarekomponente in interne und externe Metriken. Interne Metriken beschreiben die Qualität der Komponente in Bezug auf ihre Innensicht während externe Metriken solche sind, die von außen ermittelt und überprüft werden können. Die Autoren geben weiterhin an, dass es nicht immer sinnvoll ist, bestimmte interne Metriken aufgrund des Black-Box-Prinzips zu ignorieren, da sie indirekt eine Aussage über die externe Qualität machen können. Es dürfte nur sehr schwer sein, Angaben zur internen Qualität auf Seiten des Käufers einer Komponente verifizieren zu wollen. Hier ist wieder das Vertrauen oder die Hoffnung in die Korrektheit der Spezifikation des Herstellers notwendig.

Aus den genannten Problemen heraus ergab sich bei der Erstellung des Spezifikationsrahmens nach Ackermann et al. offenbar das Problem, eine geeignete Menge von Qualitätsattributen einschließlich deren Klassifikation und Ermittlung angeben zu müssen. Daher wurde die Entscheidung getroffen, die Spezifikation der Qualitätsebene offen zu halten. Das bedeutet, dass lediglich allgemeine Hinweise gegeben werden, was zu spezifizieren sein könnte und in welcher Notation dies zu erfolgen habe. Darüber hinaus wird ein an die ISO Norm 9126 (vgl. Abschnitt 3.2) angelehnter Prozess angegeben, der zur Ermittlung der benötigten Daten der Qualitätsebene dienen soll.

Dies hat jedoch den entscheidenden Nachteil, dass sowohl Arbeiten wie diese, die versuchen, standardisierte Prozesse im Zusammenhang mit komponentenorientierter Softwareentwicklung zu entwickeln, als auch die Hersteller von Werkzeugen, die diese Prozesse dann unterstützen sollen, keine gemeinsame Basis haben (s. auch Overhage (2002b)).

Daher wird sich diese Arbeit an einer Studie von Bertoa und Vallecillo (2002) orientieren. Die Autoren geben Qualitätskriterien und deren Maße an, die zur Bestimmung der Qualität von COTS Komponenten geeignet sind. Dabei besteht die Hoffnung, dass diese Kriterien auch auf feingranularere Komponenten übertragbar sind.

Tabelle 1 gibt die Kriterien und Unterkriterien an, anhand derer Bertoa und Vallecillo (2002) die Qualität von Softwarekomponenten bewerten.

Kriterium	Unterkriterium (Laufzeit)	Unterkriterium (Lebenszyklus)
Funktionalität	Genauigkeit Sicherheit	Geeignetheit Interoperabilität Übereinstimmung Kompatibilität
Zuverlässigkeit	Verhalten nach Programmfehlern	Ausgereiftheit
Benutzbarkeit		Erlernbarkeit Verständlichkeit Bedienbarkeit Komplexität
Effizienz	Zeitverbrauch Ressourcenverbrauch	
Wartbarkeit		Änderbarkeit Testbarkeit

Tabelle 1: Qualitätsmodell nach Bertoa und Vallecillo (2002)

Im Folgenden sollen die verschiedenen Hauptkriterien vorgestellt werden. Für eine genaue Beschreibung der Unterkriterien und deren messbare Attribute sei auf Bertoa und Vallecillo (2002) verwiesen. Einige dieser Attribute werden in Abschnitt 8.1.2 verwendet, um die Zielerfüllung von Qualitätszielen zu bestimmen. Falls sie verwendet werden, werden sie dort kurz beschrieben.

Das Kriterium der Funktionalität gibt an, wie gut die Komponente in der Lage ist, die versprochenen Dienste zur Verfügung zu stellen, wenn die geeigneten Voraussetzungen gegeben sind. Dazu gehören die Genauigkeit, mit der Berechnungen durchgeführt werden, die vorgesehenen Möglichkeiten, um den Datenaustausch mit der Komponente abzusichern,

die Frage, ob Standarddatenformate unterstützt werden, sowie die Frage, ob neue Versionen dieser Komponente abwärtskompatibel zu älteren Versionen derselben Komponente sind. Das Geeignetheitskriterium kann nur durch den Komponentenverwender ermittelt werden und wird daher in dieser Arbeit nicht als Teil der Spezifikation angesehen sondern als Teil des Prozesses (siehe Kapitel 5).

Das Kriterium der Zuverlässigkeit misst zum einen, wie ausgereift die Komponente bereits ist, indem beispielsweise bestimmt wird, wie viele dieser Komponenten bereits erfolgreich eingesetzt werden. Zum anderen wird angegeben, wie sich die Komponente im Fehlerfall verhält, indem spezifiziert wird, ob es unter anderem Methoden zur eigenständigen Wiederaufnahme der Arbeit nach einem Systemabsturz gibt.

Mit Hilfe der Benutzbarkeit wird weiterhin die Leichtigkeit oder die Schwere gemessen, mit der ein Entwickler mit der Komponente umgehen kann, wenn er sie in eine Anwendung integrieren möchte. Es werden Eigenschaften wie Erlernbarkeit der Dienstaufrufe, Verständlichkeit der Schnittstellenbeschreibungen oder Komplexität der gebotenen Dienste beschrieben.

Auf der Effizienzebene wird beschrieben, wie gut die Komponente mit dem vorhandenen Speicherplatz bzw. mit der vorhandenen Rechenzeit umgeht. Der Speicherverbrauch wird im Einsatz direkt gemessen, der Zeitverbrauch wird durch das Antwortzeitverhalten, den Durchsatz an Daten pro Zeiteinheit und anhand der Kapazität, also der Menge an Daten, die gleichzeitig bearbeitet werden kann, bestimmt.

Die Wartbarkeit einer Komponente bezieht sich nicht auf die Wartbarkeit der internen Realisierung der Komponente sondern vielmehr darauf, inwiefern die Komponente z.B. durch Parameter angepasst werden kann. Des Weiteren wird gemessen, in wieweit eine Komponente Testroutinen zur Verfügung stellt, mit denen das einwandfreie Funktionieren - auch unter den gerade aktiven Parametereinstellungen - überprüft werden kann.

Bertoa und Vallecillo (2002) geben zusätzlich zu ihren Ausführungen zu den Qualitätsaspekten auch eine Notation zur Spezifikation der ermittelten Daten an. Diese Notation ist XML basiert, spezifiziert jedoch nur Attribut-Wert-Paare beliebiger Art. Besser wäre es, die Spezifikation innerhalb eines XML Dokuments abzulegen, bei denen die verwendeten Auszeichnungen (Tags) sprechende Namen tragen, also charakterisieren, welches Qualitätsmerkmal in ihnen gespeichert ist.

3.5 Knowledge Base

Das zentrale und innovative Paradigma der Komponentenorientierung ist die Konzentration des gesamten Entwicklungsprozesses auf die Wiederverwendung von vorgefertigten Softwarekomponenten zur Erstellung oder Wartung neuer Anwendungen. Dies wurde in dieser Arbeit schon mehrfach hervorgehoben. Zur Unterstützung dieses Paradigmas werden nicht nur technische sondern auch organisatorische Lösungen benötigt.

In der Literatur zum Software Engineering finden sich daher auch einige Ansätze, die die Wiederverwendung im Entwicklungsprozess zu unterstützen versuchen. Von diesen Ansätzen soll hier der Ansatz der so genannten Experience Factory (s. Basili et al. (1999)) kurz vorgestellt werden.

In diesem Ansatz werden zwei zentrale Einheiten bei der Softwareentwicklung betrachtet. Zum einen die entwickelnde *Development Organization* und zum anderen die *Experience Factory*. Die Development Organization ist der Teil eines Unternehmens, der die eigentliche

Entwicklungsarbeit leistet. Hierbei werden in einzelnen Entwicklungsprojekten Ergebnisse erstellt, Prozesse verwendet oder Modelle, z.B. in Bezug auf das Einsatzgebiet (Domain Engineering), auf Softwarearchitekturen, etc., erstellt. Die dort gesammelten Erfahrungen sind wertvolles Wissen für das Unternehmen, denn es bildet (Kern-)Kompetenzen, die ihm einen Wettbewerbsvorteil vor anderen Mitbewerbern verschafft.

Daher wird eine zweite Einheit eingeführt, die Experience Factory. Dieser Teil der Organisationsstruktur ist dafür verantwortlich, die gewonnenen Erkenntnisse innerhalb der einzelnen Entwicklungsprojekte zu sammeln und so abzulegen, dass die gewonnenen Erkenntnisse später wieder eingesetzt werden können. Hierzu zählen Erfahrungen zu Prozessen, wie beispielsweise dem hier vorgestellten, Erfahrungen aus der Entwicklungsarbeit, fertige Entwicklungsprojekte oder Teile daraus, Wissen über Entwicklungskosten, etc. Die gewonnen Erfahrungen dienen dann wieder innerhalb der einzelnen Projekte dazu, die Projektarbeit zu verbessern, indem die Mitarbeiter der Experience Factory die einzelnen Projekte wieder entsprechend anleiten und kontrollieren (vgl. Abbildung 7).

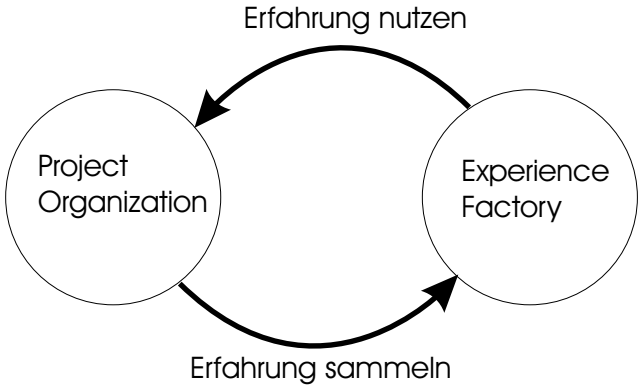

Abbildung 7: Vereinfachtes Grundmodell der Experience Factory

In den Beispielen zum Experience Factory Konzept geben Basili et al. (1999) auch das RAPID Center des amerikanischen Militärs sowie deren Aktivitäten an. Einige passen auf die Aufgaben, die im Folgenden vorgestellt werden sollen, daher werden die relevanten Aufgaben hier wiedergegeben:

- Entwicklung und Betrieb eines Repositories für wiederverwendbare Softwarekomponenten
- Unterstützung der Entwicklungsteams bei der Auswahl und der Verwendung von wiederverwendbaren Softwarekomponenten
- Unterstützung bei der Identifikation, dem Entwurf und der Realisierung von wiederverwendbaren Komponenten
- Bereitstellung von Methoden, Werkzeugen und Strategien, um eine Maximierung der Vorteile aus der Wiederverwendung zu erzielen

Diese Aufgaben sind essentiell bei der Einführung der komponentenorientierten Anwendungsentwicklung in einem Unternehmen. Diese Arbeit setzt ihren Schwerpunkt auf die zweite Aufgabe, benötigt aber die anderen Aufgabenträger, um erfolgreich zu sein.

Die Idee der Experience Factory wurde darüber hinaus in Basili et al. (1992) speziell auf die Komponentenorientierung ausgedehnt.

Hinzu kommt, dass im betrieblichen Umfeld sofort die Frage aufkommt, ob es möglich ist, nicht nur ganze Anwendungen zu verkaufen, sondern auch einzelne Teile. So ist auch vorstellbar, dass die Aufgabe der Erstellung einer Anwendung auf unterschiedliche Unternehmen verteilt werden kann, wie es in den etablierteren Ingenieursdisziplinen bereits seit langem der Fall ist. Dort ist dies an den vielen Unternehmen zu erkennen, die sich auf das Erstellen von Einzelteilen als Zulieferer spezialisiert haben. Hierzu müssen jedoch Marktplätze dem Konzept hinzugefügt werden. Diese dienen de facto als externe Experience Factories, auf denen die Erfahrungen von Drittanbietern eingekauft werden können.

Daher werden im Folgenden unternehmensinterne Komponentenrepositories sowie öffentliche Komponentenmarktplätze als wichtigste Quellen zur Suche nach Komponenten vorgestellt, damit dann in folgenden Kapiteln auf besondere Such- und Auswahlstrategien eingegangen werden kann.

3.5.1 Unternehmensinterne Repositorien

Ein Repository oder Metainformationssystem dokumentiert die Informationsverarbeitung eines Unternehmens (siehe Ortner (1999a) und Ortner (1999b)). Dabei können unterschiedlichste Bereiche dokumentiert werden. Angefangen von der eingesetzten Software hin zu Entwicklungsdatenbanken oder zu eingesetzten Datenbanksystemen. Aber auch die Fachentwürfe, die zur Erstellung der einzelnen Systeme geführt haben, sollten dokumentiert werden, damit jederzeit nachvollziehbar bleibt, welche Entscheidung auf welcher betrieblichen Notwendigkeit beruht.

Für diese Arbeit sind Entwicklungsrepositorien besonders interessant, da in ihnen die Entwicklungsergebnisse eines Softwarehauses abgelegt werden können, also insbesondere ein Repository für die erstellten Komponenten.

Repositorien benötigen hierzu ein so genanntes Metaschema, also ein Schema, mit dessen Hilfe Schemata beschrieben werden können. Solch ein Schema ist im betrachteten Fall eine Komponentenspezifikation, da diese vorgibt, nach welchem Muster die beschriebene Komponente arbeitet. Hier zeigt sich übrigens der Zusammenhang zwischen Spezifikation und Repository. Aber auch bei der Suche nach den Elementen innerhalb eines Repositories spielt das Metaschema eine wichtige Rolle, denn es gibt die maximale Menge an Suchattributen implizit vor. Nach mehr Informationen, als denen, die im Metaschema abgelegt wurden, kann logischerweise nicht gesucht werden.

Standardsoftware gibt es in diesem Bereich nach dem aktuellen Wissen nur wenige. Eins der wenigen kommerziell verfügbaren Produkte ist der „Select Component Manager“ (s. ComponentManager (2003)). Dieses Repository bietet die Möglichkeit, Komponenten zu spezifizieren, sowie die Spezifikationen mit einem Marktplatz abzugleichen.

3.5.2 Komponentenmarktplätze

Marktplätze dienen dem Ziel, Anbieter und Nachfrager von bestimmten Gütern zusammen zu bringen und damit eine optimale Allokation dieser Güter zu erreichen. Dadurch übernehmen die Marktplätze eine wichtige Rolle als Vermittler zwischen den Marktteilnehmern.

Ein funktionierender Komponentenmarkt ist die wichtigste Voraussetzung für den wirtschaftlichen Durchbruch der Softwareentwicklung unter Einsatz von Komponenten. Genau wie in den etablierten Ingenieursdisziplinen werden auf einem Komponentenmarktplatz Teile angeboten, die meist in Katalogen sortiert sind. Diese Kataloge kann der Nachfrager durchstöbern oder gezielt nach den benötigten Gütern durchsuchen.

Bisher konnten sich auch Komponentenmarktplätze - genau wie die Komponentenorientierung im Ganzen - nicht in nennenswertem Umfang etablieren. Eine aktuelle Übersicht über die verfügbaren Marktplätze und eine Klassifizierung der von ihnen unterstützten Funktionen findet sich in Fettke et al. (2003).

Unter den in dieser Schrift untersuchten Marktplätzen besonders erwähnenswert sind die Marktplätze ComponentSource und CompoNex. ComponentSource (s. ComponentSource (2003)) ist von daher besonders interessant, da dieser Marktplatz von sich selbst behauptet, der weltweit führende zu sein. Daher dient die Menge der auf ComponentSource angebotenen Komponenten als Indikator dafür, wie weit Komponentenmarktplätze bereits etabliert sind. Die Untersuchung hat ergeben, dass auf ComponentSource ca. 8000 Komponenten gehandelt werden. Diese Menge erscheint extrem gering, wenn die Breite der Anwendungsgebiete von Informationssystemen, die insbesondere in den letzten Jahren beachtlich gewachsen ist, betrachtet wird.

Die Suchkriterien, mittels derer Komponenten auf ComponentSource gesucht werden können, beziehen sich neben einer Volltextsuche mit Hilfe von Stichwörtern hauptsächlich auf Angaben zu den Basistechnologien. So können Angaben wie Betriebssystem oder einzusetzende Komponententechnologie gemacht werden, aber auch Angaben zur verwendeten Datenbank oder zu Kommunikationsmöglichkeiten mit anderen Anwendungen oder Komponenten.

An der Technischen Universität Darmstadt wurde ein Prototyp eines weiteren Marktplatzes für Softwarekomponenten im Rahmen eines Wirtschaftsinformatikpraktikums erstellt (s. Overhage (2002b)). Dieses Projekt, das den Namen CompoNex trägt, ist in der neuen Web Service Technologie realisiert, wodurch es ermöglicht wird, den Marktplatz als Komponente direkt in die Ablauforganisation bei der Erstellung von Anwendungssystemen zu integrieren. Eine solche Integrationsmöglichkeit hilft bei einem prozesszentrierten Ansatz, wie er in dieser Arbeit verfolgt wird, da die Verwendung des Marktplatzes direkt in die Entscheidungsprozesse bei der Suche und Beschaffung von Komponenten eingebunden werden kann.

Zum anderen versucht dieser Marktplatz den Spezifikationsrahmen nach Ackermann et al. sowohl in seinem Komponentenkatalog als auch bei der Suchfunktionalität abzudecken. Hierzu wurden große Teile des Spezifikationsrahmen in ein XML Schema abgebildet, das der Marktplatz als Grundlage für sein Metaschema verwendet. Damit wurde auch prinzipiell gezeigt, dass der Spezifikationsrahmen als Grundlage für Werkzeuge zur komponentenorientierten Anwendungsentwicklung geeignet ist.

Die Arbeit von Varadarajan et al. (2002) greift CompoNex auf, um es mit einem eigenen Konzept zu vergleichen, das ähnlich dem von CompoNex angelegt ist. Die Komponenten werden in dem ComponentXChange genannten Konzept ebenfalls in XML beschrieben und über einen öffentlich zugänglichen Dienst gehandelt.

3.6 Speicherstrukturen für die Ergebnisse

Bei jedem Methoden- und Werkzeugeinsatz entstehen Ergebnisse. Diese müssen festgehalten werden, damit sie für den Fortschritt innerhalb der Vorgehensweise einen Beitrag leisten können. Sind nur einzelne Personen an einer Tätigkeit beteiligt, so wird in der Praxis häufig vergessen, die Ergebnisse ausreichend zu dokumentieren. Dies führt oft dazu, dass nach dem Ausscheiden der Mitarbeiter sowohl die Erfahrung als auch das konkrete Wissen über diese Anwendungen für die Unternehmung verloren geht.

Aus diesem Grund ist es im Software Engineering unbedingt notwendig, die erstellten Ergebnisse ausreichend zu dokumentieren. Hierzu werden daher geeignete Dokumentationssprachen benötigt. Für das Konfigurationsmanagement komponentenorientierter Anwendungen lohnt sich ein Blick über den Tellerrand zu den etablierten Verfahren aus der Produktionswirtschaft oder der allgemeinen Betriebswirtschaftslehre, um Anregungen für mögliche Speicherstrukturen zu finden.

Als Ergebnis des Konfigurationsmanagements steht am Ende eine Konfiguration von Komponenten, deren Zusammenspiel in einer Anwendung realisiert wird oder wurde. Wird die Analogie zum Maschinenbau oder der Elektrotechnik gezogen, so entspricht eine konfigurierte Anwendung einer fertigen Maschine, einer Schaltung oder einem komplexen Bauteil. In den erwähnten Ingenieursdisziplinen kommen zur Dokumentation dieser Erzeugnisse Baupläne und Stücklisten zum Einsatz, daher soll im Anschluss an diese Einleitung eine Verwendung dieser Hilfsmittel im Komponentenkonfigurationsmanagement diskutiert werden.

Die Wichtigkeit der genauen Definition der Struktur der Ergebnisse wurde bisher noch nicht ausreichend motiviert. Ergebnisse stellen Wissen und Erfahrungen eines Unternehmens dar. Dieses Wissen ist gemäß Abschnitt 3.5 in einer Experience Factory abzulegen. Im Modell der Unternehmung, wie es hier beschrieben wird, dient dazu das Unternehmensrepository aus Abschnitt 3.5.1. Ein Repository benötigt jedoch zur Speicherung von Ergebnissen Informationen über diese Ergebnisse. Dazu müssen sie genau bekannt und analysiert sein. Werden beispielsweise Textdateien innerhalb von Arbeitsschritten erstellt, so wird eine Speicherstruktur benötigt, die in der Lage ist, Textdateien zu beschreiben. Daher ist es wichtig, nicht nur die Resultate eines Arbeitsschritts zu kennen, sondern auch Wissen über die Struktur der Ergebnisse zu besitzen, damit diese dokumentiert und bei Bedarf wieder gefunden werden können.

Im Rahmen des Änderungsmanagements werden diese Informationen für das betroffene Unternehmen wieder extrem wichtig. Angenommen, es wird eine Komponente verwendet, die bisher in ihrer ersten Version vorlag. Der Hersteller dieser Komponente bietet ein Update auf eine neuere Version der Komponente an. Mittels des Repositories muss es nun möglich sein, alle von dieser Änderung betroffenen Anwendungen effizient aufzufinden, um entscheiden zu können, ob die Anwendungen gewartet werden sollen oder nicht.

3.6.1 Baupläne

Baupläne werden benutzt, um aufzuzeigen, welche Einzelteile eines Erzeugnisses in welcher Art und Weise zu einem fertigen Produkt zusammengesetzt werden. Dazu wird meist genau aufgezeigt, zu welchem Zeitpunkt an exakt welche Stelle ein Einzelteil einzusetzen ist.

Es gibt in diesem Bereich bereits einige Überlegungen, wie dieses Hilfsmittel auf die

komponentenorientierte Entwicklung von Informationssystemen angepasst werden kann. Dabei wurde daran gedacht, die Klassendiagramme der Unified Modelling Language (UML) anzupassen, um damit Baupläne für Anwendungen darstellen zu können.

Diese Diagramme müssen die Parametrisierung der Einzelkomponenten speichern und zusätzlich, welche Teile zu welchem Zeitpunkt bzw. zu welcher Dispositionsstufe verbaut werden müssen. Dabei könnten die Komponenten durch eine ähnliche Symbolik wie die Klassen repräsentiert werden, während eine neu zu definierenden Beziehungen für den Zusammenhang beim Zusammenbau der Anwendung stehen könnten.

Wichtig ist auch, dass die Konstruktion von Varianten berücksichtigt wird. So könnte es an einigen Stellen im Bauplan Wahlmöglichkeiten geben, je nachdem, welche Funktionalität der Kunde für seine Anwendung wünscht. Beispielsweise könnte die Anwendung verschiedene Arten von Zahlungssystemen, wie EC-Karte oder Überweisung, beherrschen.

Da jedoch auf diesem Gebiet aktuelle Arbeiten erst im Gange sind, deren Ergebnisse noch nicht veröffentlicht wurden, soll hier nicht weiter auf diese Möglichkeit eingegangen werden.

3.6.2 Stücklisten

Der genaue Zusammenhang, welches Teil in welche Baugruppe zu welchem Zeitpunkt wie eingebunden wird, ist häufig zu detailliert für das angestrebte Ziel, eine effiziente Speicherstruktur für die Speicherung von Konfigurationen abzuleiten. Daher wurden von Ortner et al. (1999) die Idee der Verwendung von Stücklisten zur Speicherung von Konfigurationsdaten vorgeschlagen. Becker und Overhage (2003) haben diese Idee aufgegriffen und in einem Positionspapier zu dieser Diplomarbeit auf das Komponentenkonfigurationsmanagement angepasst.

Stücklisten speichern die mengenmäßige Zusammensetzung von Erzeugnissen aus Teilen und Baugruppen. Im betrachteten Kontext sind dies also Komponenten bzw. Konfigurationen von Komponenten. Wie im Beitrag von Becker und Overhage (2003) ausgeführt wurde, gibt es in der Literatur zur Produktionswirtschaft eine Klassifikation der verschiedenen Stücklisten, die hier kurz wiedergegeben werden soll (vgl. Abbildung 8). Die Klassifikation stammt zum größten Teil aus Schneeweiß (1999).

- Übersichts- bzw. Mengenstücklisten geben in aggregierter Form an, welche Teile in welchen Mengen in das Endprodukt eingehen.

- Strukturstücklisten erfassen zusätzlich Details zu den Baugruppen, aus denen das Endprodukt zusammengesetzt wird. Die einzelnen Baugruppen und ihre Weiterverwendung werden in diesen Stücklisten explizit abgebildet. Gozinto-Graphen eignen sich dabei als graphische Repräsentation für Strukturstücklisten.

- Dispositionsstücklisten führen zusätzlich zu den Eigenschaften der Strukturstücklisten die Speicherung der Dispositionsstufe ein. Dabei wird die Strukturstückliste so in Ebenen aufgeteilt, dass alle Teile einer Art im selben Produktionsschritt disponiert werden können. Kommen also gleiche Teile auf unterschiedlichen Produktionsstufen in der Strukturstückliste vor, so werden sie einheitlich auf die unterste mögliche Dispositionsstufe eingeordnet.

- Baukastenstücklisten sind Strukturstücklisten, die jeweils nur eine einzige Dispositionsstufe darstellen und damit immer nur ein einzelnes Bauteil. Becker und Overhage (2003) geben den Hinweis, dass diese Aufspaltung dem üblichen Vorgehen bei der komponentenorientierten Softwareentwicklung entspricht, Baugruppen zu bilden und zu konstruieren. Diese Baugruppen werden wieder als Black-Box-Komponente betrachtet, so dass damit weitere Bauteile oder die gesamte Anwendung zusammengesetzt werden können.

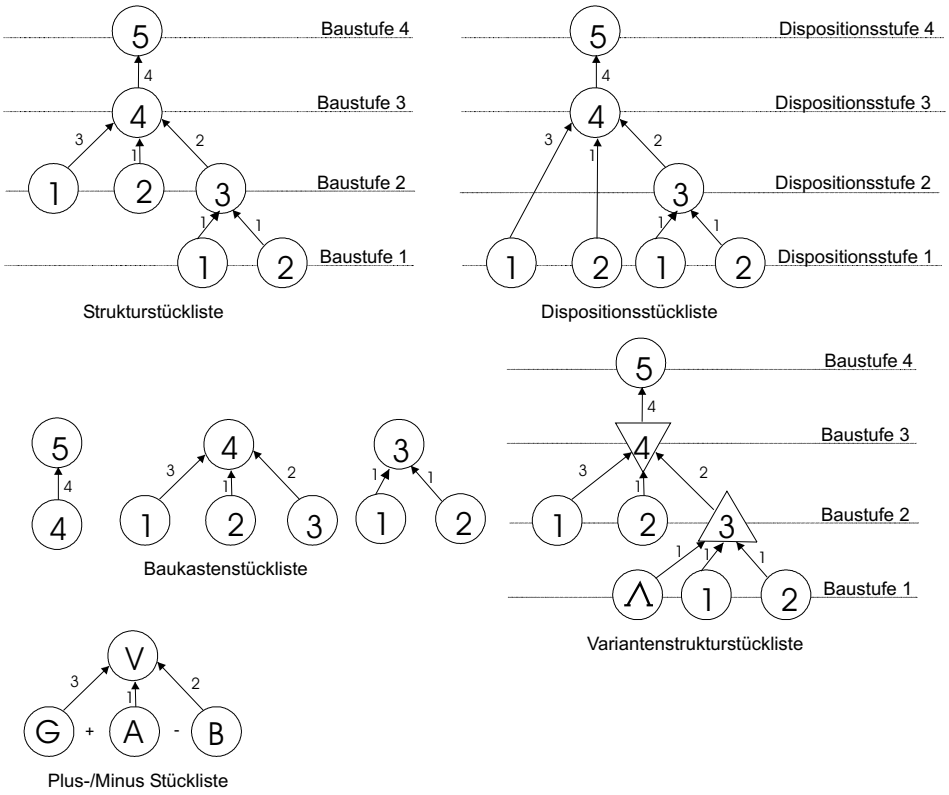

Abbildung 8: Verschiedene Arten von Stücklisten

Im Abschnitt 3.4.1 wurde bereits der Zusammenhang zwischen Varianten und parametrisierbaren Komponenten dargestellt. Auch in der Produktionswirtschaft werden häufig Varianten verwendet, beispielsweise kommen bei der Konstruktion von PKW je nach Kundenwünschen verschiedene Ausstattungsvarianten zum Einsatz. Dies hat nicht nur Auswirkungen auf die Baupläne (s. Abschnitt 3.6.1), sondern natürlich auch auf den Einsatz von Stücklisten. Daher wurden in der Literatur verschiedene Möglichkeiten vorgeschlagen, Variantenstücklisten zu speichern.

Eine Erweiterung der bereits vorgestellten Stücklisten wurde von Wedekind und Müller (1981) vorgeschlagen. Dazu werden neue Knotentypen eingeführt, die dazu dienen, Auswahlmöglichkeiten in der Stückliste abzubilden. Ein solcher so genannter „Alternativknoten“ (Dargestellt durch ein Dreieck mit der Spitze nach unten) bedeutet, dass genau ein Nachfolger dieses Knoten ausgewählt wird und an die Stelle des Alternativknotens tritt. Ein Beispiel einer Buchhaltungsbaugruppe ist in Abbildung 9 zu sehen.

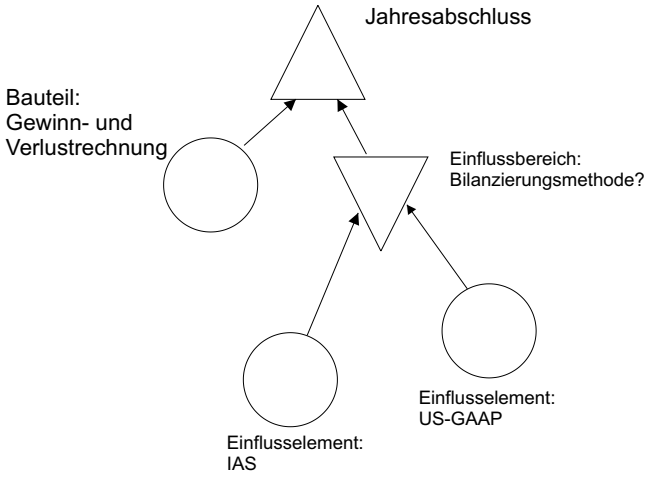

Abbildung 9: Variantenstückliste einer Buchhaltungskomponente

Zur effizienten Speicherung von Variantenstücklisten gibt es in der Literatur (vgl. Zimmermann (1988)) noch einige weitere Vorschläge. So speichern beispielsweise Plus-Minus-Stücklisten eine Grundvariante und dazu pro Variante jeweils, welche Teile hinzugefügt bzw. weggelassen werden. Ein Sonderfall hiervon sind die fiktiven bzw. Gleichteilestücklisten, die nur additive Positionen speichert, d.h. hier kann die Grundvariante immer nur um geeignete Teile erweitert werden.

Prinzipiell sind alle Formen der Stücklisten geeignet, Ergebnisse der Konfiguration zu speichern. Variantenstücklisten eignen sich dabei wohl am Besten, da bei der Anwendungsentwicklung mit größeren Variantenzahlen zu rechnen ist. Es sei hierbei nur auf die vielen Anpassungsmöglichkeiten einer Software wie das SAP R/3 System hingewiesen, wovon die meisten über Produktvarianten zu realisieren wären.

Die Speicherung in Form von Plus-Minus-Stücklisten erscheint hierbei effizient, solange die Unterschiede zwischen der Grundvariante und den Einzelvarianten relativ gering bleiben. Ansonsten wären wohl Strukturvariantenstücklisten ganz gut geeignet.

3.7 Entscheidung

Nach der Suche nach Komponenten, entweder in internen Repositorien oder auf externen Komponentenmarktplätzen, existiert eine Menge von möglichen Kandidaten für die zu verwendende Komponente oder die zu verwendenden Komponenten einer möglichen Konfiguration. In dieser Situation ist es erforderlich, dass eine Entscheidung für eine der möglichen Alternativen getroffen wird. Meistens wird diese Entscheidung ohne größere Überlegungen getroffen, die damit mehr oder weniger ad hoc fällt und damit nicht dem betriebswirtschaftlichen Anspruch durchdachter und geplanter Handlungen entspricht. Um solche Entscheidungen auf eine bessere Grundlage zu stellen, wurden in der Entscheidungstheorie Untersuchungen angestellt, wie solche Situationen systematisch analysiert werden können und wie die Entscheidungsträger zu rationalen Entscheidungen kommen können.

Da hierbei die Alternativenmenge explizit gegeben ist, handelt es sich um ein Problem der Entscheidungstheorie und nicht um ein Problem aus dem Operations Research. Pro-

bleme aus dem Operations Research sind dadurch charakterisiert, dass die Lösungsmenge implizit durch einschränkende Nebenbedingungen gegeben ist, während in der Entscheidungstheorie die Alternativen explizit gegeben sein müssen.

Wann eine Entscheidung als rational gilt, kann nicht eindeutig festgelegt werden (vgl. Eisenführ und Weber (2003, S. 5)). Anhand von Eigenschaften, die allgemein an rationale Entscheidungen gestellt werden, kann der Begriff trotzdem einigermaßen gefasst werden. Von Eisenführ und Weber (2003) wird gefordert, dass rationale Entscheidungen die Kriterien „prozedurale Rationalität“ und „Konsistenz“ erfüllen.

Prozedurale Rationalität liegt vor, wenn die Prozedur, mit der die Entscheidung getroffen wird, „mehr oder weniger rational“ ist. Zur Konkretisierung dieser Aussage wird gefordert, dass das richtige Problem zu lösen ist, der Aufwand der Informationsbeschaffung der Entscheidung angemessen ist, dass Zukunftserwartungen möglichst objektiv gebildet werden und dass sich die Entscheider über ihre Ziele und Präferenzen gründlich klar werden.

Die Entscheider werden in dem hier vorliegenden Anwendungsfall bei der prozeduralen Rationalität durch den vorgeschlagenen Prozess unterstützt, der sowohl Ideen zur Informationsbeschaffung enthält als auch helfen soll, die Ziele und Präferenzen klar zu ermitteln. Mögliche Ziele werden bereits ermittelt und vorgeschlagen und zur Ermittlung der Präferenzen wird der AHP eingesetzt (s. Abschnitt 8.1).

Konsistent soll eine Entscheidung genannt werden, wenn sie die Kriterien der Zukunftsorientierung, der Transitivität, der Irrelevanz gegenüber der Darstellung des Entscheidungsproblems und der Unabhängigkeit von irrelevanten Alternativen erfüllt. Zukunftsorientierung meint, dass die Auswahl einer Alternative ausschließlich aufgrund der zu *erwartenden* Konsequenzen getroffen wird. Transitivität bedeutet, dass gefordert wird, falls eine Alternative B besser als eine Alternative A ist und falls eine Alternative C besser als eine Alternative B ist, dass dann auch Alternative C besser als Alternative A bewertet wird.

Da Entscheidungsprobleme in der Realität häufig von komplexer Natur sind, werden sie in einzelne Bausteine aufgespalten und jeder dieser Bestandteile wird einzeln modelliert (s. Eisenführ und Weber (2003, S. 16)). Als Bausteine zur Modellierung einer Entscheidungssituation identifizieren die Autoren Handlungsalternativen, Umwelteinflüsse, die Konsequenzen von Aktionen und Umwelteinflüssen sowie die Ziele und Präferenzen der Entscheider. Auf die einzelnen Bausteine soll im Folgenden näher eingegangen werden. Abschließend werden Verfahren vorgestellt, die die Entscheidungsfindung unterstützen sollen, sowie die Frage nach der Sensitivität der Ergebnisse betrachtet.

3.7.1 Alternativen

Die Handlungsalternativen beschreiben die möglichen Aktionen, die ein Entscheider durchführen kann, oder anders gesagt, die Summe der Optionen, die er in der gegebenen Situation besitzt, um zu handeln. Alternativen müssen zu ihrer Modellierung zuerst gefunden und aufgestellt werden. Damit beschäftigt sich Kapitel 5. Als Ergebnis entsteht die so genannte Alternativenmenge A, die die relevanten Handlungsalternativen enthält.

Die Menge kann unter Umständen recht schnell sehr groß werden, was ein Problem für die meisten entscheidungsunterstützenden Verfahren ist. Zum einen können kontinuierliche Entscheidungsvariablen auftreten - in diesem Fall ist die Alternativenmenge nicht beschränkt. Eisenführ und Weber (2003, S. 18) schlagen hierfür eine Diskretisierung der

Alternativen vor, indem die Entscheidungsvariablen in Klassen eingeteilt werden und dann als Vereinfachung mit den diskreten Klassen gearbeitet wird.

Zum anderen kann sich die Menge durch die Forderung vergrößern, dass sich Alternativen gegenseitig ausschließen müssen bzw. dadurch, dass sie unabhängig voneinander sein müssen. Ist beispielsweise die Entscheidung zu fällen, am Morgen zum Bäcker oder zum Metzger gehen zu wollen und am Abend den jeweils anderen Einkauf zu erledigen, wodurch eine Abhängigkeit gegeben ist, ergeben sich die Handlungsalternativen zu

- Morgens zum Bäcker fahren und abends zum Metzger
- Morgens zum Metzger und abends zum Bäcker fahren

Für den Fall, dass sich Alternativen nicht ausschließen und nicht voneinander abhängen, sie aber trotzdem zusammen entschieden werden müssen oder sollen, ergibt sich sogar die Potenzmenge als Alternativenmenge. Anhand des Beispiels aus Eisenführ und Weber (2003) soll dies aufgezeigt werden. Dort besteht die Auswahl darin, mittags essen zu gehen oder zu Hause zu essen und abends ein Buch zu lesen oder fern zu sehen. Wird hierfür nun ein kombiniertes Entscheidungsproblem benötigt, so würden sich die folgenden vier Alternativen ergeben:

- Mittags essen gehen und abends fernsehen
- Mittags essen gehen und abends lesen
- Mittags zu Hause essen und abends fernsehen
- Mittags zu Hause essen und abends lesen

Es ist zu erkennen, dass die Größe der Alternativenmenge beim Vorliegen weiterer Einzelentscheide, die, wie hier geschehen, zu kombinierten Alternativen zusammengesetzt werden müssen, exponentiell ansteigt. Dies gilt es im Sinne einer effizienten Entscheidung jedoch zu verhindern, da ansonsten die Evaluierung der einzelnen Alternativen allein aufgrund ihrer Masse nur noch oberflächlich erfolgen kann.

Der Grund, wieso hier ein besonderer Schwerpunkt auf die Probleme im Zusammenhang mit der Alternativenmenge gelegt wird, besteht darin, dass die Entscheidungssituationen, wie sie hier angegangen werden sollen, häufig Abhängigkeiten oder Zusammenhänge in den Komponenten enthalten. Daher muss die Ermittlung der Alternativenmenge mit großer Sorgfalt erfolgen. Der in Kapitel 5 vorgestellte Prozess versucht daher, eine gute Alternativengenerierung zu ermöglichen.

3.7.2 Szenarien

Entscheidungen können unter Sicherheit erfolgen, dann ist zu jeder Alternative die Erwartung an den Nutzen unabhängig von äußeren Einflüssen. In Becker und Overhage (2003) sind die Autoren beispielsweise von der Annahme ausgegangen, dass die Auswahl einzelner Komponenten eine Entscheidung unter Sicherheit sei, da die Komponenten alle bekannt sind und eine Auswahl direkt eine Komponente bestimmt und somit auch direkt der Nutzen

dieser Komponente bekannt ist. In der Realität gibt es allerdings nahezu keine Entscheidung unter vollkommener Sicherheit. Auch im hier betrachteten Anwendungsgebiet ist dies so.

Beispielsweise führt die Einbeziehung der Frage, wie korrekt die Spezifikation einer Komponente wirklich ist, in der Realität zu verschiedenen möglichen Zukunftserwartungen. Falls die Komponente korrekt spezifiziert ist, so erfüllt sie sicher den Nutzen, der ihr direkt zugerechnet wurde. Enthält die Spezifikation jedoch Fehler oder Ungenauigkeiten, so kann die Komponente unter Umständen ihren gedachten Nutzen nicht mehr entfalten.

Weiterhin ist ungewiss, wie eine Anwendung nach ihrer erstmaligen Einführung weiterverwendet und weiterentwickelt wird. Dabei handelt es sich also um die Frage nach den erwarteten Szenarien für spätere Wartungsarbeiten. Anhand eines Beispiels soll dies kurz verdeutlicht werden. Angenommen ein Unternehmen plant schon seit längerer Zeit eine Expansion in die Vereinigten Staaten von Amerika. Dort werden Bilanzen nach den Vorschriften des US-GAAP erstellt. Wird diese Zukunftsaussicht bzw. dieses Zukunftsszenario berücksichtigt, so könnte eine Softwarekomponente, die die Buchhaltung nach HGB und nach US-GAAP beherrscht eine bessere Bewertung bekommen als eine Komponente, die nur nach HGB bilanziert. Würde der Umstand der möglichen Expansion jedoch außer Acht gelassen, fiele die Wahl vielleicht eher auf die HGB-Komponente, da sie vermutlich den gleichen Nutzen bringt und dabei aber weniger kostet.

3.7.3 Präferenzstruktur

Nach Eisenführ und Weber (2003, S. 31) sind Präferenzen „Einstellungen des Entscheiders zu Konsequenzen oder zu Handlungsalternativen“. Zwischen zwei Alternativen $a, b \in A$ gibt es eine Relation, die die Präferenz beschreibt:

$a \succ b$	a wird gegenüber b präferiert
$a \sim b$	Indifferenz zwischen a und b
$a \succeq b$	a wird gegenüber b präferiert oder es herrscht Indifferenz

Die Attribute, an denen der Entscheider seine Präferenzen festmacht, werden auch als Zielgrößen bzw. Zielvariablen bezeichnet. Präferenzen werden in der Entscheidungstheorie durch so genannte Präferenzfunktionen abgebildet. Im Fall sicherer Erwartungen werden sie Wertfunktionen und im Fall unsicherer Erwartungen Nutzenfunktionen genannt. Die Funktionen werden meist losgelöst vom Entscheidungsproblem durch geeignetes Befragen des Entscheiders bestimmt.

3.7.4 Zielsystem

In der Entscheidungstheorie kommt zu den Präferenzen der Entscheider der Begriff der Ziele hinzu. „Ein Ziel ist eine Eigenschaft (Zielvariable) in Verbindung mit einer Angabe über die Präferenz des Entscheiders bezüglich dieser Eigenschaft.“ (Eisenführ und Weber (2003, S. 31))

Dabei sind Zielfunktionen häufig monoton. Beispielsweise gilt bei der Beurteilung von Kosten meist der Grundsatz, je billiger umso besser. Oder bei der Bewertung von Grundstücken gilt für die meisten Menschen, je mehr Grundfläche, umso besser. Es gibt aber auch Zielfunktionen, die ein lokales Optimum besitzen. Ein Entscheider könnte etwa die

Motorleistung eines PKW nur bis zu einem bestimmten Punkt monoton besser bewerten. Wird der Motor zu leistungsfähig, dann könnte er Angst haben, ständig die Geschwindigkeitsbeschränkungen zu übertreten und damit wird der Motor für ihn wieder weniger nützlich.

Im Zusammenhang mit Zielen sollte noch erwähnt werden, dass in komplexen Entscheidungssituationen meist nicht nur ein Ziel zu optimieren ist, sondern dass in der Regel mehrere Ziele verfolgt werden. So soll das Auto aus dem vorangegangenen Beispiel nicht nur eine möglichst hohe Motorleistung haben, sondern dabei auch möglichst wenig Benzin verbrauchen. Wie in diesem Beispiel stehen solche Ziele häufig in Konflikt zueinander, da eine Erhöhung der Motorenleistung meistens auch den durchschnittlichen Benzinverbrauch steigert.

Weiterhin werden Ziele unterteilt in Fundamental- und in Instrumentalziele (s. Eisenführ und Weber (2003, S. 56)). Fundamentalziele sind solche Ziele, die um ihrer selbst Willen verfolgt werden, ohne dass es einer weiteren Begründung bedarf. Instrumentalziele werden verfolgt, weil sie einen positiven Beitrag zur Erfüllung eines Fundamentalziels leisten. Analog können Ziele aufgeteilt werden in Ober- und Unterziele, wobei Unterziele ebenfalls dazu dienen, die Zielerreichung des Oberziels zu verbessern. Durch diese Strukturierung ist es möglich, ganze Zielhierarchien zu erstellen. Der im Kapitel 8 verwendete AHP nutzt solche Zielhierarchien aus.

Als Zielsystem bezeichnen Eisenführ und Weber (2003) die Gesamtheit aller Ziele in einer bestimmten Entscheidungssituation. Die Ermittlung eines solchen Zielsystems für die Auswahl von Komponenten bzw. von Konfigurationen von Komponenten auf Grundlage dieses Kapitels wird die Aufgabe in Abschnitt 8.1 sein. Daher ist es sinnvoll, die in der Literatur aufzufindenden Anforderungen an ein Zielsystem zu betrachten. Eisenführ und Weber (2003, S. 60) geben hierzu fünf Kriterien an:

- *Vollständigkeit* liegt vor, wenn der Entscheider alle für die Entscheidung relevanten Ziele im betreffenden Zielsystem abgebildet hat.

- *Redundanzfrei* ist ein Zielsystem, wenn es keine Ziele gibt, die sich in ihrer Bedeutung überschneiden. Andernfalls besteht die Gefahr, dass ein Ziel überbewertet wird, da es über mehrere Ziele abgebildet wird.

- *Messbarkeit* besteht, wenn der Grad der Zielerreichung „möglichst treffend“ und „möglichst eindeutig“ messbar ist. Möglichst treffend ist ein Maß dann, wenn genau das gemessen wird, was dem Entscheider wirklich wichtig ist. Möglichst eindeutig bedeutet, dass die Unschärfe in der Messung so klein wie möglich sein sollte, das Maß damit also möglichst exakt ist.

- *Präferenzunabhängigkeit* liegt im Zielsystem vor, wenn ein Entscheider seine Präferenzen gegenüber einer einzelnen Zielvariablen unabhängig von den Ausprägungen der anderen Zielvariablen formulieren kann. Diese Eigenschaft ist daher von großer Bedeutung, da sie die Verwendung einer additiven multiattributiven Wertfunktion zur Aggregation der Gesamtpräferenz für eine Alternative ermöglicht. Anders ausgedrückt bedeutet dies, dass sich der Gesamtnutzen einer Alternative als die einfache Addition der einzelnen Bewertungen für die jeweiligen Zielvariablen berechnet. Die Präferenzunabhängigkeit ist in der Praxis mit Sorgfalt zu prüfen, da sie eine *subjektive* Eigenschaft ist.

- *Einfachheit* kann einem Zielsystem bescheinigt werden, wenn es nur wenige Ziele enthält. Dies ist unter der geforderten Vollständigkeit meist nicht einfach zu erreichen. Eventuell ist eine Aggregation von Unterzielen zu entsprechenden Oberzielen möglich, wenn die Oberziele selbst noch relativ einfach zu bewerten sind. Außerdem kann das Zielsystem vereinfacht werden, wenn sich herausstellen sollte, dass alle Alternativen in einer gegebenen Zielvariablen den gleichen bzw. annähernd selben Wert besitzen. Dann kann das ganze Ziel für die Entscheidungsfindung außer Acht gelassen werden, da es keine Alternative besser oder schlechter als die anderen stellt.

Das in dieser Arbeit zu ermittelnde Zielsystem muss sich an diesen Anforderungen messen lassen. Insbesondere die letzten beiden Eigenschaften lassen sich jedoch nur schwer überprüfen bzw. sicherstellen.

3.7.5 Entscheidungstheoretisches Verfahren

Zur Ermittlung der benötigten Informationen (Alternativen, Präferenzen, Zielsystem) gibt es in der Literatur eine Reihe von entscheidungsunterstützenden Verfahren (in der englischen Literatur „decision-making techniques“). Ncube und Dean (2002) geben einige solcher Verfahren an, darunter befinden sich MAUT, MCDA, gewichtete Scoring Modelle (WSM/WAS) oder der AHP.

MAUT steht für Multi-Attribute Utility Theory, eine Theorie, die mit strengen Axiomen versucht, rationale Entscheidungen in multiattributiven Entscheidungssituationen zu treffen. MCDA ist eine Abkürzung für Multi-Criteria Decision Aid, einer Anwendung der multiattributiven Nutzentheorie. Es beschreibt einen kompletten Prozess zur Auswahl von Komponenten unter Einbeziehung von MAUT als Entscheidungsverfahren. Darüber hinaus werden auch organisatorische Rollen definiert und Anleitungen gegeben, wie die Evaluierungskriterien zu bestimmen sind.

Die Scoring Modelle WSM (Weighted Score Method) und WAS (Weighted Average Sum) dürften den meisten Betriebswirten am ehesten ein Begriff sein, da auch die bekannte Nutzwertanalyse nach der Methode von gewichteten Punktwerten arbeitet. Dabei werden den einzelnen Kriterien auf vorgegebenen Skalen Punkte zugeordnet, beispielsweise von 1 bis 10. Die Kriterien selbst werden bezüglich ihrer Wichtigkeit untereinander mit Punktwerten versehen. Diese Punktwerte werden meist in der Summe auf eins normiert und dann dazu benutzt, um die Bewertungen der einzelnen Kriterien zu aggregieren, indem die Summe der Einzelwerte multipliziert mit ihren entsprechenden Gewichten gebildet wird:

$$\text{score}_a = \sum_{j=1}^{n} (\text{weight}_j * \text{score}_{aj})$$

Seine Bekanntheit hat dieses Verfahren aufgrund seiner Einfachheit und leichten Verständlichkeit erlangt. Nachteilig ist jedoch, dass die Vergabe der Punktwerte ohne Unterstützung für den Entscheider erfolgt, wodurch sich die Frage stellt, ob sich bei der Vergabe der Punktwerte nicht wieder systematische Fehler einschleichen können.

Der AHP (Analytic Hierarchy Process) wird in Kapitel 7.2 näher betrachtet und daher hier außer Acht gelassen.

3.7.6 Sensitivitätsanalyse

Eine Sensitivitätsanalyse sollte immer am Ende einer Entscheidungsfindung erfolgen, sowohl in der Entscheidungstheorie als auch im Operations Research. Ziel ist es herauszufinden, wie sensibel die gefundene Lösung auf Störungen in den Ausgangsdaten reagiert.

Dies ist umso wichtiger, je unsicherer diese Ausgangsdaten sind. Handelt es sich dabei um sichere Daten, so ist nicht davon auszugehen, dass die Ausgangssituation sich ändert und damit evtl. eine andere Lösung optimal wäre. In der Praxis existieren jedoch nur sehr wenige Probleme, bei denen die Entscheidungsträger eine sichere Datenbasis angeben können.

Selbst bei Problemstellungen innerhalb des eigenen Unternehmens ist es oft nicht möglich, alle benötigten Daten exakt zu ermitteln. Im hier betrachteten Einsatzfall der Komponentenauswahl ist von einer sicheren Datenbasis in der Regel nicht auszugehen. Die Eingabedaten stammen unter Umständen von Marktplätzen und damit von Dritten, die nicht immer vertrauenswürdig sind, was dazu führt, dass Spezifikationen fehlerhaft sein können. Erschwerend kommt hinzu, dass sich die Ausgangssituation an den Komponentenmärkten momentan noch sehr schnell ändert.

Aber auch die Präferenzen der Entscheider können sich ändern oder sie könnten nur unzureichend ermittelt worden sein. Beispielsweise besteht die Möglichkeit, dass Missverständnisse in den Befragungen aufgetreten sein könnten. Daher spricht alles dafür, eine kritische Haltung gegenüber den ermittelten Ergebnissen einzunehmen und eine Sensitivitätsanalyse durchzuführen, um feststellen zu können, inwiefern die Ergebnisse gegenüber Änderungen in den ermittelten Eingabedaten reagieren.

Dabei wird zum einen die (relative) Wichtigkeit der einzelnen Ziele untereinander variiert, als auch die Ordnung der Alternativen bezüglich bestimmter Ziele. Die Ergebnisse können etwa in Form von Sensitivitätsgraphen dem Entscheider präsentiert werden. Dabei kann auf der Abszisse die relative Wichtigkeit eines Ziels und auf der Ordinate die Bewertungen der Alternativen aufgetragen werden. Im Graph wird weiterhin eingetragen, wie sich die Bewertung der Alternativen bei den verschiedenen relativen Wichtigkeiten verhält. An den Schnittstellen der einzelnen Funktionen innerhalb des Graphen lässt sich daraufhin ablesen, wann sich die Präferenz für die eine oder andere Alternative ändert.

Es lässt sich also beurteilen, ob eine kleine Schwankung in den Ausgangsdaten zu einer Veränderung der Ergebnisse führt oder nicht. Nach einer ausführlichen Analyse der Sensitivität der Daten kann dann eine endgültige Entscheidung getroffen werden, womit der Prozess der Entscheidungsfindung endet.

3.8 Literaturüberblick

Im Folgenden sollen einige der Vorarbeiten zusammenfassend erwähnt werden. Ruhe (2003) gibt einen ähnlichen Überblick über die in der Literatur bekannten Arbeiten zum Themenkomplex der Einführung von Entscheidungsunterstützung in die komponentenorientierte Softwareentwicklung.

Die am häufigsten zitierte Arbeit dürfte die von Kontio (1995) sein, die in dieser Arbeit auch mehrfach zum Vergleich herangezogen wird. Der vorgeschlagene OTSO Prozess legt Schwerpunkte auf die Ermittlung eines Differenzmaßes sowie einer Zielhierarchie für den verwendeten AHP. Leider wird in keiner seiner Arbeiten die verwendete Zielhierarchie

vollständig angegeben.

Ncube hat sich ebenfalls auf dem Gebiet der entscheidungsunterstützten Komponentenauswahl betätigt. Neben dem PORE-Prozess (vgl. Ncube und Maiden (1999)), der seinen Schwerpunkt auf die iterative und gleichzeitige Ermittlung von Anforderungen und Komponenten legt, gibt es eine Erweiterung für den Bankensektor, der BANKSEC Prozess (vgl. Maiden et al. (2002)) genannt wird. BANKSEC enthält dabei ein objektorientiertes Modell der Anforderungen und der zu suchenden Komponenten, in dem der Suchprozess abgebildet wird.

Comella-Dorda et al. (2002) geben in ihrer Arbeit den PECA-Prozess an. Dieser Prozess legt seinen Schwerpunkt auf die Untersuchung von Komponenten im Bezug darauf, wie gut sie auf gegebene Anforderungen passen. Dabei wird eine ausführliche Vorgehensweise zur Komponentenevaluation vorgestellt. Besonderes Augenmerk richten die Autoren dabei auch auf organisatorische und betriebswirtschaftliche Aspekte.

Lalanda (1998) konzentriert sich in seinem Paper auf ein Kontrollmodell zur Unterstützung bei der Auswahl von Komponenten. Dabei geht es um eine Vorgehensweise, die gegebene Anforderungen an eine Anwendung systematisch in eine Architektur aus Komponenten umwandeln soll. Ein ähnlicher Ansatz wurde auch in dieser Arbeit in Kapitel 5 gewählt.

Merad und de Lemos (1999) betrachten, wie einige andere Autoren auch, die Auswahl von Komponenten aus einer Bibliothek von möglichen Alternativen. Sie wählen dabei neben dem üblichen nutzenbasierten Ansatz auf Basis der Nutzwertanalyse bzw. des AHP auch einen spieltheoretischen Ansatz.

Ochs et al. (2000) beschreiben einen Komponentenbeschaffungsprozess bei Siemens mit dem Namen CAP. Auch bei dieser Beschreibung liegt der Schwerpunkt auf der Ermittlung von Bewertungskriterien für einzelne Komponenten und die anschließende Durchführung dieser Bewertung in ablauf- und aufbauorganisatorischer Sicht. Der Prozess wird dabei in seinen Einzelschritten mit den jeweils benötigten Eingabedaten und den erzeugten Resultaten beschrieben.

4 Annahmen

Bevor geschildert wird, welche Probleme im Zusammenhang mit dem Auswahlmanagement - also der Suche nach Komponenten, der Schätzung der Entwicklungskosten und der Ermittlung des Nutzens der gefundenen Komponenten - entstehen, sollen einigen Annahmen, die für die theoretischen Überlegungen in den folgenden Kapiteln zu Grunde gelegt worden sind, vorgestellt werden. Abschließend sollen sie kritisch im Bezug auf ihre Nähe zur derzeit anzutreffenden Realität in der komponentenorientierten Anwendungsentwicklung bewertet werden.

Die folgenden Annahmen werden in den weiteren Kapiteln als erfüllt unterstellt.

(A.1) Die komponentenorientierte Anwendungsentwicklung wird als ingenieurmäßige Tätigkeit angesehen - ähnlich den Konstruktionsprozessen im Maschinenbau oder der Elektrotechnik. Dazu gehört, dass bei der Entwicklung methodisch und prozessgesteuert vorgegangen werden kann, ohne dass die notwendige Kreativität eingeschränkt wird. Weiterhin wird unterstellt, dass es Ziel der Forschung sein soll, diese Prozesse mit Werkzeugen zu unterstützen, die die Durchführung vereinfachen sollen. Diese Werkzeuge sollen dabei - wenn möglich - in Form von Softwaretools realisiert werden und die Arbeitsschritte so weit automatisieren, dass eine minimale Anzahl an manuellen Schritten verbleibt. Dabei wird an erfolgreiche Anwendungen wie das CAD (Computer Aided Design) aus dem Maschinenbau oder Software aus der Elektrotechnik, die bei der Erstellung von elektronischen Schaltungen hilft, gedacht.

(A.2) Es wird davon ausgegangen, dass die Komponentenorientierung die lange vermuteten Vorteile bringen wird und dass daher die Unternehmen bereit sind, ihre Entwicklungsprozesse auf eine komponentenorientierte Basis zu stellen. Dabei wird weiter unterstellt, dass die Unternehmen dies in einer betriebswirtschaftlichen Art und Weise vornehmen werden. Insbesondere die Erzielung von Kostenvorteilen und die Möglichkeiten zum Verkauf von Komponenten sollen genutzt werden. Weiterhin wird davon ausgegangen, dass alle Handlungen zur Erreichung dieser Ziele in sorgfältig geplanten Schritten erfolgen. Eine Führung durch das Management mit den üblichen Aufgaben Planung, Entscheidung, Durchführung und Kontrolle wird also vorausgesetzt.

(A.3) Um die benötigten Prozesse entwerfen und beschreiben zu können, ist es notwendig, zum einen eine Vorstellung von einem Komponentenmodell zu haben und zum anderen eine Beschreibungsform für die Komponenten zu besitzen, die angemessen ist, die Werkzeugunterstützung in allen Prozessschritten zu ermöglichen. Für den verbleibenden Teil dieser Arbeit wird daher, wie bereits in Kapitel 2 angedeutet, das Komponentenmodell von Shaw und Garlan (1996) angenommen. Weiterhin soll der Spezifikationsrahmen von Ackermann et al. (2002) als Beschreibungssprache für Komponenten vorausgesetzt werden, da mit ihm die Hoffnung verbunden ist, eine möglichst vollständige und umfassende Spezifikation zu besitzen, die für alle bei der Anwendungsentwicklung auftretenden Schritte ausreichende Informationen beinhaltet. Dabei sollen die hier vorgeschlagenen Erweiterungen (s. Abschnitt 3.4) ebenfalls berücksichtigt werden. Fehlerhafte Spezifikationen sollen dabei nicht auftreten, ob-

wohl sie an den entsprechenden Stellen im den folgenden Teilen der Arbeit angesprochen werden.

(A.4) Es soll weiterhin angenommen werden, dass eine ausreichende Auswahl an geeigneten Komponenten zur Verfügung steht. Diese Komponenten sollen entweder in unternehmensinternen Repositorien oder auf globalen Komponentenmarktplätzen zu finden sein. Dabei sollen diese Komponentenkataloge Suchanfragen in einem Format unterstützen, in das der Spezifikationsrahmen bzw. darauf aufbauende Suchspezifikationen übersetzt werden können. Weiterhin wird angenommen, dass die Anzahl an Komponenten in den Katalogen ausreichend ist, um die meisten der zu bewältigenden Aufgaben durch Wiederverwendung anstatt durch Eigenentwicklung lösen zu können.

(A.5) Es wird davon ausgegangen, dass alle Teile des Prozesses auf einer rationalen Basis vollzogen werden. Das bedeutet insbesondere, dass eine Vorgehensweise ausgewählt und durchgeführt wird, die einen Handlungsrahmen vorgibt. Bei zu treffenden Entscheidungen sollen die entsprechenden Verantwortlichen die Vor- und Nachteile der zur Verfügung stehenden Alternativen abwägen, sich genau über ihre Ziele im Klaren sein und auf dieser Basis eine Auswahl treffen. Ad hoc Entscheidungen sollen nach Möglichkeit durch gezielte Entscheidungsanalysen ersetzt werden.

(A.6) Beim aktuellen Stand der jungen Forschung im Bereich der Komponentenorientierung wird davon ausgegangen, dass Schwachstellen in den Vorgehensweisen und den Werkzeugen durch die Fähigkeiten erfahrener Softwareentwickler oder Konfiguratoren ausgeglichen werden können. Beispielsweise sollen die fehlenden Vorhersagemodelle (s. auch Abschnitt 6.2.8) für Konfigurationen durch den Konfigurator kompensiert werden, der dann eine Vorhersage manuell anhand seiner Erfahrungen durchführen muss.

Diese Annahmen sind streckenweise recht anspruchsvoll. Insbesondere die Annahme (A.4) ist heutzutage nicht erfüllt. Sie kann jedoch in Studien für ein eng begrenztes Anwendungsgebiet (Domain) mittels eines eigenen Repositories hergestellt werden. Von der Forderung nach einem funktionierenden Komponentenmarkt ist die Realität allerdings noch weit entfernt. Die verfügbaren Marktplätze handeln meistens Systemkomponenten, daher sind Fachkomponenten momentan kaum von externen Anbietern zu beziehen.

Dies ist auch mit der Annahme (A.2) zu begründen, die in der Realität ebenfalls noch nicht auf breiter Basis erfüllt ist. Allerdings gibt es hier erste bedeutende Bestrebungen. Auf dem WKBA 5 war beispielsweise von einem Vertreter von Microsoft Business Solutions (vormals Navision) zu hören, dass das ehemalige Navision-System auf einer komponentenorientierten Basis neu erstellt wird. Auch von SAP ist immer wieder zu hören, dass das R/3 System auf die Basis von Komponenten gestellt werden soll. Sollten diese Ankündigungen in die Realität umgesetzt werden, so ist davon auszugehen, dass in kürzester Zeit eine Reihe von Anwendungskomponenten für verschiedenste betriebliche Aufgaben zur Verfügung stehen werden.

Die Annahme (A.6) wird derzeit noch benötigt, da entsprechende Forschungsaufträge gerade erst vergeben werden. Sie ist momentan allerdings auch in der aktuellen Form nicht

einfach zu erfüllen, da nur wenig Personal mit entsprechender Ausbildung zur Verfügung steht. Außerdem sind die geforderten Vorhersagen oft selbst für erfahrene Konfiguratoren schwer zu treffen.

Die restlichen Annahmen sind dagegen einfacher zu erfüllen. Wenn nicht an die Annahme (A.1) geglaubt wird, dann wäre jeder Versuch sinnlos, Software Engineering zu betreiben. Gerade der Einsatz von Komponenten verstärkt jedoch mit seinem Wiederverwendungsgedanken die Forderungen nach einer ingenieurmäßigen Softwareentwicklung. Und die Annahme der Rationalität (A.5) ist in der Betriebswirtschaftslehre nicht unüblich.

5 Suche und Vorauswahl

In den vorangegangenen Kapiteln wurden die Grundlagen und theoretischen Annahmen behandelt, die in dem hier vorgestellten Prozess zur Unterstützung des Auswahlmanagements verwendet werden. Ein wesentlicher Bestandteil dieses Prozesses ist die Suche nach Komponenten. Dabei ist allerdings in der Literatur häufig nicht darüber nachgedacht worden, welches Ergebnis die Suche überhaupt liefern soll. Daher wird hier eine Änderung der bisher verfolgten Sichtweise angeregt und darauf aufbauend ein Suchprozess vorgestellt, der die vorgeschlagene Sicht unterstützt.

5.1 Problembestimmung

In den meisten Arbeiten, die in der Literatur zu finden sind, beispielsweise in dem am häufigsten referenzierten Werk von Kontio (1995), geht es um die Auswahl einer einzelnen (COTS) Komponente. Vergleichbare weitere Arbeiten setzen sich mit der Auswahl von Standardsoftware auseinander (z.B. Kolisch und Hempel (1996)).

An dieser Stelle soll noch einmal der generelle Unterschied zwischen komponentenorientierter Anwendungsentwicklung, so wie sie in dieser Arbeit verstanden wird, und in der Literatur auffindbaren anderen Komponentenbegriffen klargestellt werden. In dieser Arbeit wird von einer Anwendungsentwicklung ausgegangen, in der Komponenten in einer ingenieurmäßigen Art und Weise gesucht werden und mittels einer Art von Bauplan zu vollständigen Anwendungen verschaltet werden. Dabei sind die Komponenten die Grundbestandteile - im Sinne atomarer Einheiten - der Anwendungen.

In der Literatur wird häufig ein anderer Komponentenbegriff zu Grunde gelegt. Einige Autoren verstehen unter Komponenten ganze Anwendungen oder Basissysteme. In der Wirtschaftsinformatik wird dann häufig von EAI (Enterprise Application Integration) gesprochen. Dieses Teilgebiet der Wirtschaftsinformatik beschäftigt sich damit, wie einzelne Geschäftsanwendungen miteinander kommunizieren und wie die dabei auftretenden Probleme überbrückt werden können. Eine Komponente im Sinne eines EAI Szenarios besteht dabei eben genau aus einer einzelnen Anwendung. Solche Komponenten werden zwar ebenfalls von der Definition einer Komponente nach Ackermann et al. abgedeckt, sollen aber nicht im Vordergrund stehen, da der Blick hier hauptsächlich auf kleinere Einheiten gerichtet sein soll.

Durch die Verlagerung des Fokus auf die kleineren Bestandteile von Anwendungen entsteht eine neue Fragestellung, die in der Literatur nach dem aktuellen Kenntnisstand noch nicht richtig behandelt wurde. Stand bisher immer die Frage nach der Auswahl einer einzelnen Komponente im Vordergrund, so steht bei einem Wechsel der Sichtweise die Auswahl ganzer Konfigurationen im Mittelpunkt. Daher soll in dieser Arbeit dieser Wechsel der Sichtweise auch konsequent umgesetzt werden.

Eine erste Arbeit in die hier angesprochene Richtung existiert allerdings schon. Burgues et al. (2002) haben in ihrer Arbeit die Auswahl von mehreren Komponenten beim Einsatz in unterschiedlichen Organisationseinheiten eines Krankenhauses untersucht. Jedoch bezieht sich diese Arbeit auch wieder auf COTS Komponenten und damit mehr auf einen EAI basierten Ansatz. Die zentrale Erkenntnis der Autoren ist jedoch, dass Abhängigkeiten zwischen den Komponenten wichtig bei Auswahl von Komponenten sind.

Verwendet beispielsweise eine Abteilung eine bestimmt Software zur Buchung der Abrechungsdaten, dann ist es besser, in benachbarten Abteilungen die gleiche Software einzusetzen. Hier schlägt sich der Einfluss von Heterogenitäten bei der Zusammenarbeit verschiedener Anwendungen oder auch verschiedener Komponenten nieder.

Heterogenitäten können auf verschiedenen Ebenen auftreten (vgl. Abschnitt 3.3.1). Auf der syntaktischen, der semantischen und der pragmatischen Ebene kann das Zusammenspiel zwischen Komponenten gestört sein. Die Schwierigkeiten und damit auch die Kosten, die auftreten, um die einzelnen Heterogenitäten zu überwinden, nehmen dabei in der Regel in der gleichen Reihenfolge zu. Eine syntaktische Differenz zwischen zwei Komponenten kann meistens durch ein Übersetzerprogramm gelöst werden. Falls bei der Entwicklung der Komponenten jedoch ein unterschiedliches Verständnis der verwendeten Begriffen zu Grunde gelegen hat, so wird es im Allgemeinen schwer sein, diese Differenzen auszuräumen. Noch schwieriger sind verschiedene Pragmatiken zu überwinden, denn wenn die eine Komponente die Funktionalität nicht in der Art und Weise bereitstellt, wie die andere es erwartet, ist ein hoher Aufwand erforderlich, damit die Komponenten zusammenarbeiten können.

Das Problem der Heterogenitäten ist jedoch zentral bei der Suche und Auswahl mehrerer Komponenten. Heterogenitäten führen zu einem erhöhten Entwicklungsaufwand und somit zu höheren Kosten. Außerdem nimmt mit der Anzahl der in einer Konfiguration zu überwindenden Heterogenitäten das Risiko zu, das bei der Implementierung des Projekts zu tragen ist. Es besteht dann nämlich die Gefahr, dass die geplanten Adaptoren und Konnektoren nicht wie vorhergesehen arbeiten, was zwangsweise zu einer Revision der geplanten Anwendung führen muss. All dies sind weitere Gründe dafür, dass Konfigurationen betrachtet werden müssen und nicht einzelne Komponenten.

Burgues et al. (2002) haben den Einfluss der Heterogenitäten auf die Auswahl von Konfigurationen erkannt. Daher modifizieren sie bekannte Prozesse - wie beispielsweise den von Kontio (1995) - und führen eine zweite Ebene im Entscheidungsprozess ein. Auf der so genannten *lokalen Ebene*, auf der die Entscheidungen für einzelne Komponenten der jeweiligen Fachabteilungen zu fällen sind, wird eine der bislang bekannten Vorgehensweisen eingesetzt - beispielsweise der OTSO Prozess von Kontio. Zusätzlich führen die Autoren eine *globale Ebene* ein, auf der die Kontrolle über die einzelnen Auswahlprozesse übernommen wird. Dazu beobachten die Verantwortlichen der globalen Ebene die Entscheidungsfindung auf den einzelnen lokalen Ebenen, beispielsweise in Form eines Control Boards oder eines Steering Comitees. Zeichnen sich bestimmte Tendenzen einer Entscheidungsfindung ab, so werden diese in die anderen lokalen Entscheidungsfindungsprozesse integriert. Würde beispielsweise eine Abteilung eine bestimmte Datenbank eindeutig favorisieren, so würde diese Entscheidung in den anderen Abteilungen zu einer Bevorzugung von Alternativen führen, die diese Datenbank ebenfalls verwenden.

Es gibt einen weiteren Grund, der die gleichzeitige Auswahl mehrerer Komponenten sinnvoll erscheinen lässt. Entwicklungsprojekte unterliegen, wie alle Projekte im betrieblichen Umfeld, Ressourcenrestriktionen, sei es in Form von finanziellen oder personellen Mitteln. Wird auf den lokalen Ebenen jeweils isoliert voneinander eine „optimale“ Entscheidung gesucht, so muss diese Entscheidung für das gesamte System nicht ebenfalls optimal sein.

Ein Beispiel soll dies verdeutlichen. Angenommen Komponente A wird in Abteilung

A ausgewählt und benötigt das Datenbanksystem A. In Abteilung B wird Komponente B als optimal angesehen, die das Datenbanksystem B voraussetzt. Das heißt in der Konsequenz, dass zwei verschiedene Datenbanksysteme beschafft werden müssen. Die Kosten hierfür sind in der Regel viel höher als bei der Beschaffung einer einzelnen Datenbank. Eine andere Alternative wäre gewesen, in Abteilung A die Komponente C1 einzusetzen, die auf der Datenbank C aufsetzt und in Abteilung B die Komponente C2, die ebenfalls auf die Datenbank C zugreift. Obwohl lokal jeweils nicht die optimalen Komponenten ausgewählt wurden, ist jedoch aus globaler Sicht ein Optimum erreicht, da in dieser Konfiguration nur noch eine Datenbank angeschafft werden muss. Es zeigt sich also, dass die Wahl der jeweiligen lokalen Optima aus der globalen Sicht nicht immer zum Optimum führt. In Abbildung 10 ist dieser Sachverhalt noch einmal graphisch dargestellt.

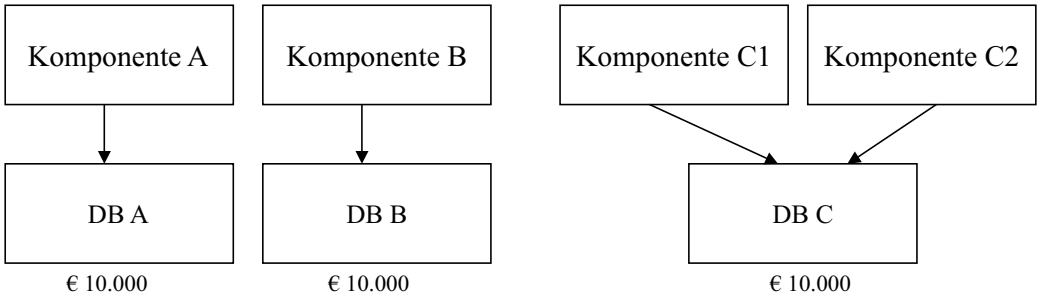

Abbildung 10: Beispiel einer abhängigen Konfiguration

Die Heterogenitätsproblematik und die Tatsache, dass lokale Optima nicht unbedingt Teil der globalen Optima sein müssen, stützt also die Annahme, dass das Ergebnis des Suchschritts Konfigurationen sein müssen. Dabei ist es nicht einzusehen, wieso ein kompliziertes Konstrukt wie die zwei Ebeneneinteilung von Burgues et al. (2002) notwendig sein soll. In der in dieser Arbeit verwendeten Sicht auf aus Komponenten aufgebauten Anwendungen gibt es keinen Grund, als Ergebnis des Suchschritts nicht direkt Konfigurationen von Komponenten zu erhalten. Diese enthalten im Gegensatz zu einer Menge an lokalen Entscheidungen sowohl die Abhängigkeiten zwischen den Komponenten als auch die globale Sicht auf die Anwendung.

Aus dieser Betrachtung folgt, dass in dieser Arbeit der Weg, Konfigurationen als Ergebnis des Suchschritts zu betrachten, eingeschlagen wird. Im nächsten Abschnitt wird daher die Frage behandelt, welche Bestandteile ein solches Ergebnis aufweisen muss.

5.2 Bestandteile einer Konfiguration

Da Konfigurationen von Komponenten als die in den AHP eingehenden Alternativen identifiziert wurden, stellt sich die Frage, welche Informationen notwendig sind, um eine Konfiguration und damit eine Alternative des Entscheidungsprozesses ausreichend zu beschreiben.

Hierzu soll eine Konfiguration, wie sie auf der Basis der Grundlagen aus Kapitel 3 erstellt wird, in einer formalen Notation aufgeschrieben werden. Eine Konfiguration besteht zumindest aus einer oder mehreren Komponenten bzw. deren Repräsentation in Form einer Komponentenspezifikation. Da eine einzelne Komponente explizit zugelassen ist, ist auch

der in der Literatur bisher betrachtete Fall abgedeckt, eine einzelne Komponente auswählen zu müssen.

Jede der Komponenten in einer Konfiguration könnte parametrisierbar sein. Daher kommen zu jeder Komponente in einer Konfiguration ihre Parameterausprägungen hinzu. Besitzt die Komponente keine Parameter, so ist dies schlicht die leere Menge.

Diese parametrisierte Komponente kann darüber hinaus mit Adaptoren versehen werden. Adaptoren sorgen dafür, dass Aufrufe von Diensten der Komponente so durchgeführt werden können wie sie benötigt werden, und sind damit ein wichtiger Bestandteil der Heterogenitätsüberwindung. Beispielsweise können Adaptoren notwendig sein, um Komponenten verschiedener Komponententechnologien miteinander verbinden zu können. Hierzu zählen unter anderem Aufrufcodes, um aus einer Microsoft.NET Anwendung heraus eine CORBA Komponente verwenden zu können, oder Übersetzer zwischen verschiedenen Datenformaten, wie bei der Umrechnung zwischen Brutto- und Nettopreisen. Von diesen Adaptoren können beliebig viele einer Komponente hinzugefügt werden.

Damit lässt sich eine Komponente, die Bestandteil einer Konfiguration ist, beschreiben als ein 3-Tupel aus Spezifikation, Parameterausprägungen und Adaptorenmenge. Bezeichne C wieder die Spezifikation und $\vec{p}$ wieder eine Parameterausprägung. Außerdem bezeichne A_C^i den i-ten Adapter der Komponente C. Dann lässt sich eine *Komponente innerhalb einer Konfiguration* $\hat{C}$ schreiben als

$$\hat{C} = (C, \vec{p}, A_C)$$
$$A_C = \left\{A_C^1, \ldots, A_C^n\right\}$$

Eine Konfiguration besteht nun aus beliebig vielen solcher Komponenten, die etwa in einer der in Abschnitt 3.6.2 beschriebenen Stücklisten abgelegt werden können. Für die Speicherung einer Konfiguration kommen noch die Konnektoren hinzu, so dass wieder die Baupläne (vgl. Abschnitt 3.6.1) als Speicherstrukturen verwendet werden können. In der oben eingeführten formalen Schreibweise lässt sich eine Konfiguration „Conf“ also wie folgt darstellen:

$$\text{Comp} = \left\{\hat{C}_1, \ldots, \hat{C}_n\right\}$$
$$\text{Con} = \left\{(\hat{C}_i, \hat{C}_j) \mid \text{falls zwischen Komponente i und j ein Konnektor ist}\right\}$$
$$\text{Conf} = (\text{Comp}, \text{Con})$$

Im nachfolgenden Beispiel soll die Notation verdeutlicht werden. Beschrieben werden soll die Konfiguration, die in Abbildung 11 zu sehen ist.

Die folgenden Komponenten treten darin auf:

- Jahresabschluss: Die Komponente, die die Resultate der Gewinn- und Verlustrechnung an die Bilanzierung übergibt und anschließend das Ergebnis darstellt.
- Gewinn- und Verlustrechnung (GuV): Erstellt eine Gewinn- und Verlustrechnung und liefert deren Saldo zurück.
- Bilanzierung: Erstellt die Bilanz aus den Bestandskonten und dem Saldo der GuV.

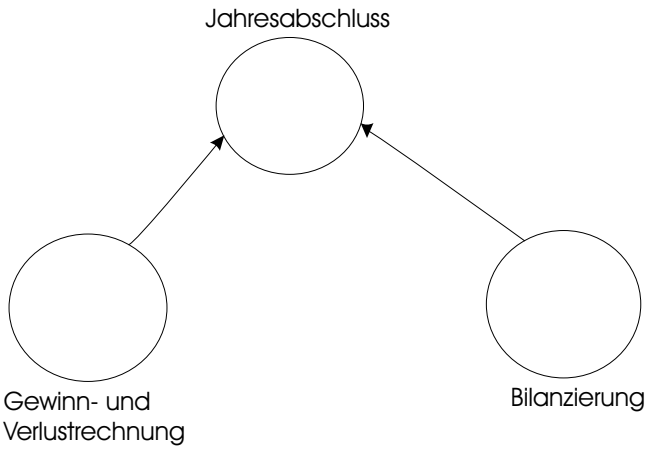

Abbildung 11: Eine einfache Konfiguration einer Buchhaltung

Die Bilanzierungskomponente arbeitet mit Binary Coded Decimals (BCD) als Datentyp, während die Gewinn- und Verlustrechnung Fließkommazahlen verwendet. Daher erhält die GuV-Komponente einen Adapter, der ihre Ausgaben auf BCD Format umsetzt. Die Jahresabschlusskomponente besitzt einen Parameter, der angibt, in welchem Dateiformat die Bilanz ausgegeben werden soll. Die Konfiguration stellt sich dann wie folgt dar:

$$\hat{C}_1 = (\text{Jahresabschluss}, \{\text{MS Word}\}, \{\})$$
$$\hat{C}_2 = (\text{GuV}, \{\}, \{\text{Fließkomma} \rightarrow \text{BCD}\})$$
$$\hat{C}_3 = (\text{Bilanzierung}, \{\}, \{\})$$
$$\text{Conf} = \left(\left\{\hat{C}_1, \hat{C}_2, \hat{C}_3\right\}, \left\{(\hat{C}_1, \hat{C}_2), (\hat{C}_1, \hat{C}_3)\right\}\right)$$

5.3 Alternativengenerierung

Es bleibt die Frage zu klären, wie die Konfigurationen generiert werden, die anschließend zur Auswahl stehen. In einer ersten, sehr groben Sichtweise besteht das Problem in diesem Schritt darin, eine Sammlung von Anforderungen (Requirements) umzusetzen in eine Konfiguration, die diese Anforderungen vollständig oder zumindest so gut wie irgendwie möglich realisiert. Während dieses Schritts müssen also anhand der Anforderungen Komponenten gefunden und zusammengesetzt werden. Das Problem ist in der Literatur zur komponentenorientierten Anwendungsentwicklung als Dekompositionsproblem bekannt. Leider gibt es bislang nur wenige Informationen darüber, wie dieser Prozess durchzuführen ist.

Dies lässt sich dadurch begründen, dass dieser Schritt nur sehr schlecht in einer ingenieurmäßigen Art und Weise durchgeführt werden kann. Vielmehr ist der Einfallsreichtum und die Kreativität des Konfigurators gefragt, der aus den vorhandenen Teilen eine vollständige Anwendung zusammensetzen muss. In der Literatur wird weitgehend die Meinung vertreten, dass aufgrund der Komplexität der Suche nach Konfigurationen ein *iterativer Prozess* der Problemstellung am Besten gerecht wird (vgl. Ncube und Maiden (1999), Alves und Castro (2001), Alves und Finkelstein (2002) und Henninger (1994)), bei dem

zwischen der Suche nach Komponenten und der Aufstellung neuer Anforderungen hin und her gewechselt wird.

In dieser Arbeit soll der Suchprozess aus Abbildung 12 vorgestellt werden, der die Suche nach Konfigurationen in ein Ablaufschema fasst. Die eigentliche Ermittlung der Konfigurationen kann dabei nicht genau vorgegeben werden, da dies der erwähnten notwendigen Kreativität beim Entwicklungsprozess widersprechen würde.

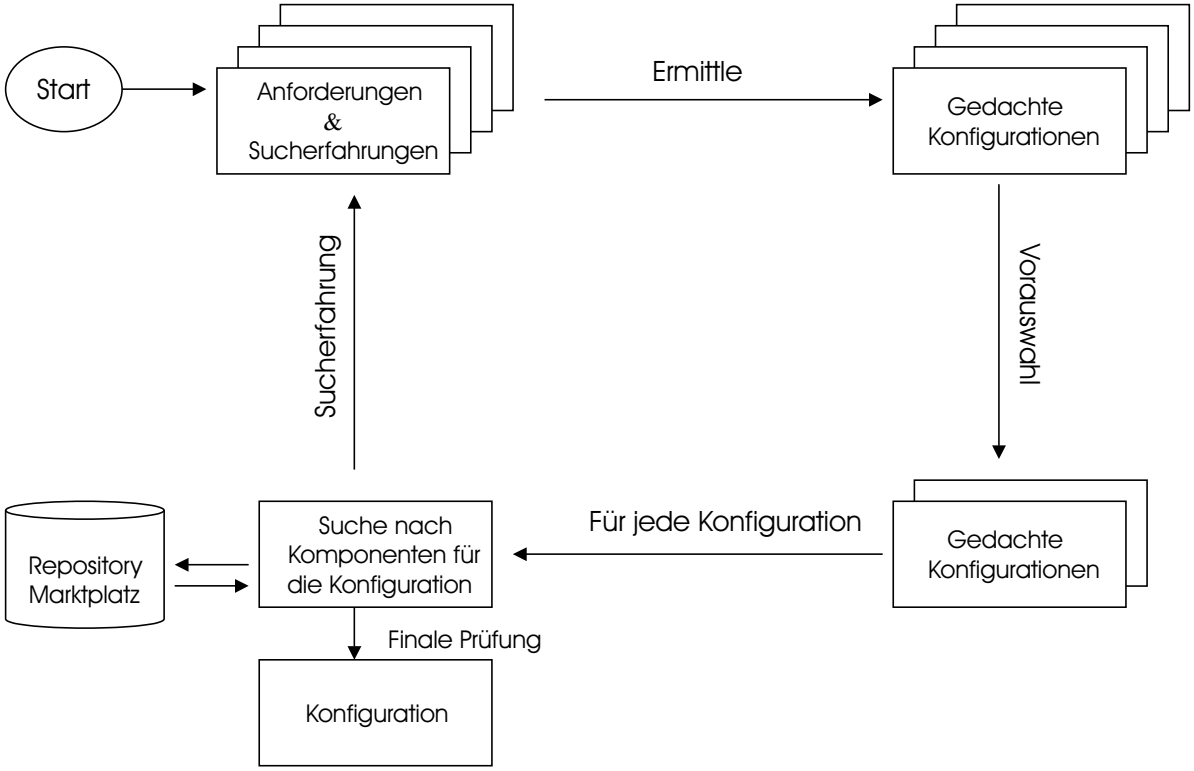

Abbildung 12: Iterativer Suchprozess

5.3.1 Experience Factory abfragen

In einem ersten Schritt, der nicht abgebildet ist, sollte die „Experience Factory“ bzw. Knowledge Base des Unternehmens zu Rate gezogen werden. Zum einen könnte es sich um eine Wartungsarbeit handeln, die es notwendig macht, vorhandene alte Baupläne, Stücklisten oder ähnliche Dokumente der zu wartenden Anwendung zu ermitteln. Zum anderen kann im Erfahrungsschatz nachgesehen werden, ob bereits ähnliche Projekte durchgeführt wurden. Falls dem so ist, können unter Umständen bereits einmal ermittelte Konfigurationen als Ideengeber herangezogen werden. Auch für die Abschätzung der Kosten der zu ermittelnden Konfigurationen sind bereits gemachte Erfahrungen wertvoll. Weiterhin können die vorhandenen Erfahrungen auch dazu beitragen, einen Ansatzpunkt für die Suche nach Komponenten zu liefern, indem Komponenten als Grundlage genommen werden, die in ähnlichen Anwendungsszenarien bereits erfolgreich eingesetzt worden sind.

Im Fall der Wartung einer Anwendung kommt hinzu, dass auch die Entscheidung abgebildet werden kann, ob eine Wartung vorgenommen werden soll oder nicht. Dazu muss nur

die Konfiguration, wie sie in der zu wartenden Anwendung realisiert ist, in die Menge der Alternativen aufgenommen werden. Kosten für diese Alternative ergeben sich dann allerdings eher über einen Opportunitätskostenansatz, in dem beispielsweise der Lebenszyklus der Anwendung oder Projektionen über die in der Zukunft gegebenenfalls ausbleibenden Supportleistungen der nicht ausgetauschten Komponente in die Kalkulation einbezogen wird.

5.3.2 Ermittlung gedachter Konfigurationen

Sind alle relevanten Informationen zusammengetragen, beginnt die erste Iteration des Suchprozesses, wie er hier vorgeschlagen wird. Aus den Anforderungen, die an die Anwendung gestellt werden, und früheren Sucherfahrungen, die aus der Experience Factory (vgl. Abschnitt 3.5) oder - in späteren Iterationen - aus bereits durchgeführten Vorgängersuchen stammen können, werden mögliche Konfigurationen erstellt, die die Anforderungen erfüllen sollen.

Hier liegt ein weiteres Problem der Entwicklung komponentenorientierter Anwendungssysteme. Es gibt keine bekannten Vorhersagemodelle, die in der Lage wären, aus einer spezifizierten Konfiguration heraus vorherzusehen, wie sich eine Anwendung, der diese Konfiguration zu Grunde liegt, später tatsächlich verhalten wird. Um prüfen zu können, ob eine Konfiguration eine Menge von an sie gestellten Anforderungen erfüllt, wäre aber genau dies notwendig. Weitere Forschung in diesem Bereich ist aktuell im Gang. Bis erste Ergebnisse vorliegen, ist es Aufgabe des Konfigurators, auf der Basis seiner Erfahrungen abzuschätzen, wie sich eine Konfiguration später verhalten wird (vgl. (A.6), Kapitel 4).

Wichtigster Bestandteil der gedachten Konfigurationen sind natürlich die Komponenten, die Teil der Konfiguration werden sollen. An sie werden bestimmte Anforderungen gestellt. In der ersten Iteration sollte sinnvollerweise eine Konfiguration erstellt werden, die genau eine Komponente beinhaltet, die all die gestellten Anforderungen erfüllt. In späteren Iterationen wird es notwendig sein, eine bereits bestehende gedachte Konfiguration weiter zu verfeinern, da die notwendigen Komponenten nicht auffindbar waren.

Diese Verfeinerung wird, wie weiter oben bereits erwähnt wurde, Dekomposition genannt. Es gibt verschiedene Varianten, eine Dekomposition vorzunehmen. Die wichtigsten beiden sind dabei funktionale und technische Dekomposition. Bei der funktionalen Dekomposition werden Komponenten anhand der zu erfüllenden Aufgaben zerteilt. Das Beispiel aus Abbildung 11 hat die Gesamtaufgabe „Jahresabschluss“ beispielsweise zerlegt in die Teilaufgaben „Bilanzierung“ und „Gewinn- und Verlustrechnung“. Technische Dekomposition sollte durchgeführt werden, wenn die fachliche Dekomposition nicht mehr möglich ist. Dabei können neue Konfigurationen gewonnen werden, indem die Basistechnologien verändert werden. Findet sich beispielsweise keine Komponente, die mit der Technologie von Microsoft zusammenarbeitet, so kann eine Konfiguration auf Java Basis ermittelt werden. Auch andere Basistechnologien können gewechselt werden. Gibt es Komponenten, die mit der Datenbank von Oracle zusammenarbeiten, aber keine, die mit Microsofts Datenbank zurecht kommen, ist es möglich, diese Technologie zu wechseln. Weitere Beispiele lassen sich problemlos angeben.

5.3.3 Vorauswahl der gedachten Konfigurationen

Sind durch Dekomposition weitere Konfigurationen entstanden, so sollte vor dem nächsten Schritt eine Vorauswahl stattfinden, da die Suche nach möglichen weiteren Konfigurationen aufgrund ihrer hohen kreativen Leistung eher in einer Art von Brainstorming durchgeführt werden sollte. Vor dem Schritt der eigentlichen Komponentensuche sollten die entstandenen Konfigurationen dabei genauer betrachtet und evtl. in eine Ordnung gefasst werden - diejenigen, die auf den ersten Blick einen guten Erfolg versprechen, werden bei der Suche nach Komponenten bevorzugt behandelt.

5.3.4 Suche nach Komponenten

Nach der Vorauswahl wird für jede der gedachten Konfigurationen eine Suche nach den in der Konfiguration enthaltenen Komponenten durchgeführt. Diese Suche findet, wie in Abschnitt 3.5 vorgestellt, auf Komponentenmarktplätzen bzw. in unternehmensinternen Komponentenrepositorien statt (vgl. (A.4), Kapitel 4).

Die Suche findet dabei anhand von „gedachten“ Komponenten statt. Diese Komponenten werden aus den Anforderungen an die Anwendung ermittelt. In Becker und Overhage (2003) wird die Verwendung einer so genannten Referenzspezifikation vorgeschlagen, die die zu suchende Komponente in den von den Repositorien verwendeten Komponentenspezifikationen beschreibt.

Allerdings ist dies heutzutage de facto nicht möglich. Eine Suche auf ComponentSource ist z.B. mehr eine Volltextsuche als eine geordnete Suche nach einer Referenzspezifikation. Es gibt allerdings auch Ansätze, die Suche nach Komponenten zu verbessern. Auf die Arbeiten von Jilani et al. (1997) und Mili et al. (1995) soll an dieser Stelle kurz eingegangen werden, da sie die grundlegenden Problemstellungen bei der Suche nach Komponenten bzw. der Formulierung entsprechender Suchanfragen behandelt haben. Eine weitere, erwähnenswerte Arbeit zu diesem Thema stammt von Ostertag et al. (1992), in der versucht wurde, den Vorgang der Suche durch künstliche Intelligenz zu verbessern.

Die Suche nach einzelnen Komponenten ist die grundlegende Tätigkeit bei der Ermittlung vollständiger Konfigurationen, da sie die elementaren Bestandteile dieser Konfigurationen sind. Komponenten sind schwer zu suchende Güter, da sie aus vielen Beschreibungsattributen bestehen können und sprachliche Probleme bei der Beschreibung an der Tagesordnung sind. Volltextsuchen haben beispielsweise sofort ein Problem, wenn in der Beschreibung der Satz „Diese Komponente tut dies und das *nicht*“ auftritt, da sie die Semantik der Negation in der Aussage nicht erfasst.

Außerdem sollte eine gute Komponentensuche ein bestimmtes Maß an Unschärfe (Fuzzy Search) ermöglichen. Werden Komponenten nach Begriffen aus einem geschlossenen oder offenen Vokabular kategorisiert, so können über diese Begriffe Verbindungen hergestellt und ausgenutzt werden. So könnte etwa Gewinn- und Verlustrechnung mit dem Begriff Jahresabschluss in Verbindung gesetzt werden, obwohl die Begriffe vom Wortstamm her nichts miteinander zu tun haben. Ein solches Netzwerk von Begriffen, die in Beziehung stehen, wird Semantic Net genannt. Die Unterstützung der Komponentensuche durch ein Semantic Net ist also wünschenswert.

5.3.5 Verwendung eines Distanzmaßes

Wichtig bei der Suche nach Komponenten ist ein so genanntes Distanzmaß (vgl. Jilani et al. (1997)), das auch bei der Suche nach den Komponenten der Konfigurationen eine wichtige Bedeutung besitzt. Das Distanzmaß gibt an, wie viele Unterschiede zwischen zwei Komponenten bzw. zwischen einer Komponente und einer Referenzspezifikation bestehen. Ein Distanzmaß von Null bedeutet dabei, dass die eine Komponente entweder ein Teil der anderen Komponente ist oder dass eine Komponente die vorgegebene Spezifikation vollständig erfüllt.

Jilani et al. (1997) unterteilen die Differenzmaße in vier Kategorien. Unter „Funktionalem Konsens“ (Functional Consensus) verstehen sie die Schnittmenge zwischen den Informationen der Suchanfrage und der Komponentenbeschreibung. Je größer diese Menge ist, umso kleiner ist das Differenzmaß. Unter der „Verbesserungsdifferenz“ (Refinement Difference) verstehen sie die Menge an Funktionalität, die einer Komponente hinzugefügt werden muss, damit sie die Anforderungen der Suchanfrage erfüllt. Je weniger hinzugefügt werden muss, umso geringer ist die Distanz. Bei der Betrachtung der „Verbesserungsdistanz“ (Refinement Distance) wird zusätzlich noch betrachtet, wie viel Funktionalität die Komponente mitbringt, die jedoch in der Suchanfrage nicht gefordert wird. Dies ist aus betriebswirtschaftlicher Sicht interessant, da auch die nicht benötigte Funktionalität natürlich über den Preis der Komponente bezahlt werden muss. Das letzte Maß, die „Funktionale Distanz“, enthält keine neue Idee mehr, sondern misst nur die zusätzliche und die fehlende Funktionalität getrennt voneinander.

Unterstützt das verwendete Repository oder der benutzte Marktplatz diese erweiterten Suchmöglichkeiten, so können die ausgegebenen Distanzmaße direkt benutzt werden, um weitere Informationen für die Konfiguration abzulesen. Gibt das Distanzmaß Auskunft darüber, wie viel Funktionalität einer Komponente im Vergleich zu der gestellten Suchanfrage fehlt, kann daraus zumindest ein erster Anhaltspunkt gewonnen werden, wie viel Aufwand in die Entwicklung von Adaptoren bzw. zusätzlicher Komponenten investiert werden muss.

5.3.6 Auswertung der Suchergebnisse

Bei der Suche nach Komponenten, die eine bestimmte (Referenz-)Spezifikation erfüllen sollen, können nun mehrere Fälle eintreten. Werden eine oder mehrere Komponenten gefunden, die die gesuchten Anforderungen erfüllen, so wird eine oder mehrere Konfigurationen generiert, die diese realen Komponente(n) zusätzlich beinhalten.

Dabei sind auch Komponenten zu berücksichtigen, die vor ihrer Anwendung parametrisiert werden müssen bzw. solche, für die entsprechende Adaptoren zu entwickeln sind. Sie werden meist durch die bei der Suche verwendete Unschärfe gefunden und durch eine manuelle Betrachtung der Suchergebnisse als Kandidaten erkannt. Ihre Parameterausprägungen bzw. die Spezifikationen der benötigten Adaptoren sind den Informationen über die gefundene Komponente hinzuzufügen.

Wird kein exakter Treffer gefunden, so können bei einer unscharfen Suche aus den Suchergebnissen wertvolle Informationen gewonnen werden. Angenommen es soll eine Jahresabschlusskomponente gesucht werden. Es werden aber nur jeweils eine GuV- und eine Bilanzierungskomponente aufgrund der erlaubten Unschärfe gefunden. Dann ist aber das Wissen über diese beiden Komponenten wichtig für die nächste Iteration, da es Hinweise

für die nächste Dekomposition in der folgenden Iteration geben kann.

5.3.7 Ausscheiden einer Konfiguration aus dem Prozess

Sind in einer gedachten Konfiguration alle Komponenten gefunden, kann die Konfiguration aus dem Prozess ausscheiden und kommt damit als potentielle Alternative für den Entscheidungsschritt in Frage. Der Konfigurator kann sich allerdings zusätzlich auch dazu entscheiden, die Konfiguration noch nicht aus dem Iterationsprozess zu entfernen, wenn er weitere, sinnvolle Dekompositionsmöglichkeiten erkennt. Beispielsweise könnte die Konfiguration zwar vollständig sein, der Konfigurator hat jedoch erkannt, dass es auch die Möglichkeit gibt, die Konfiguration auf Basis einer anderen Komponententechnologie zu implementieren. Um diese Alternative ebenfalls zu generieren, lässt er die Konfiguration im Iterationsprozess und übernimmt sie gleichzeitig in die Alternativenmenge.

Unter den Konfigurationen, die ausgewählt werden, in die Alternativenmenge aufgenommen zu werden, kann noch eine abschließende Prüfung vorgenommen werden, ob es sich wirklich zu lohnen scheint, die Alternative später genauer zu betrachten. Beispielsweise kann ein Dominanztest unter den Alternativen stattfinden. Wird die hinzukommende Konfiguration eindeutig von bereits ermittelten dominiert oder dominiert sie selbst frühere Suchergebnisse, so können die dominierten Alternativen entfernt werden.

Interessant ist in diesem Zusammenhang auch der Bandbreiteneffekt. Es erscheint sinnvoll, von den in der Vorauswahl eliminierten Alternativen zumindest die Bandbreite ihrer Attribute zu betrachten, um diese Information dem Entscheider später während der Durchführung der Paarvergleiche zur Verfügung stellen zu können.

5.3.8 Kritik

Es ist offensichtlich, dass der vorgeschlagene Suchprozess nicht alle möglichen Alternativen finden wird und von daher einen heuristischen Ansatz darstellt. Die Alternativenmenge muss jedoch auch in einem gewissen Rahmen bleiben, da ansonsten die Anzahl der Paarvergleiche im AHP nicht mehr zu bewältigen ist und die Kosten daher außer Kontrolle geraten. Diese Suche stellt aber hoffentlich eine gute Annäherung an eine umfassende Suche dar.

Bei der Durchführung der Suche sollte, wie in der Betriebswirtschaft üblich, eine Kontrolle des Suchprozesses durch den Projektverantwortlichen durchgeführt werden. So gilt es, einerseits die Generierung der Alternativen im Bezug auf ihre Anzahl zu überwachen, andererseits müssen auch die vorhandenen Ressourcen im Auge behalten werden. Die Suche kann nur solange durchgeführt werden, wie entsprechendes Geld und Zeit zur Verfügung stehen. Neigen sich diese Ressourcen dem Ende entgegen, so muss die Generierung abgebrochen werden. Das in diesem Absatz Gesagte unterscheidet sich nicht von den Aussagen in anderen Arbeiten, beispielsweise der von Kontio (1995).

5.4 Eigenentwicklung

Ein Themengebiet wurde bisher noch nicht beachtet. Es kann auch effizient sein, die benötigten Komponenten nicht von Dritten zu beziehen, sondern sie selbst herzustellen, auch wenn in der Komponentenorientierung die Wiederverwendung natürlich im Vordergrund

steht. Die in Eigenentwicklung herzustellenden Komponenten werden durch die hier angegebene Suche ebenfalls teilweise identifiziert.

Angenommen für eine gedachte Komponente existiert eine Komponente, die die Anforderungen erfüllt. Dann gibt es eine Black Box-Komponente, die geeignet ist, die geforderte Aufgabe zu übernehmen. In diesem Fall kann sich der Konfigurator die Frage stellen, ob die Eigenentwicklung einer Komponente, die die Anforderungen ebenfalls erfüllt, sinnvoll erscheint. Diese Frage hängt davon ab, wie hoch er die Kosten schätzt, die für eine Eigenentwicklung im Vergleich zur Wiederverwendung der Black-Box Komponente notwendig sind. Diese sind schwer abzuschätzen, da beispielsweise auch Kosten für die zu erwartende niedrigere Qualität der Eigenentwicklung und den daraus resultierenden Wartungsarbeiten anzusetzen sind.

In der Regel ist jedoch davon auszugehen, dass eine Eigenentwicklung immer teurer ist als eine auf dem Markt vorhandene Komponente, die die an sie gestellten Anforderungen erfüllt, da der Hersteller der Komponente seine Kosten auf mehrere Käufer verteilen kann. Auch eine Eigenentwicklung könnte einen Teil ihrer Kosten kompensieren, wenn sie sich neben ihrem Einsatz im eigenen Unternehmen auch noch verkaufen lassen würde. Jedoch kommen dann andere Probleme, wie Markteintrittsbarrieren, ins Spiel, die sich vermutlich im Voraus so gut wie gar nicht abschätzen lassen.

Findet sich jedoch keine Komponente und wird daher die Anwendung weiter zerlegt, so entstehen an den Dekompositionsstellen zu entwickelnde Komponenten, die zum einen die Teilfunktionalitäten, in die die Gesamtaufgabe zerlegt worden ist, sinnvoll koordinieren und zum anderen Funktionalitäten, die von keiner der untergeordneten Komponenten bereitgestellt werden, selbst umsetzt.

Gibt es keine weiteren Komponenten (gedachte wie auch tatsächlich existierende), die Unteraufgaben der Aufgabe erfüllen, die die zu erstellende Komponente erledigen soll, so ist an der entsprechenden Stelle eine Elementarkomponente identifiziert worden und der Zerlegungsprozess endet.

5.5 Durchführung

Im Folgenden wird vorausgesetzt, dass Stücklisten als Ergebnisstrukturen verwendet werden, um die Alternativen zu repräsentieren. In Becker und Overhage (2003) wurde die Einführung eines Entscheidungsknotens als Markerknoten für noch zu fällende Entscheidungen vorgeschlagen. Diese Knotenerweiterungen können in dem hier vorgestellten Zusammenhang weiter konkretisiert und ausgebaut werden.

Im folgenden Beispiel soll wieder eine Anwendung zur Erstellung eines Jahresabschlusses entwickelt werden. Die entsprechende Anforderungsanalyse wurde bereits durchgeführt und die Ergebnisse seien dem Konfigurator bekannt. Er steht nun vor der Aufgabe, entsprechende Konfigurationen zu generieren. Dabei könnte die in Abbildung 13 dargestellte Folge von Handlungen auftreten.

Es ist zu erkennen, dass die Knoten der Stücklistendarstellung um weitere Knotentypen erweitert wurden. Zum einen wurde zur Markierung von gedachten Konfigurationen ein Knoten eingeführt, der durch einen Kreis mit umrandendem Rechteck repräsentiert wird. Dieser Knotentyp soll eine Komponente repräsentieren, die noch nicht durch eine Suche gefunden wurde. Er steht daher für ein Gedankenkonstrukt des Konfigurators, das eine bestimmte Menge an Anforderungen an eine Komponente beinhaltet, die an der ent-

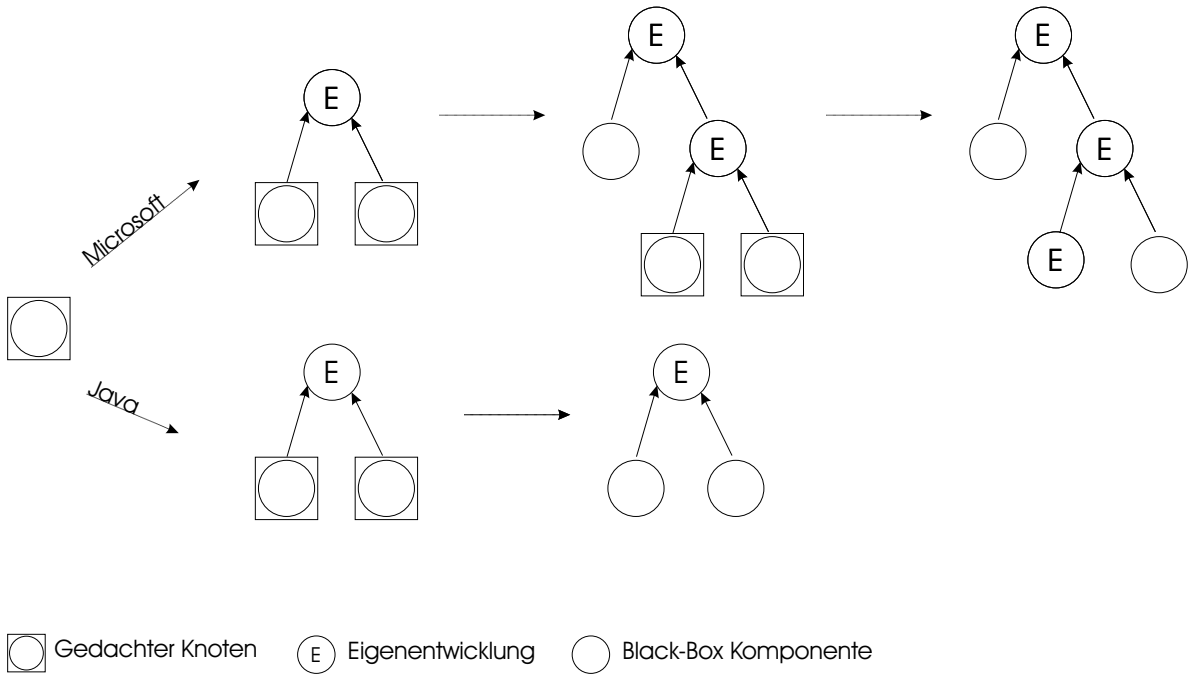

Abbildung 13: Beispiel zur Suche

sprechenden Stelle eingesetzt werden soll. Damit entspricht dieser Knoten weitgehend dem Entscheidungsknoten aus der vorangegangenen Arbeit.

Ein weiterer Knotentyp wurde eingeführt, um zwischen zugekauften, externen Komponenten und solchen, die in der eigenen Entwicklungsabteilung zu erstellen sind, unterscheiden zu können. Diejenigen Komponenten, die noch hergestellt werden müssen, sind mit einem „E“ im Knoten gekennzeichnet.

In der Abbildung ist zu erkennen, dass der Suchprozess mit einer einzelnen gedachten Komponente anfängt, die die vollständige Funktionalität der Anwendung realisieren kann, also eine Komponente, die einen Jahresabschluss gemäß der Anforderungsdefinition erstellt. Eine solche Komponente wird gesucht, jedoch erfüllt keine der gefundenen Komponenten die benötigte Funktionalität. In den Suchergebnissen finden sich allerdings Komponenten für die GuV und für die Bilanzierung.

Dies wird im nächsten Schritt zur Dekomposition benutzt. Eine weitere Dekomposition wird möglich durch die unterstützte Komponententechnologie. Es gibt einen passenden Satz von Komponenten für die Technologie von Microsoft und einen Satz an Komponenten für Java. Daher werden zwei neue Konfigurationen generiert, die die Gesamtaufgabe der Erstellung des Jahresabschlusses in die Teilaufgaben Bilanzierung und GuV-Rechnung aufteilt. Die erzeugende Konfiguration wird dabei aus der Menge der noch zu bearbeitenden Konfigurationen entfernt.

Für die beiden verbliebenen Konfigurationen wird erneut eine Suche nach den enthaltenen gedachten Komponenten durchgeführt. Dabei ist zu berücksichtigen, dass die Anforderungen an die Komponenten konkreter geworden sind. Für die obere Konfiguration kommen nur noch Komponenten von Microsoft bzw. technisch leicht in eine Microsoft Technologie adaptierbare Komponenten in Frage. Genauso sind für die untere Konfiguration nur noch

Java-kompatible Komponenten zugelassen.

Bei der Suche nach Komponenten für die Microsoft-Variante wird eine Gewinn- und Verlustrechnungs-Komponente gefunden, die die notwendigen Eigenschaften besitzt bzw. die ohne höheren Aufwand adaptierbar ist. Jedoch stellt sich heraus, dass keine geeignete Bilanzierungskomponente vorliegt, da das geforderte Ausgabeformat sehr speziell ist und daher noch kein Anbieter eine solche Komponente entwickelt hat. Für die andere Technologie werden zwei entsprechende Komponenten gefunden. Der Konfigurator entschließt sich, diese zweite Konfiguration nicht weiter zu zerlegen und hat somit eine erste Alternative zur Verwendung bei der Entscheidung gefunden.

Die letzte verbleibende Konfiguration wird erneut verfeinert. Bei der letzten Suche wurde erkannt, dass es eine Komponente gibt, die die Bestandskonten einzeln verwalten kann. Daher entschließt sich der Konfigurator, eine Komponente für das benötigte Ausgabeformat selbst herstellen zu lassen. Da er die Aufgabe nicht weiter zerlegen möchte bzw. kann, entsteht an dieser Stelle eine Elementarkomponente, die dann mit der Bestandskontenverwaltung zu einer neuen Bilanzierungskomponente vereinigt wird. Damit kann auch diese gedachte Konfiguration in eine real mögliche Konfiguration verwandelt werden, die der Alternativenmenge hinzugefügt wird.

5.6 Abschlussprüfung

Nach der Ermittlung der möglichen Konfigurationen sollte noch einmal eine intensive Prüfung aller verbliebenen Alternativen durchgeführt werden. Dabei ist noch einmal besonders zu prüfen, ob die ermittelten Konfigurationen überhaupt in der Lage sind, die von ihnen erwarteten Aufgaben zu übernehmen, und somit keine Anforderungen während des Suchvorgangs verloren gegangen sind.

In der Regel können dazu die Anforderungen eingeteilt werden in „harte“ und „weiche“. Unter „harten“ Anforderungen sollen solche verstanden werden, die für die Aufgabenerfüllung unerlässlich sind. Es handelt sich also um KO-Kriterien, ihre Nichterfüllung führt sofort zum Ausschluss der Konfiguration. Eigentlich sollte dieser Fall nicht eintreten, da in der Suche alle harten Anforderungen von Anfang an berücksichtigt werden sollen und diese nur noch verfeinert werden. Jedoch sind Fehler bei der Durchführung natürlich nie ausgeschlossen, so dass eine intensive Überprüfung an dieser Stelle unnötige Kosten in späteren Schritten ersparen kann.

Unter „weichen“ Anforderungen sollen solche Anforderungen verstanden werden, die für die Erfüllung der Kernaufgabe nicht unbedingt notwendig wären, deren Vorhandensein jedoch der Konfiguration einen Vorteil gegenüber anderen Konfigurationen verschafft, die diese Anforderungen nicht erfüllen. Da diese Anforderungen optional sind, gehen sie in Form eines Optimierungsziels in den Entscheidungsprozess ein. Das Ziel soll dabei sein, möglichst viele der zusätzlichen Anforderungen zu erfüllen - je mehr umso besser.

6 Ein Kostenmodell für Konfigurationen

Um eine Entscheidung für oder gegen eine Konfiguration zu fällen, sollten zwei Dinge bekannt sein. Zum einen der erwartete Nutzen, den sich ein Entscheider von einer bestimmten Konfiguration erhofft. Dieser Nutzen wird durch den AHP bestimmt. Zum anderen stehen einem bestimmten Nutzen aber auch bestimmte Kosten gegenüber, um diesen Nutzen zu realisieren. Das Verhältnis aus Kosten und Nutzen wird daher häufig betrachtet, wenn betriebliche Entscheidungen zu treffen sind.

Es stellt sich die Frage, wie die Kosten einer Konfiguration zur Realisierung einer komponentenorientierten Anwendung im Voraus abzuschätzen sind. Leider sind entsprechende Vorarbeiten nicht auffindbar. Gleiches gilt auch für Arbeiten über die Abschätzung von Kosten bei der Integration oder dem Einsatz einzelner Komponenten. Hier gibt es nur einige wenige Modelle für COTS Komponenten, die in der Vergangenheit entwickelt und erprobt wurden.

Die Probleme bei der Erstellung eines Kostenmodells für Softwarekomponenten liegen in der hohen Komplexität, Kreativität und den hohen Risiken bei der Entwicklung mit Komponenten. Die hohe Komplexität resultiert aus den vielen Möglichkeiten, in denen Komponenten eingesetzt werden können. Daher müssen sie auch umfangreich beschrieben werden. Die hohe Kreativität ist beispielsweise bei der Erstellung von Konfigurationen notwendig, um auch außergewöhnliche Lösungen finden zu können. Die Risiken können vielfältige Ursachen haben, angefangen bei schlechtem Herstellersupport bis hin zu ungenügenden oder fehlerhaften Komponentenspezifikationen.

Je nachdem, welche Arten von Kosten betrachtet werden sollen, können also Kostenmodelle, wie sie an dieser Stelle benötigt werden, sehr komplex werden. Da es weit über den Rahmen dieser Arbeit hinausgehen würde, ein komplexes Modell aufzustellen und in der Praxis zu erproben, soll sich dieses Kapitel darauf beschränken, die wichtigsten Kostenarten aufzulisten und Anhaltspunkte zu vermitteln, was zu schätzen ist, wenn die Kosten für eine gegebene Konfiguration zu ermitteln sind.

In Abschnitt 5.1 wurde definiert, dass in dieser Arbeit eine Konfiguration als eine Ansammlung von parametrisierbaren Komponenten sowie ihren jeweiligen Adaptoren und Konnektoren zu verstehen ist. Ein Kostenmodell sollte nun in der Lage sein, eine solche formal beschriebene Konfiguration in eine Vorhersage über die Kosten zur Realisierung dieser Konfiguration umzuwandeln. Der Fall der Anwendungswartung ist dabei bisher noch nicht berücksichtigt worden, da in diesem Fall ja bereits Teile der Konfiguration existieren. Hier wird also zusätzlich die Information benötigt, welche Teile der Konfiguration für eine Kostenabschätzung benutzt werden sollen.

Wird dieser letzte Einwand außer Acht gelassen, so ergibt sich ein Kostenmodell als Funktion einer Konfiguration auf eine Zahl, die als Kosten für die Konfiguration interpretiert werden kann. Allerdings erscheint es sinnvoller, die Einführung eines ganzen Intervalls von Zahlen vorzuschlagen. Dieses Intervall würde die Unter- und die Obergrenze der zu erwartenden Kosten angeben, womit der hohen Unsicherheit bei der Schätzung Rechnung getragen werden kann. Je höher diese Unsicherheit ausfällt, umso größer wird das Intervall zu wählen sein.

Damit ergibt sich ein Kostenmodell formal als Abbildung

$$\text{Conf} \longrightarrow [\text{min. erw. Kosten}, \text{max. erw. Kosten}]$$

Zur späteren Darstellung in einem Kosten-/Nutzendiagramm kann dabei approximativ etwa der Mittelwert zwischen oberer und unterer Intervallgrenze benutzt werden.

6.1 Kosten einer Komponente

Bevor die Kosten einer Konfiguration ermittelt werden können, müssen die Kosten für die einzelnen Komponenten der entsprechenden Konfiguration betrachtet werden, da sie einen großen Anteil an den Kosten der Konfiguration haben. Im Bereich der COTS Komponenten gibt es bereits einige Arbeiten, die in diesem Abschnitt vorgestellt werden sollen. Im folgenden Abschnitt wird auf diesen Vorarbeiten basierend ein Modell für diese Arbeit entwickelt.

Aufbauend auf den Modellen COCOMO und COCOMO II wurde das COCOTS Modell entwickelt (vgl. u.a. Abts et al. (2000), Abts und Boehm (1997)). Dabei schätzt COCOTS die Integrationskosten für COTS Komponenten in eine Anwendung ab. Unbeachtet bleiben also Kosten für die Suche nach den Komponenten wie auch die Kosten für einen vollständigen Lebenszyklus einer Softwarekomponente, während dessen weitere Kosten, beispielsweise für Wartungsarbeiten, anfallen.

COCOTS teilt die betrachteten Kosten grob in vier Teilbereiche ein, die im folgenden kurz wiedergegeben werden sollen (vgl. Abts et al. (2000)).

- *Assessment*: Diese Kostenart bündelt alle Kosten, die mit der Bewertung und Analyse der Komponenten nach dem Suchvorgang entstehen. Dabei unterscheidet COCOTS in die Analyse der Bereiche Funktionale Anforderungen, Performance Anforderungen und Nicht-Funktionale Anforderungen. Außerdem werden die Kosten noch weiter unterteilt, in solche, die durch eine oberflächliche Analyse entstehen, um gänzlich ungeeignete Kandidaten auszusondern und solche, die durch eine genauere Analyse der scheinbar geeigneten Komponenten entstehen.

- *Tailoring*: Hierunter werden alle Kosten zusammengefasst, die durch das Anpassen der Komponente an die benötigten Bedürfnisse entstehen. Darunter fallen z.B. Kosten für die Parametrisierung, die Erzeugung benötigter Datenbestände, das Anpassen von Benutzeroberflächen oder die Erstellung von Sicherheitseinstellungen.

- *Glue Code*: Unter diesem Schlagwort werden die Kosten erfasst, die zur Erstellung der Konnektoren zwischen der neu zu verbauenden Komponente und der bestehenden Anwendung entstehen. Diese Kosten sollen in dieser Arbeit jedoch nicht direkt der Komponente zugeordnet werden, sondern fallen vielmehr in einen Kostenblock, der direkt der Konfiguration zugeordnet werden soll.

- *System Volatility*: Kosten, die unter diesem Punkt geschätzt werden sollen, betreffen Kosten, die durch häufige Aktualisierungen der Komponente durch den Hersteller entstehen. Es geht also um Kosten, die von außen über die Abhängigkeit zum Hersteller ausgelöst werden können.

Wird das Modell einer Konfiguration betrachtet, wie es in dieser Arbeit verwendet wird, so ist zu erkennen, dass die Kosten für die Parametrisierung Teil der Kosten des Tailoring sind. Die Kosten für die Konnektoren sollen der Konfiguration direkt zugerechnet werden. Bleiben die Kosten für die Erstellung der Adaptoren. Diese können entweder dem Glue Code im COCOTS Modell zugerechnet werden oder in einem eigenen Punkt aufgenommen werden. Hier wird der letzte Weg gewählt, da er die Kosten transparenter aufzuschlüsseln vermag. Daher wird ein fünfter Punkt ergänzt.

- *Adapter Generation*: Diese Kostenstelle fasst alle Kosten zusammen, die im Zusammenhang mit der Generierung oder Erstellung von Adaptoren entstehen. Dabei können die Kosten je nach der zu überwindenden Heterogenitätsart gegliedert werden, also nach syntaktischer, semantischer oder pragmatischer Heterogenität.

Damit ergeben sich für diese Arbeit die Kosten einer einzelnen Komponente aus den Bestandteilen Assessment, Tailoring, System Volatility und Adapter Generation. Hinweise, wie die einzelnen Kostenstellen im Voraus geschätzt werden können, lassen sich kaum geben. COCOTS verwendet jeweils Submodelle pro eingeführter Kostenstelle, die aus zu bewertenden Attributen bestehen. Beispielsweise wäre die Anzahl der Parameter beim Tailoring ein solches Attribut. Jedes der Attribute wird auf einer abgestuften Skala zwischen sehr schwach bis sehr stark bewertet. Anhand dieser Aussagen werden Faktoren aus empirischen Studien entnommen, die dann, mit Basiskosten multipliziert, die erwarteten Kosten ergeben. Das Vorgehen gleicht damit etwas der Referenzpunktschätzung aus dem Software Engineering.

6.2 Kosten der Konfiguration

Kostenmodelle für ganze Konfigurationen sind aus der Literatur nicht bekannt. Daher muss für diese Arbeit ein eigenes Kostenmodell für die Schätzung der Kosten einer ganzen Konfiguration angedacht werden.

Wie bereits im vorangegangenen Abschnitt erläutert wurde, machen dabei offensichtlich die Kosten für die einzelnen Komponenten einer Konfiguration den größten Anteil aus. Es soll noch kurz darauf hingewiesen werden, dass im Spezialfall einer Konfiguration mit nur einer Komponente, also genau dem aus der Literatur bekannten Fall, nur die Kosten dieser Komponente betrachtet werden müssen, da die Kostenarten, die spezifisch für Konfigurationen sind, wegfallen.

In Abbildung 14 ist ein Kostenmodell für Konfigurationen grafisch dargestellt, das Grundlage für die weitere Diskussion sein soll.

In der Abbildung sind vier große Kostenblöcke abgegrenzt. Der erste Block fasst die Kosten zusammen, die direkt einer einzelnen Komponente zuzuordnen sind, der zweite Block enthält alle Kosten, die durch die Interaktion der Komponente mit ihrer Umgebung entstehen, der dritte Block enthält die Kosten, die durch die Vorhersage des Verhaltens der Konfiguration entstehen und der vierte und letzte Block enthält schließlich die Kosten, die durch die administrativen Aufgaben bei der Anwendungsentwicklung anfallen.

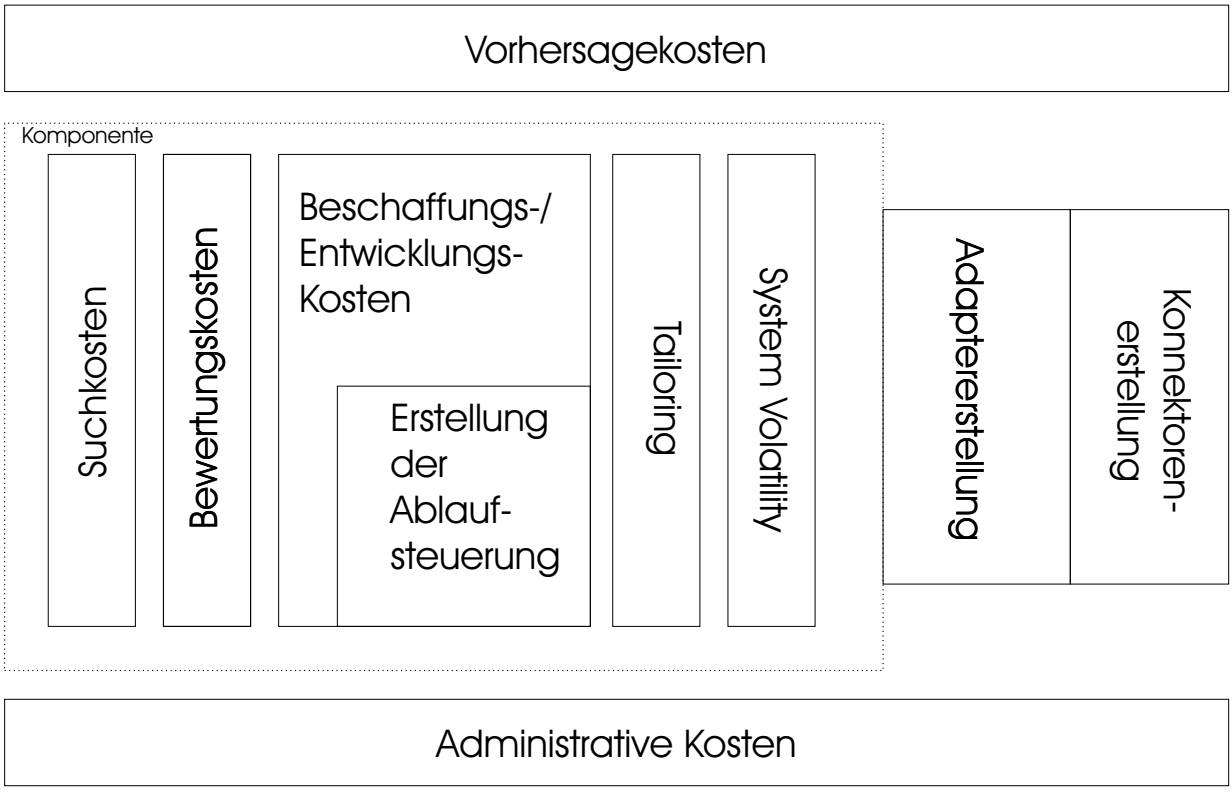

Abbildung 14: Kostenmodell für Konfigurationen - Kostenarten

6.2.1 Suchkosten

Die Kosten des ersten Blocks wurden bereits weitgehend im vorangegangenen Abschnitt erläutert. Hinzugekommen sind die Kosten für die Suche nach der Komponente an sich. Diese Kosten lassen sich beim Vorliegen der Konfiguration ex post relativ genau bestimmen. Dazu müssen während der Durchführung des Suchvorgangs für *jede* gedachte Komponente, die dann in den verfügbaren Repositorien gesucht wird, die verbrauchten Ressourcen erfasst werden. Dies kann beispielsweise in Form von Mannstunden eines oder mehrerer Entwickler geschehen. Neben diesen Ressourcen werden zwar noch Räume, Computer, etc. verwendet, die werden jedoch eher den Gemeinkosten zuzuordnen sein und daher außer Acht bleiben.

Die entstehenden Kosten aus der Benutzung des Repositories können jedoch meistens bestimmt und den Suchkosten zugerechnet werden. Auf einem Komponentenmarktplatz dürften sie sehr gering sein, da sie meist in Form einer Anmelde- bzw. Teilnahmegebühr pauschal berechnet werden. Eine Marktplatzbenutzung erhöht in der Regel eher die Transaktionskosten, wie weiter unten noch einmal erläutert wird. Bei der Nutzung eines unternehmensinternen Repositories ist die Beanspruchung der Dienste über eine interne Kostenrechnung umzulegen. Mögliche Verteilungsschlüssel ergeben sich über die Häufigkeit des Zugriffs, die Anzahl der Suchanfragen oder die Menge an entnommenen Komponenten.

Die Kosten für die Suche umfassen auch den Aufwand für nicht gefundene oder fälschlicherweise gefundene Komponenten. Diese Kosten müssen auf die gefundenen Komponenten umgelegt werden, da sie zur Suche eben dieser anschließend aufgefundenen Komponenten beigetragen haben. Die Kosten für Fehlsuchen können beispielsweise proportional auf die

Kosten für erfolgreiche Suchen aufgeschlagen werden.

In der Regel werden die gesamten Suchkosten allerdings nicht im Voraus geschätzt. Vielmehr wird bei der Erstellung eines Projektplans eine bestimmte Summe an Mitteln für den Suchschritt vorgegeben werden, die es nicht zu überschreiten gilt. Stößt der Aufwand für die Suche an diese Grenzen, sollte in der Regel die Suche abgebrochen werden. Sind bis dahin noch keine zufriedenstellenden Ergebnisse erzielt worden, so wird eine Revision des Projektplans notwendig sein.

6.2.2 Bewertungskosten

Unter die Bewertungskosten fallen in diesem Modell die Kosten, die entstehen, um die gefundenen Komponenten näher zu untersuchen. Es wird davon ausgegangen, dass in den Repositorien eine Komponentensuche nur über die Metadaten der Komponenten (in Form ihrer Spezifikationen) durchgeführt wird. Dadurch erhält der Suchende eine Beschreibung einer Komponente und evtl. die Komponente selbst in einer Art von Binärformat. Bewertungskosten entstehen nun hauptsächlich durch den Zeitaufwand, der notwendig ist, die Spezifikationen der während der Suche als erfolgversprechend erkannten Komponenten weitestgehend oder vollständig zu erfassen und zu bewerten. Insbesondere die im Suchschritt unbeachtet gebliebenen Attribute der Komponente sollten dabei einer genaueren Analyse unterzogen werden.

Die am Besten geeignet erscheinenden Komponenten können anschließend noch direkt untersucht werden, sofern eine Version im Binärformat vorliegt. Bei Komponenten, die von externen Anbietern bezogen werden können, gibt es sehr häufig die Möglichkeit, Testversionen der angebotenen Komponenten zu beziehen und somit diese anstelle der eigentlichen Komponente zu evaluieren. Dabei ist ein Test auf Spezifikationskonformität, insbesondere in den für das zu entwickelnde Projekt besonders kritischen Komponentenattributen, anzuraten. Beispiele hierfür können versprochene Leistungsmerkmale, Geschwindigkeitsangaben oder Sicherheitsmechanismen sein. Die Kosten hierfür ergeben sich unmittelbar aus den für die Evaluation benötigen Mannstunden.

6.2.3 Beschaffung

Der große Kostenblock aus Abbildung 14, der mit „Beschaffungs- bzw. Entwicklungskosten“ beschriftet ist, kennzeichnet die Kosten, die mit der tatsächlichen Bereitstellung der einsatzbereiten, physischen Komponente zu tun haben. Beschaffungskosten fallen an, wenn die Komponente von Dritten eingekauft wird, während Entwicklungskosten anfallen, wenn die Komponente vom eigenen Unternehmen hergestellt werden muss.

Unter Beschaffungskosten fallen alle Kosten, die mit der Bereitstellung der Komponente verbunden sind. Dies reicht von evtl. anfallenden Kosten für die Vertragsverhandlungen, über die eigentlichen Lizenz- bzw. Erwerbskosten, bis hin zu den Kosten, die durch die Installation der Komponente entstehen. Beim Erwerb der Komponente auf einem Komponentenmarktplatz kommen hier meist noch Provisionskosten für den Marktplatzbetreiber hinzu, die häufig anteilig auf den Verkaufspreis berechnet werden. Bei der Bestimmung der Kosten soll die gesamte Zeitspanne von der Entscheidung, diese Komponente einzusetzen, bis zu dem Moment, in dem sie erstmalig eingesetzt werden könnte, erfasst werden. Außen vor bleiben dabei die Kosten, die durch das Tailoring entstehen - diese Kosten werden in

einem eigenen Block erfasst.

Wird die Komponente dagegen im eigenen Unternehmen entwickelt, so entstehen Entwicklungskosten, die durch einen vollständig durchzuführenden Entwicklungszyklus für eine Softwarekomponente ausgelöst werden. Hierauf soll an dieser Stelle nicht weiter eingegangen werden, da dieses Thema in der Software Engineering Literatur schon häufiger behandelt worden ist und sich bei der komponentenorientierten Softwareentwicklung kaum unterscheidet.

Ein Teil der im letzten Absatz betrachteten Kosten entsteht durch die Entwicklung von Komponenten, die eine Ablaufsteuerung übernehmen, so genannten „Orchestrator Komponenten“. Die Jahresabschlusskomponente aus den vorangegangenen Beispielen ist eine solche Ablaufsteuerung, da sie die Koordination der ihr unterstellten Komponenten für die Bilanzierung und die GuV übernimmt. Solche koordinierenden Komponenten, die eine Baugruppe von Komponenten zusammenfassen, werden in nahezu jedem Projekt benötigt werden. Daher kann es durchaus sinnvoll sein, die Kosten für diese Komponenten getrennt aufzuführen, um sie von der Entwicklung von Elementarkomponenten unterscheiden zu können. In der Regel dürften die Kosten für solche „Orchestrator Komponenten“ auch niedriger sein als die für Elementarkomponenten, da sie normalerweise nur Funktionalität bündeln und nur selten eigene zur Verfügung stellen.

6.2.4 Tailoring

Nachdem die Komponenten physisch zur Verfügung gestellt worden sind, müssen sie an ihren Einsatzzweck angepasst werden. Dazu wurde in Abschnitt 3.4.1 die Verwendung von Parametern vorgesehen. Aber auch die anderen im Abschnitt 6.1 dargestellten Kosten, die unter dem Oberbegriff Tailoring gefasst worden sind, sollen hier genau wie bei COCOTS verstanden und berücksichtigt werden.

Kosten entstehen hier durch die intensive Einarbeitung in die Funktionsweise der Komponente und das Setzen der Parameter bzw. das Einrichten von Datenbeständen anhand der verfügbaren Dokumentation. Diese Kosten sind anhand der geschätzten Mannstunden für diese Arbeiten zu ermitteln.

6.2.5 System Volatility

Auch diese Kostenart ist vom COCOTS Modell übernommen. Die Vorhersagen über Kosten, die durch zukünftige Entwicklungen im Zusammenhang mit der betrachteten Komponente entstehen, sind sicher schwierig zu treffen. COCOTS versucht anhand von Einschätzungen über den Anbieter der Komponenten diese Kosten ober besser, das Risiko, dass solche Kosten entstehen können, abzuschätzen.

Bei einer Eigenentwicklung sind diese Kosten zu ersetzen, durch die Kosten aus den Gefahren, die sich dadurch ergeben, dass sich Fehler in die selbst erstellten Komponenten eingeschlichen haben könnten. Diese Fehler sind wahrscheinlicher, da sie aufgrund der fehlenden Verbreitung der Komponente schwerer zu entdecken sind, wodurch die Qualität der Komponente nicht so hoch sein könnte, wie bei einer eingekauften.

6.2.6 Adaptorenentwicklung

Bei dieser Kostenart entstehen pro Adapter pro Komponente Entwicklungskosten. Diese Kosten müssen Planung, Entwurf, Realisierung und das Testen der zu erstellenden Adaptoren umfassen. Da es sich hierbei weitgehend um Programmierung im herkömmlichen Sinn handelt, können wieder die bekannten Methoden des Software Engineering zur Schätzung der Kosten herangezogen werden.

Bei der Abschätzung gilt es insbesondere, die zu überwindende Heterogenitätsart zu berücksichtigen. Da die Problemstellung in der Reihenfolge „syntaktisch, semantisch, pragmatisch“ in der Regel schwieriger wird, sind die Kosten auch entsprechend zu bewerten. Es ist insbesondere zu prüfen, ob und falls ja, wie hoch das Risiko eines Fehlschlags bei der Erstellung der entsprechenden Adaptoren berücksichtigt werden soll.

6.2.7 Konnektorenentwicklung

Die Entwicklung von Konnektoren, auch Glue Code oder einfach nur Glue genannt, ist den Kosten für die Konfiguration direkt zuzuschreiben und nicht mehr einzelnen Komponenten, da sie mindestens zwei Komponenten verbinden und bei solchen Verbindungen oft nicht eindeutig bestimmt werden kann, zu welcher Einzelkomponente der Konnektor gehört.

Die Entwicklung eines Konnektors umfasst die Erstellung von Programmcode der die Anbindung der Komponenten realisiert und den Entwurf eines Kommunikationsprotokolls für den Austausch von Daten bzw. Nachrichten. Das Protokoll regelt dabei das Datenformat der auszutauschenden Daten sowie die zulässigen Aufrufreihenfolgen.

Kosten entstehen in diesem Zusammenhang also sowohl für Design als auch für die Realisierung der Konnektoren. Für jeden Konnektor müssen die Kosten dieser Erstellung geschätzt werden.

6.2.8 Vorhersagekosten

Diese Kostenart ist eindeutig den Kosten für Konfigurationen zuzuordnen. Die zu berücksichtigenden Kosten entstehen durch die Betrachtung der Gesamtkonfiguration unter der Fragestellung, wie sich das Komplettsystem verhalten wird, wenn es in der beschriebenen Art und Weise realisiert wird. Hierzu ist es notwendig, die ermittelten und genau untersuchten Komponenten einschließlich ihrer Parameter und ihrer Adaptoren im Zusammenspiel zu betrachten.

Beispielsweise seien alle Qualitätsattribute der einzelnen Komponenten bekannt und auch der Einfluss, den die Adaptoren auf diese Attribute ausüben, sei vorhersehbar. Dann stellt sich die Frage, welche Qualitätseigenschaften das Endprodukt eigentlich besitzen wird. Um dies zu ermitteln, ist beim aktuellen Stand der Forschung einiges an Handarbeit und Erfahrung notwendig, da entsprechende Vorhersagemodelle nicht existieren.

Daher ist es noch schwieriger, eine Kostenabschätzung für die Vorhersage der Konfiguration abzugeben. Sie hängt mindestens von den Fähigkeiten des Konfigurators und der Größe der Konfiguration ab. Hinzu kommt, dass weitere Kosten hierbei evtl. in die Kalkulation einbezogen werden müssen. So ist beispielsweise das Risiko einer Fehleinschätzung mit Strafkosten zu bewerten.

Ein anderer Ansatz wäre hier, den Nutzen einer solchen Konfiguration entsprechend zu senken, indem die Gefahr eines solchen Fehlers in die Nutzenermittlung übernommen wird.

In dieser Arbeit soll die Gefahr, eine nicht funktionsfähige Konfiguration auszuwählen, über ein Ziel in der AHP Zielhierarchie abgebildet werden (s. Abschnitt 8.1.4).

6.2.9 Administrative Kosten

Dieser Kostenblock fasst die klassischen betrieblichen Kosten zusammen, die durch das Management und den allgemeinen Betrieb des Projekts verursacht werden. Darin enthalten sind beispielsweise die Kosten, die der Projektleiter durch seine Tätigkeiten verursacht.

In der Betriebswirtschaftslehre werden diese Kosten meist über einen multiplikativen Faktor zu den anderen Kosten berücksichtigt, da diese Kosten sich im Allgemeinen proportional zu den anderen Kosten verhalten.

6.2.10 Formel

Werden alle diese Kosten zusammen genommen, so ergeben sich die Kosten der Konfiguration wie folgt.

Es sei eine Konfiguration (vgl. hierzu auch Abschnitt 5.1) gegeben als

$$\text{Conf} = (\text{Comp}, \text{Con})$$

mit

$$\hat{C} = (C, \vec{p}, A_C)$$
$$A_C = \left\{A_C^1, \ldots, A_C^n\right\}$$
$$\text{Comp} = \left\{\hat{C}_1, \ldots, \hat{C}_n\right\}$$
$$\text{Con} = \left\{(\hat{C}_i, \hat{C}_j) \mid \text{falls zwischen Komponente i und j ein Konnektor ist}\right\}$$

Und sei weiterhin

$$\text{cost} : \text{object} \longrightarrow \mathbb{R}$$

eine Abbildung eines der beschriebenen Objekte auf die ihm zugeordneten geschätzten Kosten.

Dann lassen sich die Kosten für eine einzelne Komponente des hier vorgestellten Modells schreiben als

$$\text{cost}(\hat{C}) = \text{cost}_{\text{suche}}(C) + \text{cost}_{\text{bewertung}}(C) + \text{cost}_{\text{beschaffung}}(C) + \text{cost}_{\text{volatility}}(C) + \sum_{i}^{|\vec{p}|} \text{cost}_{\text{tailoring}}(p_i) + \sum_{j}^{|A_C|} \text{cost}_{\text{adapter}}(A_C^j)$$

Und die Kosten der Konfiguration ergeben sich zu

$$cost_{\text{Conf}} = \sum_{i}^{|\text{Comp}|} \text{cost}(\hat{C}_i) + \sum_{j}^{|\text{Con}|} \text{cost}_{\text{connector}}(\text{Co}_j) + \text{cost}_{\text{vorhersage}}(\text{Conf})$$

Vorausgesetzt wird dabei, dass keines der Objekte mehrfach auftritt. Wird beispielsweise eine Komponente an mehreren Stellen einer Konfiguration eingesetzt, so entfallen, je nach Lizenzvertrag, wenigstens die Kosten für eine erneute Beschaffung. In diesen Fällen müssen die doppelt vorkommenden Einträge entsprechend wieder aus der Summe herausgerechnet werden.

7 Entscheidungsprozess

Das folgende Kapitel beschreibt die Grundlagen für den Einsatz eines entscheidungstheoretischen Verfahrens zur Bestimmung des Nutzens von Komponentenkonfigurationen. Dabei werden diese Grundlagen bereits im Hinblick auf dieses Einsatzgebiet kritisch betrachtet, um zu einer Empfehlung zu kommen, welches Verfahren in Kapitel 8 eingesetzt werden sollten.

7.1 Probleme beim Einsatz von Entscheidungsunterstützung

7.1.1 Allgemeine Probleme

Wie in Abschnitt 3.7.5 bereits erläutert wurde, ist ein Verfahren notwendig, um aus den ermittelten Alternativen diejenige auszuwählen, die am Besten zu den Präferenzen des Entscheiders passt. Hierbei stellt sich die Frage, ob die bekannten Verfahren der Entscheidungstheorie geeignet sind, die komplexe Aufgabe der Komponentenauswahl bzw. Konfigurationsauswahl geeignet zu unterstützen.

Daher werden in diesem Abschnitt die existierenden Verfahren der Entscheidungstheorie kritisch betrachtet, um deren Stärken und Schwächen bei diesen Aufgaben zu ermitteln. Hierbei kann auf eine Vorarbeit von Ncube und Dean (2002) zurückgegriffen werden, in der entscheidungstheoretische Verfahren zur Auswahl einer einzelnen COTS Komponente untersucht werden. Die Auswahl ganzer Konfigurationen haben die Autoren nicht betrachtet. Da die Auswahl einer einzelnen Komponente jedoch ein Spezialfall der Auswahl einer ganzen Konfiguration ist, ist davon auszugehen, dass die Komplexität weiter ansteigt.

Ncube führt in seiner Arbeit generelle Probleme auf, die beim Einsatz von Entscheidungsverfahren im gegebenen Kontext eine Rolle spielen können. Diejenigen, die im Rahmen dieser Arbeit am wichtigsten erscheinen, seien hier in erweiterter Form wiedergegeben. Die vollständige Aufzählung kann in Ncube und Dean (2002) nachgeschlagen werden.

- Es gibt eine sehr große Anzahl an Komponentenattributen, die in der Entscheidungsfindung zu berücksichtigen sind. Dabei sei hier nur auf den Spezifikationsrahmen nach Ackermann et al. verwiesen, der eine sehr große Anzahl an Spezifikationsattributen benutzt, um die Komponenten möglichst vollständig zu beschreiben.

- Ausgebildete Konfiguratoren gibt es bislang, wenn überhaupt, nur sehr wenige. Diese werden jedoch benötigt, da klassische Entwickler nicht auf die Auswahl von Komponenten geschult sind. Insbesondere der Einsatz von Entscheidungstheorie, um rational gerechtfertigte Entscheidungen zu treffen, gehört normalerweise nicht zu den Ausbildungszielen klassischer Informatiker. Auch für das Überprüfen einer Komponente auf ihre Konformität zu ihrer Spezifikation gibt es keine bekannten standardisierten oder gar automatisierten Prozesse.

- Im Allgemeinen existiert eine große Bandbreite in den Ausprägungen der einzelnen Attribute bei der Betrachtung verschiedener Komponenten. Insbesondere nicht quantitativ erfassbare Attribute, beispielsweise bei der Beschreibung von Benutzeroberflächen, bereiten hier besondere Schwierigkeiten.

Zu dieser Aufzählung können folgende Punkte ergänzt werden, die neben den von Ncube angeführten Punkten relevant erscheinen.

- Die meisten der in der Literatur auffindbaren Prozesse betrachten die Auswahl einer einzelnen Komponente. Dies ist jedoch eine Betrachtung, die an den Problemstellungen der Realität vorbeigeht. Dort ist es nämlich meist notwendig, ganze Konfigurationen auszuwählen, d.h. eine Kombination verschiedener Komponenten. Diese hängen natürlich voneinander ab, daher bestehen die Alternativen des Entscheidungsproblems dann in Konfigurationen von Komponenten (z.B. in Form von Stücklisten dargestellt). Die Menge an Alternativen kann dabei - je nach Generierungsmethode - so groß werden, dass der Einsatz eines Entscheidungsverfahrens mit paarweisen Vergleichen (wie dem AHP) nicht mehr praktikabel ist.

- Abhängigkeiten innerhalb der Komponentenattribute können die Grundannahmen der meisten Verfahren verletzen, wenn sie direkt als Zielgrößen verwendet werden und einzelne Komponenten ausgewählt werden sollen. Beispielsweise könnten sich Qualitätsattribute gegenseitig beeinflussen. So geht meistens eine hohe Verarbeitungsgeschwindigkeit zu Lasten des Speicherverbrauchs während der Laufzeit. Jedoch ist hier im Einzelnen zu Überprüfen, ob die Daten nur korrelieren oder ob auch die Annahme der Präferenzunabhängigkeit verletzt ist. Da letzteres nur subjektiv, mit Bezug auf den jeweiligen Entscheider ermittelt werden kann, können keine allgemein gültigen Aussagen an dieser Stelle getroffen werden.

- Ein weiteres Problem bereitet der Bandbreiteneffekt. Zu einer betrachteten Zielgröße der Alternativen gibt es eine minimale und eine maximale Ausprägung. Der Bereich zwischen diesen Extremen wird Bandbreite des Attributs genannt. Die Bandbreite hat direkten Einfluss auf die Bewertung der Alternativen (vgl. auch Eisenführ und Weber (2003, S. 142)). Es seien etwa zwei alternative Konfigurationen bekannt, die jeweils aus nur einer einzelnen Komponente bestehen. Die eine Komponente erledigt eine bestimmte Aufgabe in 10 Sekunden die andere in 20. Kommt eine dritte hinzu, die bislang unbekannt war, weil sie übersehen wurde, die die Aufgabe in einer Sekunde erledigt, ändert sich die Bewertung der beiden bislang bekannten Komponenten. Die Komponente mit einer Laufzeit von 10 Sekunden ist nicht mehr die Beste und wird abgewertet. Aus diesem Grund ergibt sich die Forderung, dass eigentlich alle für das Entscheidungsproblem relevanten Alternativen bekannt sein müssen, damit die Bandbreiten der einzelnen Attribute bekannt sind. Diese Forderung lässt sich zwar in der Theorie leicht aufstellen - sie dürfte jedoch in der Praxis nahezu nie zu realisieren sein. Daher sollte zumindest für eine optimale Annährung an das theoretische Optimum die Suche umfassend und sorgfältig durchgeführt worden sein.

Ein weiteres generelles Problem liegt in der Verwendung von additiven multiattributiven Wertfunktionen in allen für die Praxis relevanten Verfahren, die mehrere Attribute berücksichtigen. Bei additiven multiattributiven Wertfunktionen wird einer Alternativen A ein bestimmter Nutzen $N(A)$ zugeordnet, indem die einzelnen Nutzwerte der Zielgrößen gewichtet aufsummiert werden. Hierbei ist es möglich, dass besonders gute Zielausprägungen in einzelnen Zielgrößen in der Lage sind, besonders schlechte Zielausprägungen in anderen Zielgrößen zu kompensieren. Besonders brisant ist dies, wenn die vorausgesetzte

Präferenzunabhängigkeit zusätzlich verletzt ist, weil dann die Verwendung einer additiven Wertfunktion nicht mehr zulässig ist.

Hierzu soll ein Beispiel betrachtet werden. Angenommen, das Qualitätsziel einer zu bewertenden Konfiguration sei in die Unterziele Effizienz und Standardkonformität aufgeteilt. Beide Ziele sollen gleichermaßen wichtig sein, daher werden die ermittelten Nutzenwerte pro Ziel mit je 50% gewichtet. Eine Alternative A_1 sei eine durchschnittliche Komponente und erziele 0,5 Punkte in jedem der Ziele. Eine zweite Alternative A_2 sei eine Komponente mit extrem schwacher Effizienz, benutzt aber ausschließlich bekannte Standarddatenformate. Daher soll sie eine Bewertung von 0 in Effizienz und von 1 in Standardkonformität bekommen. Wird nun eine additive Wertfunktion verwendet, so ergibt sich bei beiden Komponenten ein Gesamtnutzen von 0,25, sie wären also bezüglich der Präferenzen des Entscheiders als identisch anzusehen.

In der Realität wären jedoch Zweifel an dieser Entscheidung angebracht, die daher rühren, dass die beiden betrachteten Attribute nicht präferenzunabhängig sind - zumindest nicht in den betrachteten Extremfällen. Denn wenn die Effizienz besonders schlecht ist, hilft auch eine besonders gute Standardkonformität nicht mehr viel, um den Nutzen der Komponente zu erhöhen. Am ermittelten Endergebnis ist jedoch dieses Problem nicht abzulesen. Um genau diese extremen Fälle, die häufig dazu führen, dass eine Präferenzunabhängigkeit nicht mehr gegeben ist, zu vermeiden, können Alternativen, die Attribute nicht mindestens bis zu einem bestimmten Niveau erfüllen, ausgeschlossen werden.

Die additive Wertfunktion zur Aggregation der Einzelwertfunkionen wird trotz dieser Probleme aufgrund ihrer Einfachheit in allen bekannten Verfahren verwendet. Sowohl praktische Anwendungen von MAUT, die Nutzwertanalyse oder der AHP verwenden sie in ihren theoretischen Grundlagen. Daher erben all diese Verfahren die Probleme, die mit der angesprochenen Präferenzunabhängigkeit verbunden sind. Eisenführ und Weber (2003, S. 123) geben jedoch an, dass die Herstellung der Präferenzunabhängigkeit durch Redefinition von Attributen „in praktisch jedem Anwendungsfall möglich sein dürfte“.

7.1.2 Probleme beim Einsatz von MAUT

MAUT besitzt gute theoretische Grundlagen. Die Theorie ist auf Axiomen aufgebaut, anhand derer ermittelt werden kann, ob Entscheidungen rational im Sinne von MAUT getroffen wurden oder nicht. Allerdings geht die Theorie davon aus, dass die Präferenzen der Entscheider ermittelt werden können. An dieser Stelle kann dann auch die Kritik an diesem Verfahren angesetzt werden, da bei MAUT wie auch bei gewichteten Scoring-Verfahren zusätzlich noch das Problem besteht, für einzelne Attribute Wertfunktionen bzw. Scores festlegen zu müssen. Darüber hinaus müssen zusätzlich noch die Zielgewichte in der Wertfunktion festgelegt werden.

Zur Bestimmung von Wertfunktionen kommen unter anderem die Direct-Rating Methode, die Methode der gleichen Wertdifferenzen oder die Halbierungsmethode zum Einsatz (vgl. Eisenführ und Weber (2003, S. 105ff)). Insbesondere bei der Direct Rating Methode, bei der die Nutzwerte durch direkte Punktvergabe durch den Entscheider bestimmt werden, wird offensichtlich, dass eine methodische Unterstützung nicht gegeben ist. Ohne diese Unterstützung ist jedoch auch fraglich, in wie fern das gesamte MAUT Verfahren dem Entscheider hilft.

Auch bei der Bestimmung der Zielgewichte (vgl. Eisenführ und Weber (2003, S. 125ff))

ist neben dem Trade-Off-Verfahren und dem Swing-Verfahren das Direct-Ratio-Verfahren mit seiner direkten Punktevergabe wieder vertreten. Letztlich wird also vom Entscheider nahezu immer erwartet, die benötigten Funktionen (Einzelwertfunktionen und aggregierende Wertfunktion) mehr oder weniger direkt und ohne methodische Unterstützung anzugeben.

7.2 Beschreibung des AHP

Um den in Abschnitt 7.1.2 angesprochenen Schwächen zu begegnen, wurde von Saaty (1980) der Analytic Hierarchy Process eingeführt. Vargas (1990) gibt neben einer Beschreibung des grundlegenden Ablaufs des Verfahrens auch einen umfassenden Literaturüberblick, der viele erfolgreiche Anwendungen des AHP beinhaltet, die die Beliebtheit des Verfahrens aufzeigt. Der Ablauf des AHP unterscheidet sich nicht prinzipiell von dem anderer Verfahren und soll im Folgenden detailiert vorgestellt werden. Identisch zu anderen Verfahren ist, dass auch beim Einsatz des AHP die Präferenzen bezüglich der Alternativen und Zielattribute ermittelt werden müssen. Darüber hinaus werden auch die Zielgewichte zur Aggregation der einzelnen Nutzwerte zu einem Gesamtnutzen benötigt.

Dabei wird das zu verfolgende Ziel in eine Hierarchie von Zielen aufgespalten, indem das betrachtete Gesamtziel - also hier die Auswahl einer möglichst gut geeigneten Konfiguration oder Einzelkomponente - in Unterziele aufgeteilt wird. Diese Unterziele können weiter aufgespalten werden, bis eine vollständige Hierarchie der zu berücksichtigenden Ziele erstellt ist. Aufgrund der Verwendung dieser Zielhierarchien hat der AHP übrigens seinen Namen bekommen. Ein einfaches Zielsystem zeigt Abbildung 15.

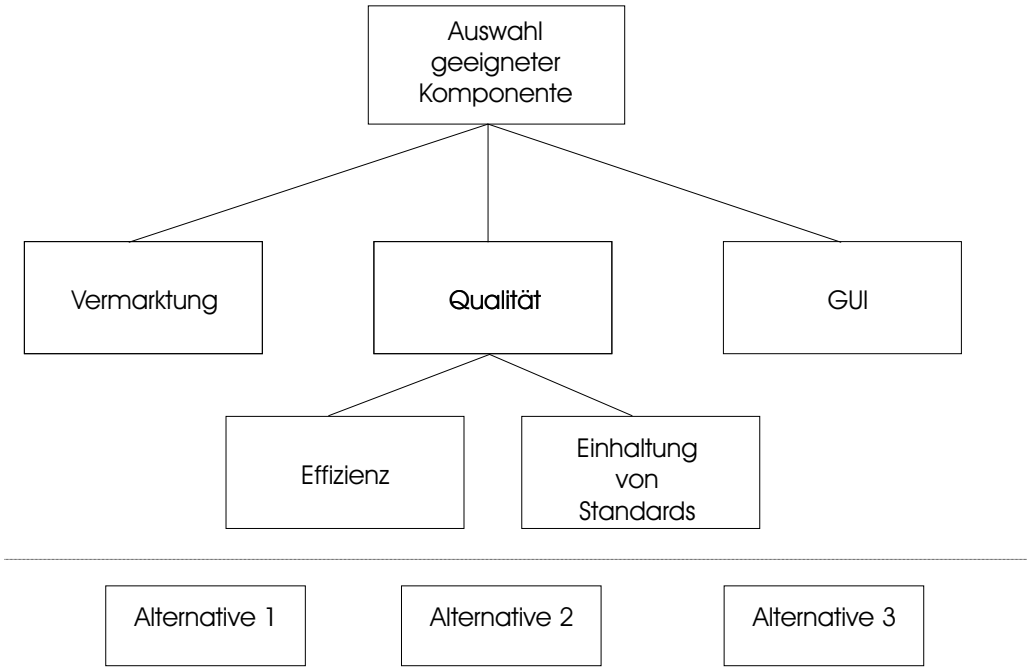

Abbildung 15: Eine einfache Zielhierarchie

Ausgehend von diesem Zielsystem werden durch geeignete Befragungen die Präferenzen des Entscheiders ermittelt, aufgrund derer sowohl die Zielgewichte als auch die Nutzwerte

ermittelt werden. Dabei wird beim AHP jedoch nicht auf holistische Aussagen zurückgegriffen, sondern die benötigten Präferenzen werden vom Entscheider durch Paarvergleiche erfragt. Die Idee hierbei ist, dass Menschen leichter in der Lage sind, Paarvergleiche durchzuführen als direkte Punktbewertungen für einzelne Zielgrößen. Oft vergleichen Texte zum AHP diese Art der Bewertung mit der Bewertung des Gewichtes von Steinen. Menschen fällt es leichter zu sagen, ob ein Stein doppelt so schwer ist als ein zweiter, anstatt das exakte Gewicht eines einzelnen Steins schätzen zu müssen.

Ein Grundprinzip des AHP ist hierbei, keinen Unterschied zwischen der Ermittlung von Artenpräferenzen und Höhenpräferenzen zu machen. Artenpräferenzen ermitteln die Wichtigkeiten der Unterziele eines Oberziels untereinander, während die Höhenpräferenzen die Präferenzen für die Alternativen bezüglich eines Elementarziels beschreiben. Zur Ermittlung der Höhenpräferenzen werden dabei die einzelnen Alternativen paarweise im Bezug auf jedes einzelne Elementarziel verglichen. Bei der Bestimmung der Artenpräferenzen werden für jede Gruppe von Unterzielen Paarvergleiche dieser Ziele vorgenommen.

7.2.1 Ermittlung der Präferenzen

Da das Verfahren für Alternativen und Unterziele äquivalent abläuft, soll es im Folgenden aus Vereinfachungsgründen nur für die Ermittlung der Artenpräferenzen dargestellt werden. Die Höhenpräferenzen werden analog ermittelt.

Zur Ermittlung der Präferenzen muss der Entscheider für jedes Paar von Unterzielen eines übergeordneten Ziels angeben, welches Unterziel für ihn wichtiger ist. Außerdem muss er das Ausmaß einschätzen, um wie viel wichtiger bzw. unwichtiger er ein Ziel gegenüber einem anderen einschätzt. Aus diesen Aussagen werden Präferenzurteile ermittelt. Dabei werden die ermittelten Präferenzurteile auf einer Verhältnisskala gemessen.

Die Verwendung einer Verhältnisskala lässt sich durch eine der Grundannahmen des AHP erklären, die lautet, dass die Zielgewichte in Form von Verhältnissen ausgedrückt werden können. Dies lässt sich unmittelbar aus der Verwendung der additiven Gesamtnutzenfunktion ableiten. Sei λ_j^h das Zielgewicht des Ziels j bezüglich des Oberziels h. Dann gibt der Quotient aus λ_j^h und λ_k^h direkt an, wie viele Nutzeneinheiten bezogen auf das Ziel k notwendig sind, um die Reduktion einer Nutzeneinheit bezüglich des Ziels j um eine Einheit zu kompensieren. Das heißt, es wird direkt das Verhältnis der Wichtigkeit der beiden Ziele untereinander bestimmt.

Bleibt die Frage, wie die Präferenzurteile auf der Verhältnisskala gemessen werden. Dies geschieht durch eine Einstufung der Präferenzaussagen der Entscheider auf einer Verbalskala. Nach Saaty (1980) werden die Präferenzaussagen dann gemäß Tabelle 2 in Skalenwerte transformiert.

Die Verbalskala dient dabei nur der Unterstützung des Entscheiders, da es Menschen in der Regel schwer fällt, das Verhältnis zweier Ziele untereinander direkt anzugeben. Daher werden die möglichen Verbalaussagen in die entsprechenden Verhältnisse umgewandelt.

Aus der Tatsache, dass Verhältnisse bestimmt werden, ergibt sich auch unmittelbar, dass bei Aussagen, deren Inhalt umgekehrt geäußert wird (z.B. „viel schlechter als“) der reziproke Skalenwert (Kehrwert des Verhältnisses) zuzuordnen ist. Der Aussage „viel schlechter als“ wäre also folglich der Skalenwert $1/7$ zuzuordnen.

Vorteilhaftigkeit	*Wichtigkeit*	*Skalenwert*
gleichwertig	gleichwichtig	1(=1:1)
etwas besser	etwas wichtiger	3(=3:1)
deutlich besser	deutlich wichtiger	5(=5:1)
viel besser	viel wichtiger	7(=7:1)
extrem viel besser	extrem viel wichtiger	9(=9:1)
Zwischenwerte		{2,4,6,8}

Tabelle 2: Verbalskala nach Saaty (1980)

7.2.2 Erstellung der Vergleichsmatrix

Aus den Aussagen des Entscheiders wird eine Vergleichsmatrix erstellt. Die dazu notwendigen Paarvergleiche können beschränkt werden durch die Erkenntnis, das ein Alternativenpaar (i, i) ein Verhältnis von 1 bekommt und dass alle Alternativenpaare (j, i) das reziproke Verhältnis desjenigen von (i, j) bekommen. Daher müssen nur Aussagen für Alternativenpaare mit $i < j$ ermittelt werden. Bezeichne $U(h)$ die Menge der Unterziele eines Ziels h, so ergibt sich die Anzahl der benötigten Paarvergleiche zu

$$|U(h)|(|U(h)| - 1)/2$$

Eine beispielhafte Vergleichsmatrix ist in der Abbildung 16 dargestellt.

	A_1	A_2	A_3
A_1	–	etwas wichtiger	etwas unwichtiger
A_2	etwas unwichtiger	–	extrem viel unwichtiger
A_3	etwas wichtiger	extrem viel wichtiger	–

$$\Rightarrow \begin{bmatrix} 1 & 3 & \frac{1}{3} \\ \frac{1}{3} & 1 & \frac{1}{9} \\ 3 & 9 & 1 \end{bmatrix}$$

Abbildung 16: Eine Vergleichsmatrix

7.2.3 Konsistenz

Die Vergleichsmatrizen besitzen eine besondere Eigenschaft, mittels derer die Konsistenz in den Aussagen des Entscheiders überprüft werden kann. Dies ist besonders wichtig, da die Aussagen von Entscheidern in der Praxis dazu neigen, Inkonsistenzen zu beinhalten.

Bezeichne v_{ij}^h einen Eintrag (ein Präferenzurteil) in einer Vergleichsmatrix V zu einem Oberziel h. Eine Vergleichsmatrix wird als konsistent bezeichnet, wenn die folgende Bedingung erfüllt ist (vgl. u.a. Saaty (1994)):

$$v_{ik}^h = v_{ij}^h * v_{jk}^h \text{ für alle } i, j, k \in U(h)$$

Die Matrix aus Abbildung 16 erfüllt diese Bedingung. Wird die Bedingung nach Aufstellen der Matrix jedoch nicht erfüllt, so gibt es die Möglichkeit, den Entscheider auf die Inkonsistenz seiner Aussagen aufmerksam zu machen. Aus der Praxis ist jedoch bekannt, dass die Entscheidungsträger häufig dazu neigen, ihre getroffenen Aussagen nicht revidieren zu wollen oder zu können. Außerdem würde das Erzwingen von konsistenten Aussagen dazu führen, dass das Vertrauen der Entscheider in das Ergebnis abnehmen würde. Der AHP ist daher in der Lage, mit solchen inkonsistenten Vergleichsmatrizen umzugehen.

Um jedoch trotzdem bestimmen zu können, welches Ausmaß die Inkonsistenz in einer Vergleichsmatrix annimmt, wurde darüber hinaus ein Maß zur Verfügung gestellt, das angibt, wie groß die Inkonsistenz einer Vergleichsmatrix ist. Wird dieser Indikator zu groß, so kann der Entscheider immer noch gebeten werden, seine Aussagen zu revidieren. Dieses Maß wird in Abschnitt 7.2.6 vorgestellt, wenn weitere mathematische Grundlagen eingeführt wurden.

In konsistenten Vergleichsmatrizen gilt darüber hinaus die lineare Abhängigkeit der Zeilen- bzw. Spaltenvektoren. Kann sichergestellt werden, dass der Entscheider ausschließlich konsistente Aussagen trifft, so genügt es, $|U(h)|-1$ Aussagen zu erfragen - beispielsweise die Einträge der ersten Zeile. Die restlichen Einträge ergeben sich aus logischen Schlußfolgerungen gemäß der Konsistenzbedingung. Abschnitt 7.3.4 geht auf diese Problematik noch einmal ein.

7.2.4 Ermittlung der Zielgewichte und Nutzwerte

Ist eine Vergleichsmatrix aufgestellt, so müssen aus dieser die Zielgewichte bzw. die Nutzwerte ermittelt werden. Sei $\vec{w}^h$ der Vektor der zu ermittelten Werte bezüglich des Ziels h (Der Superskript wird zur Vereinfachung im Folgenden vernachlässigt, wenn ausschließlich ein festes Ziel h betrachtet wird). Ist h ein Elementarziel, so ergeben sich Nutzwerte, andernfalls lassen sich Zielgewichte ablesen. Im Folgenden wird dieser Unterschied wie bereits im vorherigen Abschnitt vernachlässigt.

Die zu bestimmenden Einträge des Vektors $\vec{w}$ müssen so gewählt werden, dass sie möglichst gut die durch die Präferenzaussagen erfragte Verhältnisskala abdecken. Das heißt, dass für zwei Einträge des Vektors $\vec{w}$ gelten muss:

$$v_{ij} \approx \frac{w_i}{w_j}$$

Außerdem müssen die ermittelten Zielgewichte $\vec{w}$ auf allen Ebenen der Zielhierarchie sowie die Nutzwerte untereinander vergleichbar bleiben. Hierzu nimmt der AHP eine Summennormierung der Werte w_i auf eins vor. Das heißt, die Werte werden so gewählt, dass gilt:

$$\sum_{i=1}^{|U(h)|} w_i = 1$$

Diese Normierung bringt aber auch Probleme mit sich, da bei der Aufnahme einer weiteren Alternative die Normierung erneut durchgeführt werden müsste, was dazu führen würde, dass die bereits bekannten Alternativen schlechtere Bewertungen erhalten. Auf diese Problematik geht Abschnitt 7.3.1 näher ein.

Ist die Vergleichsmatrix konsistent, so kann die Bestimmung der v_{ij} exakt geschehen. Dazu kann ein beliebiger Spaltenvektor der Matrix ausgewählt werden, zu dem dann die Summe der Elemente gebildet wird. Mittels dieser Summe wird dann der Spaltenvektor durch einfache Division auf eins summennormiert. Der erhaltene Vektor wird als Vektor $\vec{w}$ benutzt. Dieser Vektor ist unabhängig von der ausgewählten Matrixspalte, da die Spalten bei einer konsistenten Matrix gerade linear abhängig sind, was durch eine einfache Überlegung überprüft werden kann.

In der Beispielmatrix aus Abbildung 16 ergibt sich für den zweiten Spaltenvektor eine Summe von 13 für die Einträge des Vektors. Daher ergeben sich für die Elemente des Vektors $\vec{w}$ die Einträge $(3/13, 1/13, 9/13)$.

7.2.5 Näherungsverfahren

In der Regel ist die Vergleichsmatrix nicht konsistent. Dann ist es notwendig, ein Näherungsverfahren einzusetzen, das die Beziehung $v_{ij} \approx \frac{w_i}{w_j}$ möglichst gut erfüllt. Dazu zählen einfache Verfahren, die mit der Bildung von Durchschnitten arbeiten, das in der Literatur weit verbreitete Verfahren der Bestimmung des maximalen Eigenvektors der Vergleichsmatrix oder Verfahren des Goal Programming.

Bei der bekanntesten Methode der Ermittlung desjenigen Eigenvektors der Vergleichsmatrix V, der zu dem *größten Eigenwert* μ_{max} gehört, wird als Annährung an den Vektor $\vec{w}$ eben dieser Eigenvektor gewählt. Für eine mathematische Begründung dieser Vorgehensweise siehe Saaty (1980).

Zur Bestimmung des Vektors $\vec{w}$ ist wie folgt vorzugehen. Zuerst ist die so genannte charakteristische Gleichung

$$\det(V - I\mu) = 0$$

aufzustellen und die Nullstellen im Bezug auf μ zu lösen. Mit dem größten Eigenwert μ_{max} kann der zugehörige Eigenvektor $x(\mu_{max})$ durch Lösen der Gleichung

$$(V - \mu_{max} I)x(\mu_{max}) = 0$$

bestimmt werden. Der Wert des gesuchten Vektors $\vec{w}$ wird dann gleich $x(\mu_{max})$ gesetzt.

Bei der Nutzung von *Goal Programming* zur Bestimmung des Vektors $\vec{w}$ wird ein lineares Gleichungssystem aufgestellt und gelöst. Die Zielfunktion enthält dabei die Summe aller Abweichungen in der Gleichung $v_{ij} \approx \frac{w_i}{w_j}$ vom Idealzustand, in dem die Gleichheit der beiden Ausdrücke gegeben wäre. Diese Abstände sollen in ihrer Summe minimal werden. Dies geschieht durch die Einführung von Schlupfvariablen für positive bzw. negative Abweichungen von der Gleichheit in dieser Gleichung. Zusätzlich wird als Nebenbedingung die notwendige Summennormierung der w_i in das Optimierungsproblem aufgenommen. Die Elemente des Vektors $\vec{w}$ sowie die Schlupfvariablenwerte sind aus der Lösung des Gleichungssystems abzulesen, die in der Vergleichsmatrix erfassten Präferenzurteile v_{ij} gehen als Konstanten über die Näherungsgleichung in die Berechnungen ein.

7.2.6 Inkonsistenzindex

Wird die Eigenvektormethode verwendet, so ist es darüber hinaus möglich, zu einer Vergleichsmatrix V einen *Inkonsistenzindex* zu ermitteln. Aus mathematischen Überlegungen

heraus ergibt sich, dass der größte Eigenvektor der Matrix V μ_{max} größer oder gleich ihrem Rang $|U(h)|$ ist (vgl. Saaty (1980)). Daher kann der Abstand zwischen dem größten Eigenvektor und dem Rang als Ansatzpunkt eines Inkonstistenzindizes genommen werden, da dieser mit zunehmender Inkonsistenz größer wird.

Der Inkonsistenzindex $IK(V) = (\mu_{max} - |U(h)|)/(|U(h)| - 1)$ kann hierzu eingesetzt werden, wobei der Nenner nur zur Normierung dient. Allerdings kann dieser Index nicht ohne weitere Überlegungen verwendet werden, wenn es um die Frage geht, ob der Entscheider eine Vergleichsmatrix überarbeiten sollte oder nicht.

Dies liegt an der Tatsache, dass der Inkonsistenzindex auch von der Größe der Vergleichsmatrix abhängt. Mit zunehmender Anzahl der Einträge in dieser Matrix lassen sich Verletzungen der Konsistenzbedingung immer schwerer vermeiden. Daher hat Saaty (1980) in einer Untersuchung ermittelt, welche Inkonsistenz Matrizen haben, die in Zufallssimulationen ermittelt wurden. Von 500 dieser zufällig bestimmten Matrizen hat Saaty die durchschnittlichen Inkonsistenzindizes bestimmt und diesem Wert den Namen Zufallsindex gegeben ($ZI(|U(h)|)$. Aus dem praktischen Einsatz des AHP hat sich eine Überarbeitung der Vergleichsmatrix als sinnvoll erwiesen, falls gilt:

$$\frac{IK(V)}{ZI(|U(h)|)} > 0.1$$

Nachdem alle Vergleichsmatrizen so bestimmt wurden, dass ihre Inkonsistenz innerhalb der tolerierbaren Grenze liegen und nach der anschließenden Ermittlung aller Zielgewichte und Nutzwerte ist es nur noch notwendig, die aggregierten Nutzwerte der betrachteten Alternativen zu bestimmen. Dazu werden, angefangen bei den Elementarzielen der Zielhierarchie, die Nutzwerte gemäß der unterstellten additiven Wertfunktion bestimmt. Am oberen Ende der Hierarchie kann dann die gesuchte Reihenfolge der Alternativen abgelesen werden.

Um die Verwendung des AHP als Werkzeug der Entscheidungsunterstützung näher kennen zu lernen, kann Standardsoftware eingesetzt werden. ExpertChoice (2003) dürfte das am meisten benutzte Programm sein, dass den AHP verwendet. Auf der Homepage kann eine Testversion heruntergeladen werden. Außerdem finden sich dort viele Anwendungsbeispiele des AHP.

Mustajoki und Hämäläinen (2002) haben ein kleines Java-basiertes Webapplet entwickelt, das den AHP verwenden kann. Einfache Zielhierarchien können hiermit ohne Installationsaufwand ausprobiert werden.

Es soll nicht unerwähnt bleiben, dass die Untersuchung von Ncube und Dean (2002) zu dem Schluss kommt, dass eigentlich keines der Verfahren der Entscheidungstheorie für den Einsatz zur Auswahl von Komponenten geeignet ist. Gegen den AHP spricht gemäß den beiden Autoren, dass keine Unabhängigkeit in den Attributen der Komponentenbeschreibungen besteht, die beim AHP wegen der additiven Wertfunktion vorausgesetzt wird.

Weitere Bedenken werden wegen der hohen Anzahl an Paarvergleichen gegeben, die durch eine große Anzahl an Alternativen bzw. eine hohe Anzahl an Elementarzielen entstehen können. Allerdings gehen Ncube und Dean (2002) auch davon aus, dass einzelne Komponenten auszuwählen sind. Die Auswahl ganzer Konfigurationen wird nicht betrachtet.

Der AHP wird in dieser Arbeit trotzdem verwendet, da auch Ncube bislang nicht in der Lage ist, ein besseres Verfahren anzugeben. Der AHP hat dagegen durch seinen erfolgreichen langjährigen Einsatz und die methodische Unterstützung des Entscheiders gezeigt, dass er zur Entscheidungsunterstützung benutzt werden kann. Zum Beispiel verwendet auch die Gartner Group (s. GartnerGroup (2003)) den AHP in ihren Werkzeugen zur Entscheidungsunterstützung.

7.3 Untersuchung des AHP

Der AHP ist in der Literatur zur Entscheidungstheorie häufig insbesondere von den Verfechtern der multiattributiven Nutzentheorie (MAUT) angegriffen worden, da er die axiomatischen Grundsätze dieser Nutzentheorie verletzt. Einige der wichtigen Kritikpunkte sollen hier aufgegriffen werden, wobei jeweils analysiert werden soll, ob im Anwendungsgebiet der Auswahl von Konfigurationen komponentenorientierter Anwendungen Gegenmaßnahmen gegen die im AHP verankerten Probleme notwendig sind.

7.3.1 Rangumkehr

Ein in der Literatur viel beachtetes Problem bei der Anwendung des AHP ist die Umkehr der Präferenzordnung bezüglich der Alternativen, wenn zusätzliche Alternativen betrachtet werden. Zuerst soll analysiert werden, aus welchen Gründen sich die Reihung der Alternativen ändern kann. Anschließend wird kritisch beleuchtet, welchen Einfluss dies auf die zu treffende Entscheidung hat.

In der Nutzentheorie wird die Anforderung an rationale Entscheidungen gestellt, dass die Aufnahme irrelevanter Alternativen nicht zu einer Änderung der Reihenfolge in der Bewertung der Alternativen führt. Insbesondere die Aufnahme einer exakten Kopie einer der Alternativen sollte nicht dazu führen, dass sich das Ergebnis des Entscheidungsverfahrens ändert. Dies ist beim AHP jedoch nicht gewährleistet.

In der Literatur finden sich einige Beispiele zu dem genannten Sachverhalt (vgl. u.a. Forman und Gass (2001, Abschnitt 8.3), Millet und Saaty (2000), Schenkerman (1994) und Vargas (1994) sowie Saaty (1994)). Es stellt sich die Frage, wieso sich die Ordnung der Alternativen umkehrt. Dies hat seine Ursache in der Tatsache, dass der AHP die Nutzwerte anhand einer Verteilung vergibt, die auf eins summennormiert wird. Das heißt konkret, dass die für ein Oberziel zur Verfügung stehenden 100% gemäß der Präferenzaussagen auf die Menge der Unterziele bzw. Alternativen aufgeteilt wird.

Wird eine neue Alternative hinzugefügt, so müssen, falls die neue Alternative eine Bewertung größer Null erhalten soll, von anderen Alternativen Nutzwerte abgezogen werden, die auf die hinzugefügte Alternative übergehen. Dadurch ändern sich also die Nutzwerte der bislang bekannten Alternativen. Diese Änderung kann sich so stark auswirken, dass sich die Reihenfolge der Alternativen im Ergebnis verändert - eine Rangumkehr findet statt.

Forman und Gass (2001) stellen fest, dass es offenbar zwei Varianten gibt, wie die Nutzwerte an den Elementarzielen auf die einzelnen Alternativen verteilt werden können. Sie unterscheiden dabei in geschlossene und offene Systeme. Bei geschlossenen Systemen erfordert die Einführung neuer Alternativen die Wegnahme von bereits vergebenen Nutzwerten, um sie auf die neue Alternative zu verteilen, eine Rangumkehr ist möglich. Geschlossen heißt das System dabei deshalb, weil bei der Verteilung der Nutzwerte eben nur eine durch

die Summennormierung begrenzte Anzahl von Punkten vergeben werden können. Oder anders ausgedrückt stellen die Nutzwerte eine begrenzte Ressource dar. Bei offenen Systemen ändert die Einführung weiterer Alternativen nichts an der Rangfolge bereits bekannter Möglichkeiten, da die Verteilung weiterer Nutzwerte auf zusätzliche Alternativen unabhängig von den bisherigen Nutzwerten der Alternativen erfolgen kann. Die Ermittlung der Nutzwerte muss allerdings für dieses Vorgehen entsprechend angepasst werden.

Welche Art der Ermittlung der Nutzwerte angewendet wird, hängt also von der Art des Systems ab. Bei der Auswahl von Konfigurationen aus Komponenten erscheint der Ansatz eines offenen Systems sinnvoller. Es muss die Frage beantwortet werden, ob das Bekanntwerden einer weiteren, bislang unbekannten Alternativkonfiguration Einfluss auf die Reihenfolge der bekannten Konfigurationen hat und dort zu einer Umkehr der Reihenfolge führen sollte oder nicht. Dies würde aber bedeuten, dass es Elementarziele gibt, deren Bewertungen in einem gewissen Sinne begrenzte Ressourcen im Bezug auf die alternativen Konfigurationen darstellen.

Eigenschaften von Konfigurationen erfüllen dieses Kriterium wohl kaum. Die Art und Anzahl der Alternativen sollten eigentlich keinen Einfluss darauf besitzen, wie die Konfigurationen zu bewerten sind. Daher wird die Bewertung der Alternativen als offenes System empfohlen.

7.3.2 Skala

Die Bewertungsskala beim AHP wurde durch Saaty mehr oder weniger willkürlich gewählt. Die Einteilung in 9 verschiedene Klassen von Präferenzaussagen ist viel mehr aus der Praxis entstanden, da Menschen mittels ihres kognitiven Aufnahmevermögens eher in der Lage sind, kleine Skalen zu erfassen. Auch eine andere vorgeschlagene Skala orientiert sich an Schulnoten und hat daher nur 15 verschiedene Abstufungen.

Es stellt sich die Frage, ob die gewählte Skaleneinteilung gerechtfertigt ist bzw. welche Probleme bei ihrer Verwendung auftreten können. Um dies zu erläutern, sollen hier drei Alternativen A, B und C betrachtet werden. B wird vom Entscheider als deutlich besser beurteilt als A und C wird ebenfalls als deutlich besser beurteilt als B. Damit ergibt sich ein Verhältnis von 1:5 zwischen A und B, sowie ein Verhältnis von 1:5 zwischen B und C. Rein mathematisch ergibt sich damit ein Verhältnis von 1:25 zwischen A und C, falls eine konsistente Bewertung unterstellt wird. Dieses Verhältnis ist jedoch auf der Verbalskala nicht vorhanden.

Aufgrund dieser Ausführungen ist zu erkennen, dass zu fordern ist, dass die Alternativenmenge aus Alternativen bestehen muss, die sich nicht zu stark voneinander unterscheiden. Denn wenn die Alternativen zu unterschiedlich sind, ist die Verbalskala bei der Bewertung der Alternativen nicht ausreichend und die Inkonsistenz in der Vergleichsmatrix wird unvermeidlich vergrößert.

Wie stark sich die Konfigurationen von komponentenorientierten betrieblichen Anwendungen unterscheiden, lässt sich schwer vorhersehen. Es könnte jedoch der Fall eintreten, dass Konfigurationen sehr stark von der zugrundeliegenden Komponententechnologie, dem verwendeten Betriebssystem oder anderen benötigten Basissystemen abhängen.

Ein Beispiel soll dies kurz illustrieren. Für eine zu erstellende Anwendung sollen Konfigurationen ermittelt werden. Es finden sich zwei Konfigurationen auf der Basis von Microsofts .NET Framework und drei auf der Basis von Suns Java. Ob diese Konfigurationen direkt

vergleichbar sind, ist mehr als fraglich, da die Erfahrung zeigt, dass einige Schlüsseldesignentscheidungen von dem zur Verfügung stehenden Komponentenframework abhängig sind. Es kann daher bezweifelt werden, dass die resultierenden Konfigurationen beim Vergleich zwischen den Frameworks ähnlich sind.

Dies ist eine Vermutung über den Regelfall. Dass es Ausnahmen geben wird, in denen die verschiedenen Konfigurationen ähnlich sein werden, soll nicht bestritten werden. Dies muss im Einzelfall überprüft und entsprechend berücksichtigt werden.

Falls die Alternativenmenge zu unterschiedlich sein sollte, kann sie eventuell in Cluster aufgeteilt werden (vgl. Saaty (1994)). Dabei würden beispielsweise alle auf der Microsoft Technologie basierenden Konfigurationen in einen Cluster sortiert werden und alle Java basierten in einen anderen. Dann wird mit jedem der beiden Cluster das Entscheidungsverfahren einzeln durchgeführt. Anschließend wird ein dritter Entscheidungsprozess durchgeführt, dessen Alternativenmenge aus der jeweils besten Alternative aus jedem der Cluster besteht.

Finan und Hurley (1999) stellen noch weitere Probleme mit der von Saaty eingeführten Verbalskala dar. So geben sie beispielsweise eine Untersuchung an, in der die Aussagen der Entscheidungsträger mit den Aussagen, die aufgrund der Konsistenzbedingung notwendig gewesen wären, nicht übereinstimmten. Da es sich hierbei jedoch eher um Verfahrensfragen des AHP handelt, sei auf die angegebene Quelle verwiesen. Im konkreten Fall ist zu prüfen, ob Unternehmen, die die verschiedenen Varianten des AHP in ihrer (Standard-)Software umgesetzt haben, entsprechende Möglichkeiten zur Skalenanpassung vorgesehen haben.

7.3.3 Anzahl der Paarvergleiche

Die Anzahl der Paarvergleiche, die zur Erstellung einer vollständigen Vergleichsmatrix notwendig sind, kann einfach errechnet werden. Es werden so viele Paarvergleiche benötigt, dass die obere Dreiecksmatrix einer Vergleichsmatrix ausgefüllt werden kann. Für jeden Eintrag muss der Entscheider einmal befragt werden. Daher sind $|U(h)|(|U(h)|-1)/2$ solcher Vergleiche erforderlich.

Die Anzahl der Vergleiche nimmt also quadratisch mit der Anzahl der Alternativen bzw. der Anzahl der Unterziele eines Oberziels zu. Es stellt sich also die Frage, welche Mengen an Alternativen und Unterzielen praktikabel sind.

Es kann die Frage gestellt werden, wieso eine hohe Anzahl von Paarvergleichen eigentlich schlecht ist. Aus der Praxis ist bekannt, dass eine hohe Anzahl an Paarvergleichen ein Risikopotential für das Entscheidungsverfahren darstellt. Sie führen logischerweise dazu, dass der Entscheider mehr Präferenzaussagen zu treffen hat. Eine Befragung nach Präferenzen über einen längeren Zeitraum hinweg, führt zu einigen psychologischen Problemen. Der Entscheider wird ermüden, an Konzentrationsmangel leiden oder vielleicht häufiger gleiche Antworten geben, weil er nicht mehr intensiv genug nachdenkt. Kontio (1996) hat in seiner Fallstudie solche Erfahrungen machen können. Durch eingeplante und ausreichende Pausen wurde in der Studie versucht, das Problem zu kompensieren.

Ein anderes Problem ergibt sich sowohl aus betriebswirtschaftlicher Sicht als auch aus Sicht der Entscheidungstheorie durch die steigenden Kosten, da ja Personal für einen bestimmten Zeitraum benötigt wird. Steigende Kosten verteuern das Endprodukt und lassen das Vorgehen nicht mehr als rational erscheinen, wenn die Kosten für eine Auswahlheuristik

- wie den AHP - deutlich über dem zu erwartenden Nutzen liegen.

Paarvergleiche treten bei der Ermittlung der Höhen- und Artenpräferenzen auf. Werden im Zusammenhang mit der Ermittlung von Artenpräferenzen Unterziele betrachtet, gibt es eventuell die Möglichkeit, durch eine andere Aufstellung einer Zielhierarchie Paarvergleiche einzusparen. Ein Rechenbeispiel soll dies aufzeigen. Angenommen es gäbe ein Oberziel mit 10 Unterzielen. Dann wären $10 * 9/2 = 45$ Paarvergleiche notwendig. Ist es möglich, eine weitere Hierarchieebene bestehend aus 2 Zielen mit jeweils 5 Unterzielen dazwischen einzufügen, so reduzieren sich die Paarvergleiche auf $2 * 5 * 4/2 + 1 = 21$. Die eingezogene Zwischenebene muss dabei allerdings auch entsprechend sinnvoll sein. Das heißt, dass jedes eingeführte neue Ziel auch die ihm untergeordneten Ziele wirklich aggregieren muss.

Zur Einsparung von Paarvergleichen an den Elementarzielen der Hierarchie, also an den Stellen, an denen die Mächtigkeit der Alternativenmenge eine Rolle spielt, kann nur die Alternativenmenge entsprechend in ihrer Größe beschränkt werden. Forman und Selly (2001) schlagen für den Fall zu großer Alternativenmengen die Einführung von Clustern vor, die mittels Cluster Linking verbunden werden.

Es stellt sich die Frage, ob große Alternativenmengen im betrachteten Einsatzfall eigentlich zu erwarten sind. Konfigurationen, die eine Anwendung gemäß eines bestimmten Fachentwurfs umsetzen, können vielfältig sein. Beispielsweise könnte eine Konfiguration Komponenten verwenden, die ausschließlich mit Bruttopreisen rechnen. Eine andere verwendet teilweise Komponenten, die mit Nettopreisen umgehen und enthält dafür Adaptoren, die die Nettopreise bei den Aufrufen der Komponentendiensten in Bruttopreise umrechnen. Auch technisch verschiedene Konfigurationen könnte es einige geben. Daher ist in der Alternativengenerierung, die im hier betrachteten Fall der Suche nach Konfigurationen gleich kommt, unter Umständen eine Vorauswahl notwendig, um die Massen in den Griff zu bekommen.

Es ist wohl aufgrund dieser Überlegungen davon auszugehen, dass es Fälle geben kann, in der die Menge der Alternativen recht groß wird. Damit trifft einer der Einwände von Ncube und Dean (2002) zu, dass Verfahren, die auf paarweisen Vergleichen beruhen, bei einer hohen Alternativen- bzw. Zielanzahl kritisch werden können.

7.3.4 Unvollständige Vergleichsmatrizen

Falls die Alternativenmenge trotz aller Versuche, sie so klein wie möglich zu halten, noch immer eine gewisse Größe besitzt, gibt es noch eine weitere Möglichkeit Paarvergleiche einzusparen.

Die Anzahl der Fragen, die dem Entscheider gestellt werden müssen, kann weiter gesenkt werden. Wie bereits in vorangegangenen Abschnitten erläutert wurde, genügt zur Ermittlung einer konsistenten Vergleichsmatrix die Ermittlung einer einzigen Zeile dieser Matrix.

Falls die Matrix jedoch nicht konsistent ist, so dienen die übrigen ermittelten Einträge dazu, die Inkonsistenzen in den Aussagen des Entscheiders aufzudecken und bei der Ermittlung der Präferenzen zu berücksichtigen. Dieses Vorgehen ist dabei natürlich auch kein exaktes Verfahren, um die Präferenzen des Entscheiders genau zu bestimmen. Da somit sowieso eine Heuristik zum Einsatz kommt, kann die Frage gestellt werden, wie viele Einträge in der Vergleichsmatrix wirklich notwendig sind, damit ein ausreichendes Maß an Redun-

danz enthalten ist, um die Inkonsistenzen zu kompensieren, ohne dass diese Inkonsistenzen signifikanten Einfluss auf die ermittelten Werte haben.

Carmone et al. (1997) haben eine Analyse angestellt, bei der sie ermittelt haben, wie groß der Informationsverlust ist, wenn Einträge in der Vergleichsmatrix ausgelassen werden. Dazu haben die Autoren Simulationen durchgeführt, bei dem sie den Informationsgehalt von Vergleichsmatrizen mit vollständigen Informationen verglichen haben mit solchen, bei denen Informationen aus Paarvergleichen weggelassen worden sind. Sie kamen zu dem Schluss, dass in Anwendungen des AHP, in denen große Hierarchien oder eine hohe Anzahl an Attributen verwendet werden müssen, es durchaus zu keinem nennenswerten Informationsverlust kommt, selbst wenn 50% der Informationen einer Vergleichsmatrix weggelassen werden. Das heißt also, dass die Hälfte der Paarvergleiche weggelassen werden könnte.

Da die Verwendung einer standardisierten Software zum Einsatz des AHP empfohlen wird, ist auch hier, wie schon bei den anderen Modifikationen zu prüfen, ob die ausgewählte Software die entsprechenden Modifikationen unterstützt.

8 Auswahl einer Konfiguration

Nachdem die zur Auswahl stehenden Konfigurationen während der Suche (Kapitel 5) bestimmt worden sind, letztmalig auf ihre Fähigkeit, die geforderte Aufgabe zu lösen, überprüft wurden und die Kosten für die Erstellung dieser Konfigurationen (Kapitel 6) geschätzt wurden, bleibt nur noch, den subjektiven Nutzen der einzelnen Konfigurationen zu bestimmen. Hierzu wird auf die Grundlagen des vorangegangenen Kapitels eingegangen.

Da der Nutzen eine subjektive Einschätzung ist, kann die Frage gestellt werden, wessen Nutzen zu ermitteln ist. Im hier betrachteten Szenario wird davon ausgegangen, dass in der Regel ein Anwenderunternehmen einen Auftrag zur Erstellung einer Anwendung erteilt. Daher ist es sinnvoll, den Nutzen des Anwenderunternehmens zu betrachten, denn dieses Unternehmen muss mit der erstellten Anwendung arbeiten. Daher wird hier vorgeschlagen, die Paarvergleiche in Zusammenarbeit mit dem Auftraggeber durchzuführen. Sie können eventuell als Gruppenentscheidung (vgl. Bryson (1996)) zwischen den späteren Anwendern und dem entwickelnden Unternehmen durchgeführt werden.

Mit Hilfe dieses Nutzens ist es dann möglich, die Kosten und den Nutzen pro Alternative gegenüber zu stellen, um dann zu einer Entscheidung für eine Konfiguration zu gelangen. Die ausgewählte Konfiguration kann dann realisiert werden. Diese Realisierung sowie die anschließend notwendigen Tests und die Auslieferung der erstellten Anwendung liegen außerhalb des Interesses dieser Arbeit. Diese Schritte werden durch das bei der Entwicklung verwendete Vorgehensmodell (s. Abschnitt 3.2) vorgegeben.

Es soll an dieser Stelle jedoch noch einmal hervorgehoben werden, dass auch nach der Durchführung des Entscheidungsprozesses weiterhin Informationen gesammelt werden, die dazu dienen können, die verwendeten Prozesse zu verbessern. Hierzu zählen Erfahrungen aus den durchgeführten Suchvorgängen, aus einem Vergleich der tatsächlich entstandenen Kosten mit den geschätzten Kosten sowie Unterschiede zwischen dem Verhalten der erstellten Anwendung und der Vorhersage, wie sich die Anwendung verhalten sollte.

8.1 Ableitung einer Zielhierarchie

Zentrale Aufgabe der Nutzenermittlung unter Einsatz des AHP ist die Erstellung einer Hierarchie von Zielen. Die hier vorgestellte Hierarchie soll dabei einen ersten Anhaltspunkt geben. Dies kann jedoch nicht darüber hinwegtäuschen, dass eine solche Zielhierarchie situativ anzupassen ist.

Die Sinnhaftigkeit einzelner Ziele hängt zum einen von der verwendeten Alternativenmenge ab und zum anderen von der zu erstellenden Anwendung. Dass die Zielhierarchie von den Alternativen abhängt, ist anhand eines Beispiels leicht einzusehen. Angenommen es sei eine Middle-Tier Anwendung zu erstellen. Dann macht es wenig Sinn, diese Anwendung mit einer Benutzeroberfläche zu versehen. In diesem Fall ist es damit ebenso sinnlos, das Optimierungsziel „Gute Benutzeroberfläche“ zu betrachten. Genauso sinnlos ist ein Ziel, wenn alle Alternativen in diesem Ziel gleich gut abschneiden, also alle die selbe Ausprägung in einer Zielvariablen besitzen.

Von der zu erstellenden Anwendung hängt die Zielhierarchie daher ab, weil es einige Ziele gibt, die anwendungsspezifisch sind und daher nicht vorgegeben werden können. Beispielsweise könnte für ein Projektplanungsprogramm die Güte einer Heuristik zur Lösung des kapazitierten Projektplanungsproblems ein wichtiges Ziel sein. Dieses Ziel macht bei

einer Anwendung zur Lagerverwaltung wenig Sinn. Daher gibt es markierte Auslassungen in der hier vorgestellten Zielhierarchie, die noch situativ angepasst werden müssen. Bei der Verwendung einer Standardsoftware wie ExpertChoice sollte die Anpassung der Zielhierarchie problemlos möglich sein. Falls eine Spezialanwendung für den hier vorgestellten Prozess erstellt und verwendet werden soll, so muss diese dem Anwender die Möglichkeit bieten, die Zielhierarchie vor der Durchführung des AHP zu editieren, um die mit der zu erstellenden Anwendung verbundenen Ziele ebenfalls berücksichtigen zu können.

Die oberste Ebene der hier vorgeschlagenen Zielhierarchie ist in Abbildung 17 zu erkennen.

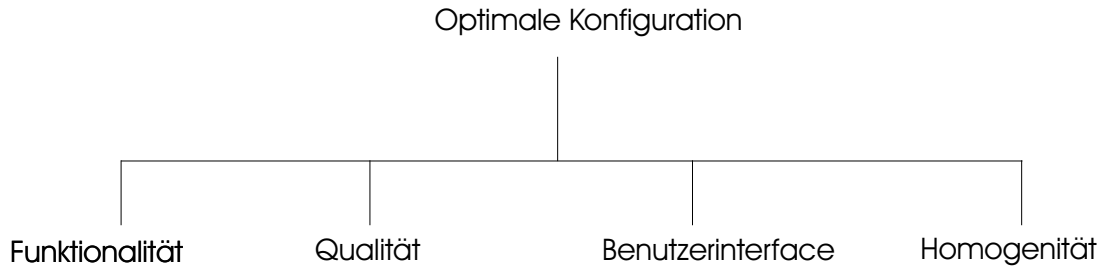

Abbildung 17: Erste Ebene der Zielhierarchie

Das Oberziel der Auswahl einer möglichst guten Konfiguration aus Komponenten wird unterteilt in die Unterziele Funktionalität, Qualität, Benutzeroberfläche/GUI und das Ziel, eine möglichst homogene Konfiguration zu erhalten. Diese einzelnen Unterziele sowie die ihnen wiederum zugeordneten Unterziele werden in den folgenden Abschnitten vorgestellt.

Dabei ist zu beachten, dass es Überschneidungen zwischen den Zielen auf der Ebene der Benutzeroberfläche und anderen Ebenen geben kann. Dies ist in den Paarvergleichen so zu berücksichtigen, dass Dinge, die sich auf die GUI beziehen, auch nur dort in die Entscheidung einbezogen werden. Alle Aspekte, die nichts mit der Benutzeroberfläche zu tun haben, sollten in den anderen Zielen bewertet werden.

8.1.1 Funktionalität

In Abbildung 18 ist der Teilbaum für das Zielsystem im Bezug auf die funktionalen Ziele abgebildet. Funktionale Ziele sind solche, die direkt oder indirekt mit dem angebotenen Funktionsumfang zu tun haben. Sie beschreiben daher die Möglichkeiten, die gestellte Aufgabe oder evtl. auch weiterführende Aufgaben mit Hilfe der Software zu erfüllen. Dabei sei noch einmal auf Abschnitt 5.6 verwiesen, der einen Schritt beschreibt, der sicherstellen soll, dass die hier betrachteten Alternativen alle die Mindestanforderungen erfüllen. Es geht daher in diesem Teil der Zielhierarchie nur um die Güte der Erfüllung dieser Aufgaben, um zusätzliche Funktionen, Werkzeuge, etc. Die benötigten Informationen sollten weitgehend der Aufgabenebene der Komponentenspezifikation zu entnehmen sein.

Beim ersten Unterziel der funktionalen Ebene, der *Güte der Aufgabenerfüllung* handelt es sich gleich um ein anpassbares Ziel, das von der zu konstruierenden Anwendung abhängt. Um zu verdeutlichen, wie diese Lücke aufzufüllen ist, sei wieder das Beispiel der Jahresabschluss-Anwendung aus früheren Kapiteln aufgegriffen. Da ein Jahresabschluss

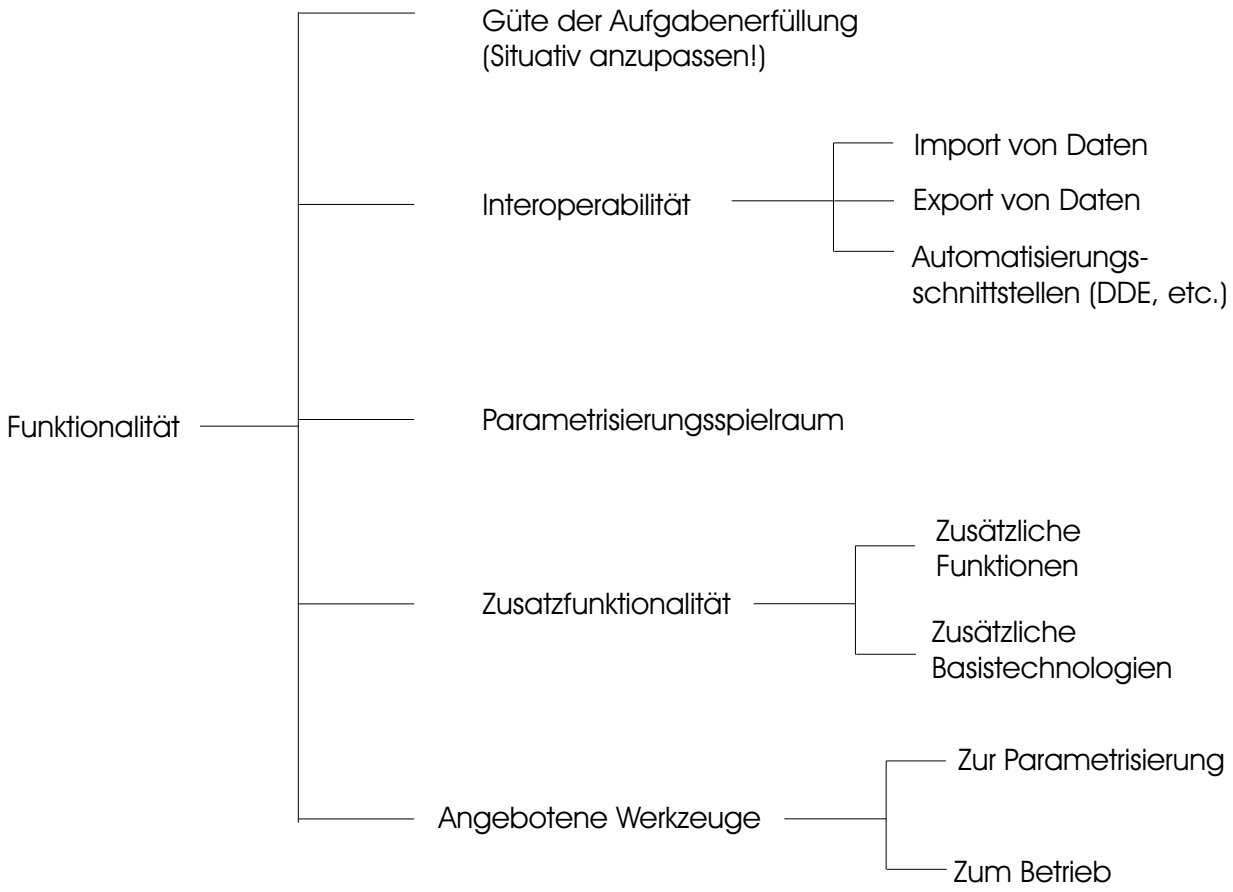

Abbildung 18: Teilbaum der funktionalen Ziele

nach gesetzlichen Vorschriften erstellt werden muss, kann die Güte der Aufgabenerfüllung nicht an vielen Punkten festgemacht werden. Eventuell können das Ausgabeformat oder die Übersichtlichkeit des erstellten Abschlusses als Gütekriterien eingeführt werden. Ein besseres Beispiel findet sich in der Einleitung zu diesem Kapitel, bei dem es um die Güte des Projektmanagementprogramms ging.

Das Ziel der *Interoperabilität* versucht die Funktionen zum Austausch von Daten mit anderen Programmen oder Komponenten zu erfassen. Die Unterziele Im- und Export von Daten sollen messen, wie viele verschiedene Datenformate die Anwendung verarbeiten kann. Die Art und Anzahl der unterschiedlichen Formate kann dabei als Entscheidungsgrundlage verwendet werden. Zusätzlich zu den Ein- und Ausgabeformaten bieten moderne Komponententechnologien Möglichkeiten, Daten mit anderen Komponenten direkt auszutauschen oder diese fernzusteuern. Büroanwendungen nutzen diese Fähigkeiten meistens, um die Dokumente der unterschiedlichen Anwendungen in andere Dokumenttypen zu integrieren. Beispielsweise kann eine Tabelle einer Tabellenkalkulation in ein Dokument der Textverarbeitung eingebettet werden. Die Anzahl der zur Verfügung stehenden Automationsschnittstellen dient als Maß, wie gut die Konfiguration hier abschneidet.

Unter dem Ziel des *Parametrisierungsspielraums* wird erfasst, in wie weit sich die Konfiguration auch nach ihrer Realisierung noch anpassen lässt. Im betrieblichen Umfeld werden diese Anpassungen sehr häufig vorgenommen, um allgemein gehaltene Programmteile auf

die Bedürfnisse des jeweiligen Anwenderunternehmens anzupassen. Die Anzahl der Parameter sowie ihre Art können beim Paarvergleich zweier Konfigurationen als Hinweise zur Beurteilung herangezogen werden.

Die von einer Konfiguration gebotene *Zusatzfunktionalität* soll durch zwei untergeordnete Ziele erfasst werden. Dabei geht es um Funktionalität, die von den unbedingt zu erfüllenden Anforderungen abweicht oder anders ausgedrückt um die „weichen Anforderungen“, wie sie in Abschnitt 5.6 eingeführt wurden. Das Ziel wurde dabei aufgeteilt in „Zusätzliche Funktionen“ und „Zusätzliche Basistechnologien“. Die zusätzlichen Funktionen können dabei eventuell situationsabhängig weiter präzisiert werden. Ansonsten kommen als Maß für die Erfüllung dieses Ziels die Art und die Anzahl der zusätzlich realisierten Funktionen in Frage. Bei den zusätzlichen Basistechnologien wurde an Komponenten mit zusätzlichen Konnektoren im Bezug auf ihre vorausgesetzten Basisanwendungen gedacht. Beispielsweise könnte die Konfiguration entsprechende Adaptoren besitzen, um sowohl mit der Datenbank von Microsoft als auch der von Oracle zusammen zu arbeiten. Oder eine Komponente kann gleichzeitig mit mehreren unterschiedlichen Anwendungen des Rechnungswesens Daten austauschen.

Um die Komplexität der Zusatzfunktionen oder der Parametrisierung in den Griff zu bekommen, bieten viele Anwendungen *Werkzeuge*, um die Parametrisierung zu unterstützen. Daher ist es sinnvoll zu ermitteln, ob die in der Konfiguration eingesetzten Komponenten entsprechende Werkzeuge bzw. Plug-Ins zum Einsatz in einem integrierenden Werkzeug bereits mitbringen. Dabei wurde die Art der Werkzeuge unterschieden in Werkzeuge zur Parametrisierung und solche, die während des Betriebs eingesetzt werden können. Erstere dienen dazu, die Anpassung auf das Anwenderunternehmen zu unterstützen, indem sie alle möglichen Parameter und deren Wertebereiche auflisten, damit diese entsprechend editiert werden können. Die Werkzeuge, die für einen Einsatz während des Betriebs vorgesehen sind, können dazu dienen, das System zu überwachen, es geordnet anzuhalten und zu starten, im Fehlerfall einen Report zu generieren, etc. Gemessen werden kann die Güte der Werkzeuge an ihrer Verfügbarkeit, ihrer Anzahl und dem Eindruck, den sie bei der Erfüllung ihrer Aufgabe hinterlassen.

8.1.2 Qualität

Die Zielhierarchie des Qualitätsziels ist in Abbildung 19 dargestellt. Die verwendeten Ziele orientieren sich an der Arbeit von Bertoa und Vallecillo (2002) (vgl. auch Abschnitt 3.4.3), an der Arbeit von Kontio (1995) und der Arbeit von Endres (2003). Alle genannten Autoren basieren ihre Qualitätsmerkmale auf der ISO Norm 9126, daher sind sie grundsätzlich vergleichbar, abgesehen von kleineren Erweiterungen, die jeder der Autoren eingeführt hat. Außerdem geben einige Autoren Attribute an, anhand derer die Erfüllung der einzelnen *Qualitätsziele* gemessen werden sollen. Einige dieser Attribute sollen hier aufgegriffen werden. Die entsprechenden Informationen sind der Qualitätsebene der Kompoentenspezifikation zu entnehmen, falls diese, wie in Abschnitt 3.4.3 beschrieben, angepasst wurde.

Unter dem Ziel der *Sicherheit* einer Konfiguration können mehrere Aspekte aufgefasst werden, daher kann sie auch anhand dieser Teilaspekte beurteilt werden. Mögliche Fähigkeiten der Komponente umfassen den verschlüsselten Austausch von Daten und hier insbesondere die unterstützten Verschlüsselungssysteme. Weiterhin können Zugangssperren, die in Form eines Benutzermanagements realisiert werden können, oder Verfahren, um den

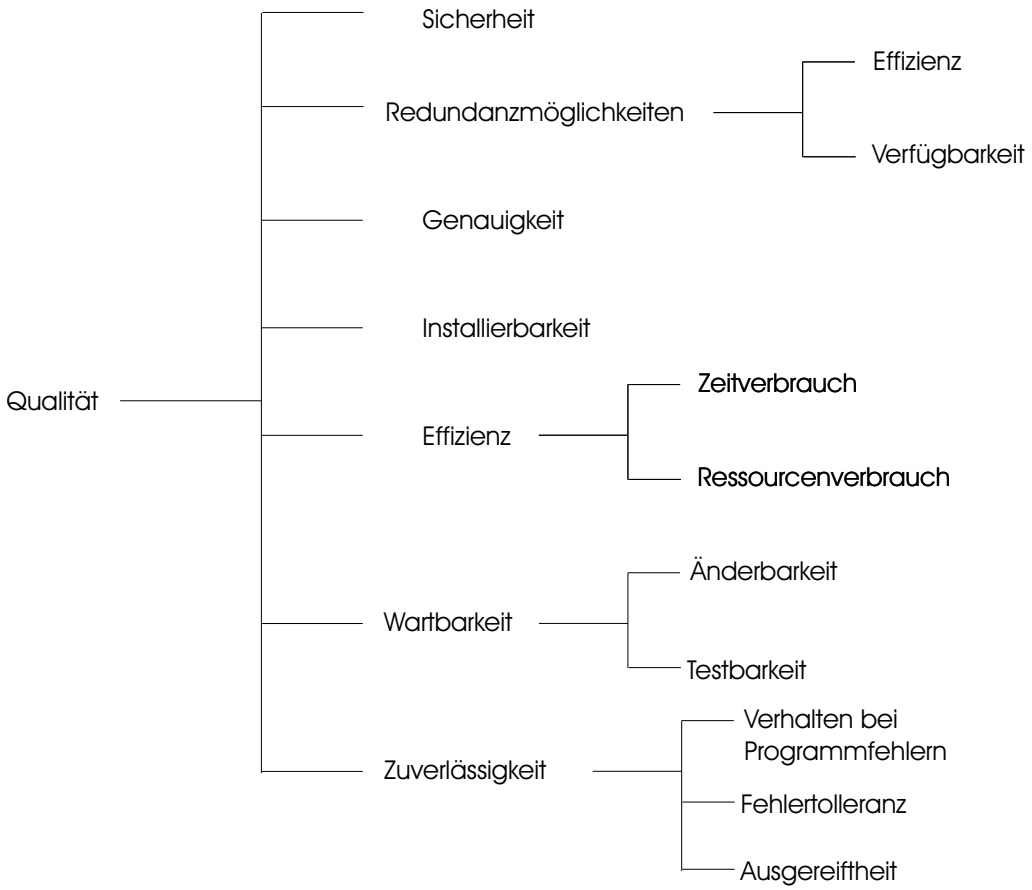

Abbildung 19: Teilbaum der Qualitätsziele

Zugriff auf die Daten zu protokollieren, berücksichtigt werden. Diese Kriterien können auch bei Bedarf in die Zielhierarchie als Unterziele des Sicherheitsziels übernommen werden, um die Vergleiche der Konfigurationen klarer zu gestalten. Als Hinweise bei den Paarvergleichen dient die generelle Verfügbarkeit der entsprechenden Sicherheitsvorkehrungen bzw. deren Art und Umfang.

Unter *Redundanzmöglichkeiten* sollen in der Konfiguration vorgesehene bzw. einfach zu realisierende Redundanzen erfasst werden. Komponententechnologien sind im Allgemeinen besonders gut geeignet, um Redundanzen in einem System vorzusehen, da es einfach ist, einzelne Komponenten auf verschiedenen Rechnern vorzuhalten. Daher wurde dieses Ziel explizit in die Zielhierarchie aufgenommen. Die Redundanzmöglichkeiten können weiter unterteilt werden in Möglichkeiten, die Bearbeitungsgeschwindigkeit zu erhöhen, indem Aufgaben auf redundant gehaltenen Komponenten gleichzeitig bearbeitet werden, und Möglichkeiten, die Verfügbarkeit zu verbessern, indem ausfallende Komponenten von redundanten Komponenten auf anderen Rechnersystemen ersetzt werden. Einige Komponententechnologien bieten für diese Aufgaben sogar eine explizite Unterstützung für die Programmierer an. Ermittelt werden soll hier, ob die untersuchte Konfiguration von solchen Möglichkeiten Gebrauch macht, und falls ja, in welchem Umfang.

Das *Genauigkeitsziel* erfasst, mit welcher Genauigkeit die verwendeten Datentypen arbeiten können oder wie präzise die Ergebnisse ausgegeben werden können. Gemessen wer-

den könnte die Anzahl der Bits zur Speicherung der Zahlenwerte oder die Anzahl der unterstützten Nachkommastellen. Dabei ist zu beachten, dass die Fähigkeiten der einzelnen Komponenten in der Konfiguration aggregiert werden müssen. So bestimmt diejenige Komponente mit der geringsten Genauigkeit die Genauigkeit der Konfiguration.

Unter *Installierbarkeit* wird die Einfachheit der Installation der Konfiguration zusammengefasst. Es handelt sich dabei um den Prozess, auf einem Zielrechner, der die Minimalanforderungen erfüllt, die Konfiguration in einen einsatzfähigen Zustand zu versetzen. Dabei ist bei ganzen Konfigurationen insbesondere zu beachten, dass alle benötigten Komponenten installiert werden müssen. Diese sollten daher Methoden zur Verfügung stellen, sich von externen Installationsroutinen auf das Zielsystem übertragen zu lassen. Bewertet werden sollte die Einfachheit dieser Installation, die Verfügbarkeit von Installationsroutinen, etc.

Das Ziel, eine möglichst hohe *Effizienz* zu erreichen, ist eine der ältesten Forderungen, die schon seit der Etablierung der Informatik an Software gestellt wird. Dabei wird klassischerweise unterschieden in Berechnungsgeschwindigkeit und Speicherverbrauch. Die Berechnungsgeschwindigkeit sollte dabei möglichst unabhängig von konkreten Zielrechnern oder auf eine genau definierte Testplattform hin geschätzt werden, um einen Vergleich zu ermöglichen. Evtl. sind zur Ermittlung sinnvoller Werte Testinstallationen der Komponenten oder der Konfigurationen notwendig, was nicht immer im Rahmen der Kosten liegen könnte (vgl. auch die Diskussion in Abschnitt 3.4.3). Der Speicherplatzverbrauch lässt sich dabei in der Regel besser bestimmen und direkt in Paarvergleichen benutzen.

Unter *Wartbarkeit* wird die Leichtigkeit von Änderungen aufgrund von Programmfehlern oder neuen Anforderungen zusammengefasst. Dabei wird in der Literatur häufig unterteilt in Änderbarkeit und Testbarkeit. Das erste Kriterium erfasst, wie einfach Änderungen an einem System vorgenommen werden können, während das zweite Kriterium beschreibt, wie gut die Konfiguration zu testen ist. Die Testbarkeit kann anhand von vorhandenen Testfällen und Testtreibern sowie aufgrund von Erfahrungen ermittelt werden. Die Änderbarkeit einer Konfiguration lässt sich in der Regel nur schwer vorhersehen. Es gibt normalerweise nur wenige Indizien, die eine gute Änderbarkeit implizieren. Neben der Tatsache, dass eine komponentenorientierte Architektur sowieso gut geeignet sein sollte, Änderungen umzusetzen, kommt es hauptsächlich auf eine gute Systemarchitektur an. Die Güte einer Architektur kann allerdings nur mit Erfahrung und einem Gespür für gute Systemdesigns bewertet werden. Daher gibt es hier wenig systematische Unterstützung in Form von messbaren Attributen für den Entscheider bei den Paarvergleichen.

Als letztes Unterziel des Qualitätsziels wird hier die *Zuverlässigkeit* in die Zielhierarchie aufgenommen. Die Zuverlässigkeit einer Konfiguration kann durch die Fehlertoleranz sowohl gegenüber Programmfehlern als auch gegenüber Bedienerfehlern bestimmt werden. Daher werden als Unterziele des Zuverlässigkeitsziels das Verhalten bei Programmfehlern, die Fehlertolleranz gegenüber Falscheingaben sowie die Ausgereiftheit erfasst. Das Ziel „Verhalten bei Programmfehlern“ versucht zu vergleichen, wie sich die Konfigurationen in unvorhergesehenen Situationen verhalten. Es geht um Fragestellungen wie die, ob eine Anwendung bei einem Fehler ordnungsgemäß weiterarbeitet oder ob sie beispielsweise einfach abstürzt. Und falls sie abstürzen sollte, ob Mechanismen vorgesehen sind, die dafür sorgen, dass die Anwendung von alleine wieder anläuft. Bei der Toleranz gegenüber Eingabefehlern gilt ähnliches. Es wird untersucht, ob die Anwendung bei Falscheingaben entsprechende Warnungen

generiert oder ob sie mit den falschen Daten einfach weiter arbeitet oder gar abstürzt. Die Ausgereiftheit soll erfassen, wie gut zum einen die verwendeten Komponenten zum anderen auch die verwendeten Adaptoren und Konnektoren bereits getestet sind. Als Maß hierfür wird häufig die Installationsbasis verwendet, die angibt, auf wievielen Systemen die betrachteten Komponenten sich bereits im Einsatz befinden. Je mehr solcher Systeme existieren, umso wahrscheinlicher ist es, dass alle Fehler der Komponente bereits gefunden wurden. Da es normalerweise mehrere Komponenten in einer Konfiguration geben wird, können zur Aggregation der Daten evtl. die größte, kleinste und durchschnittliche Installationsbasis über alle Komponenten der Konfiguration bestimmt und als entscheidungsunterstützender Hinweis verwendet werden.

8.1.3 Benutzeroberfläche

Falls die betrachteten Komponenten eine Anwendung auf der Client-Seite realisieren, so wird sie gemäß den heute üblichen Standards eine grafische Benutzeroberfläche besitzen. Daher kommen Komponenten zur Darstellung und Ein- bzw. Ausgabe der Daten zum Einsatz (vgl. auch Abschnitt 3.4.2). Sollen dagegen serverseitige Anwendungen erstellt werden, so werden die Komponenten in aller Regel keine Benutzeroberfläche besitzen. In diesem Fall kann dieser Teilbaum der Zielhierarchie vollständig entfallen. Auch hier gilt also wieder, dass die vorgestellte Hierarchie ein Leitfaden geben soll, jedoch nicht den Anspruch auf Allgemeingültigkeit erheben kann.

In Abbildung 20 ist eine Zielhierarchie für die Bewertung des Nutzens der Benutzeroberflächen der verschiedenen Konfigurationen dargestellt. Die benötigten Informationen sollen bei einem fortgeschrittenerem Stand der Forschung der Komponentenspezifikation zu entnehmen sein. Wie Abschnitt 3.4.2 erläutert, sind entsprechende Überlegungen über eine Erweiterung des Spezifikationsrahmens momentan im Gang. Bis diese zu einem Ergebnis kommen, sind die notwendigen Informationen manuell zu ermitteln.

Die vorgeschlagenen Ziele beginnen bei dem Ziel der *Standardkonformität*. Hierbei soll bewertet werden, in wie fern sich die Oberfläche an bekannte Oberflächenstandards hält. Solche Standards könnten der Microsoft Windows Oberflächenstandard sein oder der Motif Standard unter Unix. Als Maß kann die Güte der Einhaltung dieser Standards dienen. Hierbei ist auch wieder zu beachten, dass dieses Maß für jede Komponente einzeln bestimmt werden kann und hier wieder die aggregierten Maße verwendet werden müssen. Daher sollten bei der Bewertung wieder die Extremfälle besonders betrachtet werden, also insbesondere Komponenten, die sich an keine Standards halten.

Wurde im vorangegangenen Ziel bestimmt, wie standardkonform die einzelnen Komponenten sind, so soll beim Ziel der *Einheitlichkeit* bewertet werden, wie die Komponenten im Zusammenspiel innerhalb einer Konfiguration zurecht kommen. Beispielsweise wird eine Komponente im Windows Look and Feel nur sehr schlecht mit einer Komponente mit Motif Look and Feel zusammen benutzbar sein. Daher wurde für die Bewertung von Konfigurationen dieses Ziel aufgenommen. Als Maß könnte die Anzahl unterschiedlicher Look and Feels innerhalb der Konfiguration dienen. Außerdem helfen Screen Shots von den einzelnen GUI Komponenten, um abschätzen zu können, ob diese im Zusammenspiel gut aussehen und bedienbar bleiben.

Die *Dokumentation und das Hilfesystem* werden im folgenden Ziel berücksichtigt. Dabei ist auch wieder die Aggregation der einzelnen Komponenten zu beachten. Auch hier emp-

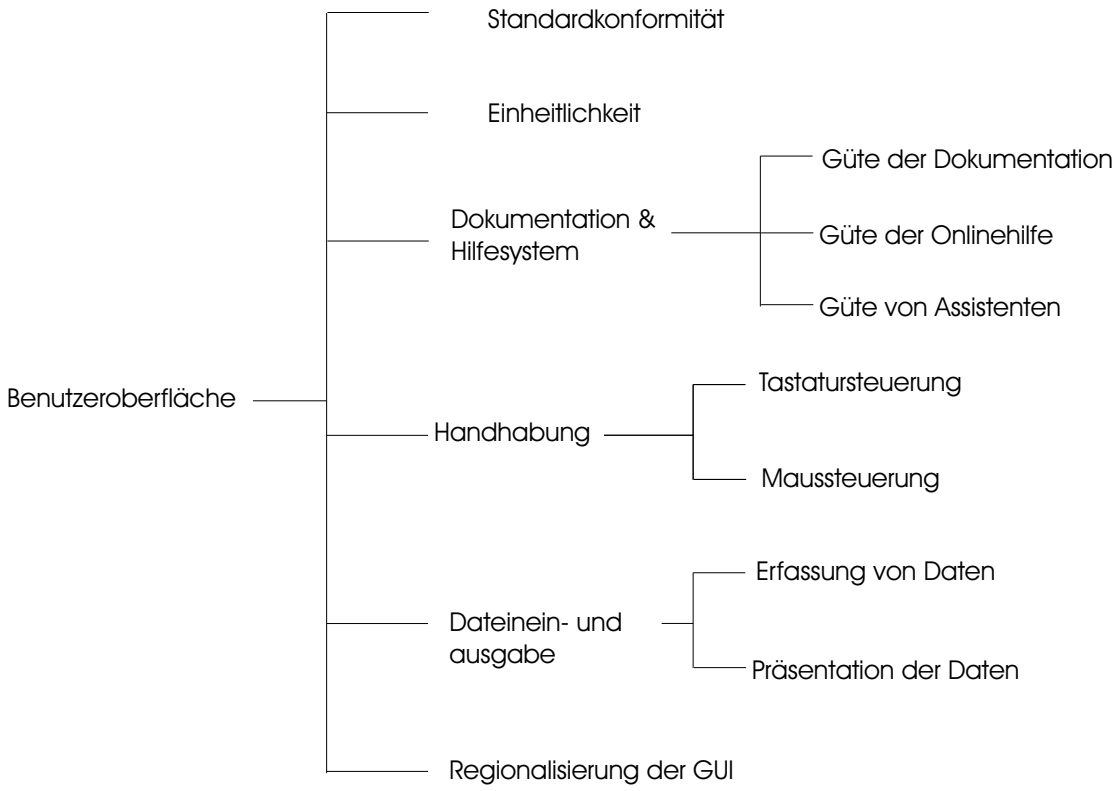

Abbildung 20: Teilbaum der benutzeroberflächenbezogenen Ziele

fiehlt es sich wieder, die Extremfälle besonders anzusehen, also Komponenten mit besonders guter bzw. besonders schlechter Benutzerdokumentation. Als Maß soll die Bewertung der Dokumentation zur Benutzung der Oberflächenkomponente, die von der Komponente angebotene Onlinehilfe sowie eventuell vorhandene Assistenten (Wizards) dienen. Der Funktionsumfang dieser Dokumentationssysteme ist der Komponentenspezifikation oder einer Testinstallation zu entnehmen.

Die *Handhabung* der Komponenten sowie der Konfiguration im Zusammenspiel lässt sich im Voraus wohl eher schwer bewerten. Unter Umständen stehen aber Testinstallationen sowie Testbenutzeroberflächen zur Verfügungen, die zur Bewertung herangezogen werden können. Dabei könnten die in der Konfiguration verwendeten Komponenten in einer Benutzeroberfläche zusammengestellt worden sein, ohne dass Funktionalität implementiert wurde. Falls diese Informationen zur Verfügung stehen, kann die Tastatur- und Maussteuerung bewertet werden.

Wie gut die in der Konfiguration zusammengestellten Komponenten in der Lage sind, den Benutzer bei der Erfassung der Daten zu unterstützen, soll das Ziel Unterstützung bei der *Dateneingabe und -ausgabe* ermitteln. Dabei hängt es von den Fähigkeiten der einzelnen Bausteine der Konfiguration ab, welche Möglichkeiten dem Benutzer letztendlich zur Verfügung stehen. Beispielsweise könnte eine Konfiguration Komponenten zur Darstellung der Daten als Kuchen- und Balkendiagramme bieten, die andere enthält Komponenten zur tabellarischen Darstellung. Bewertet werden soll anhand von Screen Shots oder Testtreibern die Angemessenheit der Ein- bzw. Ausgabemasken.

Als letztes Ziel wurden die Fähigkeiten zur *Regionalisierung* aufgenommen. Eine Regionalisierung auf ein bestimmtes Land bzw. eine bestimmte Kultur macht jedoch nur dann Sinn, wenn alle Komponenten in einer Konfiguration mit der entsprechenden Lokalisierung umgehen können. Daher ist zuerst die Schnittmenge aller gebotenen Lokalisierungen über alle Komponenten zu bilden, um die tatsächliche Menge an möglichen Regionalisierungen zu bekommen. Die Anzahl und die einzelnen möglichen Ländereinstellungen können als Bewertungsgrundlage beim Paarvergleich dienen.

8.1.4 Homogenität/Zusammenarbeit

Das Unterziel der Homogenität wurde als Bewertung der Konfiguration an sich eingeführt. Bei der Bewertung der Funktionalen Ebene und der Benutzeroberfläche wurden die aus der Zusammenstellung und Zusammenarbeit der einzelnen Komponenten entstehenden Fähigkeiten der Konfiguration betrachtet. Bei der Bewertung der Qualitätsaspekte wurde dagegen der Blick schon eher auf die eigentliche Architektur der Konfiguration gerichtet. Unter dem Oberbegriff der Homogenität soll der Blick auf das Zusammenspiel der Komponenten in einer Konfiguration gerichtet werden. Die Überwindung von Heterogenitäten ist besonders kritisch bei der Entwicklung von Anwendungen aus Komponenten (s. auch Abschnitt 5.1). Daher sollte eine Konfiguration einen möglichst hohen Grad an Homogenität aufweisen, wenn die zu erwartenden Probleme aus der Heterogenitätsüberwindung möglichst klein gehalten werden sollen. Hierzu werden einige Unterziele zum Homogenitätsziel betrachtet, die in Abbildung 21 dargestellt wurden.

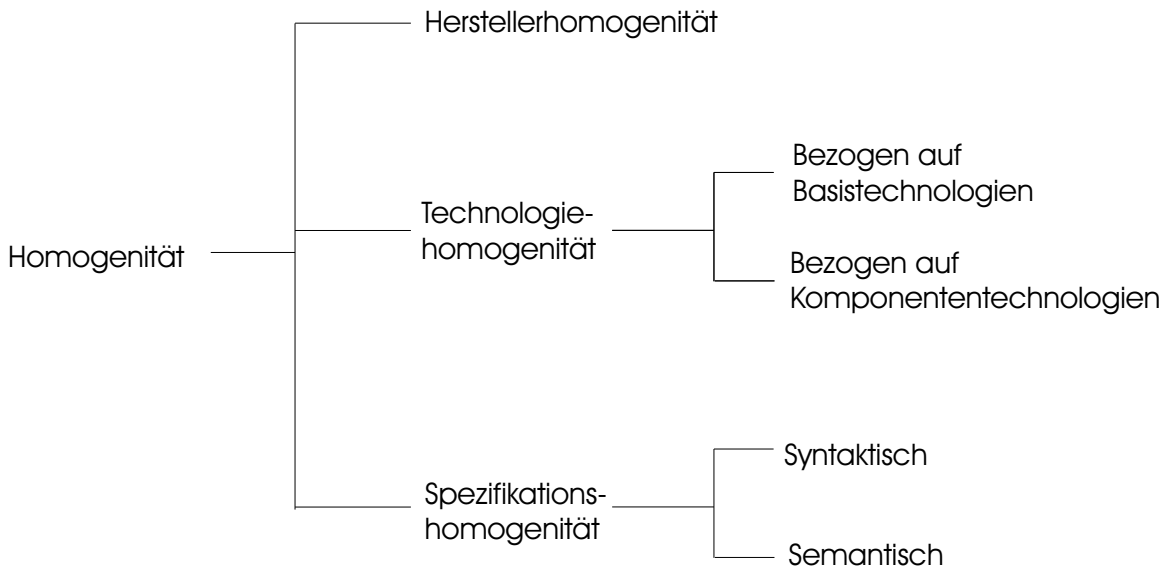

Abbildung 21: Teilbaum der auf die Zusammenarbeit bezogenen Ziele

Unter *Herstellerhomogenität* soll erfasst werden, von wie vielen verschiedenen Herstellern die verwendeten Komponenten stammen. Dies ist insbesondere bei später anfallenden Wartungsarbeiten von Interesse. Sollte ein Hersteller eine seiner Komponenten austauschen, so kann in der Regel davon ausgegangen werden, dass die anderen Komponenten aus seinem Angebot ebenfalls angepasst werden, damit sie mit der aktuellen Version zusammenarbei-

ten, auch falls diese nicht mehr kompatibel zur Vorgängerversion sein sollte. Werden statt dessen viele Komponenten von unterschiedlichen Herstellen eingesetzt, so ergeben sich mit größerer Wahrscheinlichkeit bei späteren Wartungsarbeiten Probleme.

Das Ziel der *Technologiehomogenität* soll erfassen, wie viele unterschiedliche Technologien als Grundlage für die betrachtete Konfiguration notwendig sind. Je weniger unterschiedliche Technologien notwendig sind, umso wahrscheinlicher ist es, dass es keine Probleme bei der Zusammenarbeit geben wird. Verwendet dagegen jede Komponente eine andere Datenbank, so steigert dies nicht nur die Kosten, sondern verringert auch die Chancen einer sinnvollen Zusammenarbeit. Das Oberziel wird dabei aufgeteilt in die Betrachtung der Basistechnologien (wie die betrachteten Datenbanken) und die Betrachtung der Komponententechnologien. Die Verwendung verschiedenster Komponententechnologien führt zu einer größeren Menge an notwendigen Adaptoren bzw. Brückencode und kostet in der Regel spürbar Effizienz. Beide Unterziele können mit Hilfe einer Auflistung der unterschiedlichen Technologien bewertet werden.

Unter *Spezifikationshomogenität* soll zusammengefasst werden, wie viele verschiedene Spezifikationssprachen für die verwendeten Komponenten zum Einsatz kamen. Würden die Annahmen dieser Arbeit erfüllt, so wäre diese Betrachtung überflüssig, da alles auf einem erweiterten Spezifikationsrahmen nach Ackermann et al. basiert wäre. In der Praxis ist diese Forderung jedoch noch nicht durchzusetzen. Daher soll gefordert werden, dass zumindest so wenig verschiedene Spezifikationssprachen verwendet werden wie möglich. Dies wird durch das Unterziel der syntaktischen Homogenität erfasst. Aber auch auf der semantischen Ebene sollten so wenig Unterschiede bestehen wie möglich. Dies könnte zum Beispiel durch die Verwendung von Branchenstandards unterstützt werden. Oder es könnte überprüft werden, ob alle Komponenten die gleiche Normsprache verwenden. Als Hinweis bei der Abgabe der Präferenzaussagen könnte die Relation zwischen der Anzahl verwendeter Spezifikationssprachen bzw. semantischer Standards zur Anzahl der insgesamt in der Konfiguration auftretenden Komponenten dienen.

8.1.5 Zusammenfassung

Mit der hier vorgeschlagenen Zielhierarchie sollten die meisten Evaluierungen weitgehend abgedeckt sein. Wie bereits mehrfach angesprochen wurde, ist es immer notwendig, die Zielhierarchie situativ anzupassen, sie dient daher nur als Ausgangspunkt bei der Durchführung des AHP.

Eine Zielhierarchie muss die in Abschnitt 3.7.4 aufgeführten Kriterien erfüllen: Vollständigkeit, Redundanzfreiheit, Messbarkeit, Präferenzunabhängigkeit und Einfachheit.

Die Vollständigkeit muss in jedem konkreten Fall überprüft werden, nachdem die Zielhierarchie an die Situation angepasst worden ist. Die Vorgabe eines Rahmens an Zielen sollte jedoch dazu führen, dass relevante Ziele nicht so einfach vergessen werden. Insgesamt ist es mindestens genauso wichtig, sich über die relevanten Ziele im Klaren zu sein, als die eigentliche Bewertung vorzunehmen.

Redundanzfrei sollte die angegebene Hierarchie sein und die Messbarkeit wurde jeweils durch die Angabe der zu betrachtenden Hinweise bzw. Maßzahlen berücksichtigt. Einfachheit kann der erstellten Zielhierarchie vermutlich noch unterstellt werden, sie enthält keine Teilbäume mit wesentlich mehr als fünf Zielen und ihre Tiefe beträgt maximal drei Ebenen. Allerdings kommt auch hier noch die situative Anpassung hinzu, die einige Ziele überflüssig

machen könnte, dafür aber andere Ziele weiter konkretisieren bzw. neu einführen wird.

Die Durchführung des eigentlichen AHP wurde bereits in Kapitel 7 geschildert. Bei der Auswertung der Daten sowie der Durchführung der Sensitivitätsanalyse sollte eine entsprechende Standardsoftware wie ExpertChoice hilfreich sein.

Als hilfreich erscheint eine Auswertung der Daten in der Art, dass nicht nur das aggregierte Endergebnis betrachtet wird, sondern zusätzlich auch das Abschneiden der Alternativen unter den jeweiligen Unterzielen auf Auffälligkeiten hin untersucht wird. Weiterhin sollte mindestens eine Sensitivitätsanalyse der Zielgewichte der ersten Ebene der Zielhierarchie durchgeführt werden. ExpertChoice bietet hierfür eine Reihe verschiedener Graphen oder dynamisch veränderbarer Darstellungen an.

Eine Bemerkung soll an dieser Stelle noch gemacht werden. Es könnte angedacht werden, die Erwartungen des Entscheiders zur Güte der verwendeten Spezifikationen in die Nutzenbewertung aufzunehmen. Damit könnte das Risiko erfasst werden, dass sich eine Komponente nicht gemäß ihrer Spezifikation verhält. Ist dieses Risiko als hoch einzuschätzen, so würde die Komponente einen entsprechend geringeren Nutzen bekommen. Erfasst werden könnte dieses Risiko durch das Vertrauen, das der Entscheider in den Hersteller der Komponente bei der Spezifikation dieser Komponente legt. Da es sich hierbei jedoch um die Erfassung eines Mangels handelt, der gemäß der getroffenen Annahme (A.3) aus Kapitel 4 nicht auftreten darf, wurde das Herstellervertrauen in dieser Arbeit nicht berücksichtigt, obwohl es in der aktuellen Praxis in der Komponentenorientierung eine nicht zu verachtende Rolle spielt.

8.2 Szenarien

Eine Erweiterung des AHP, um Vorhersagen über mögliche zukünftige Ereignisse in den Entscheidungsprozess zu integrieren, machen Forman und Selly (2001, Kapitel 7). Diese Erweiterung soll auch hier genutzt werden, um bei der Entscheidung die mögliche, für die Zukunft geplante Nutzung der zu erstellenden Anwendung zu berücksichtigen.

Ein Beispiel, das schon in einem früheren Kapitel verwendet wurde, soll hierzu noch einmal aufgegriffen werden. Aus den strategischen Planungen des Unternehmens, das den Auftrag für eine zu erstellende Bilanzierungssoftware vergibt, ist zu entnehmen, dass das Unternehmen plant, seine Geschäftstätigkeiten nach USA auszuweiten. Bislang wurde gemäß einer Anforderungsanalyse eine Bilanzierung nach den europäischen Richtlinien benötigt. Wird die strategische Planung einbezogen, so könnte es notwendig werden, auch nach den amerikanischen Standards zu bilanzieren. Dass unter Berücksichtigung dieser Planung Konfigurationen, die die Bilanzierung nach US-GAAP beherrschen, besser abschneiden als ohne die Betrachtung dieser Planung, ist einleuchtend.

Forman schlägt daher für solche Fälle vor, die Wahrscheinlichkeiten für das Eintreten des einen oder des anderen Szenarios in den AHP zu integrieren. Anstelle von Präferenzaussagen werden dabei in den Paarvergleichen Wahrscheinlichkeitsaussagen abgefragt. Die Frage könnte so gestellt werden: „Wie viel wahrscheinlicher ist Szenario i als Szenario j?“. Die Szenarien werden dabei wie in Abbildung 22 unterhalb des obersten Ziels integriert. Unter den Szenarien wird die vollständige Zielhierarchie eingehängt, die ohne Szenarien betrachtet worden wäre. Ansonsten läuft die Durchführung des AHP wie gewohnt ab.

Allerdings müssen nun für die gesamte Zielhierarchie die Paarvergleiche pro Szenario durchgeführt werden, was recht schnell unpraktikabel wird, da die Anzahl der durchzufüh-

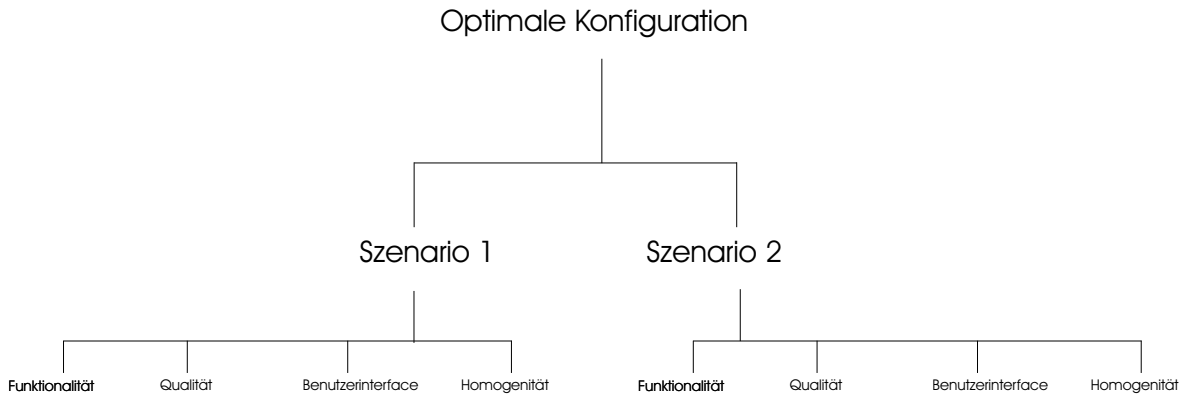

Abbildung 22: Anpassung des AHP auf die Verwendung von Szenarien

renden Paarvergleiche zu hoch wird. Ein Vorschlag, der gemacht werden könnte, wäre, die Paarvergleiche für ein Szenario durchzuführen, die Ergebnisse für das andere Szenario zu übernehmen und dann anzupassen. Dies birgt allerdings die große Gefahr, dass eigentlich anzupassende Aussagen bei der Überarbeitung übersehen werden.

8.3 Kosten-Nutzen-Diagramm

Nachdem der AHP auf die Alternativen angewendet wurde, können im letzten Schritt die ermittelten Kosten den aus den Präferenzen ermittelten Nutzwerten gegenübergestellt werden. In Abbildung 23 wurde dies anhand fiktiver Alternativen einmal grafisch durchgeführt.

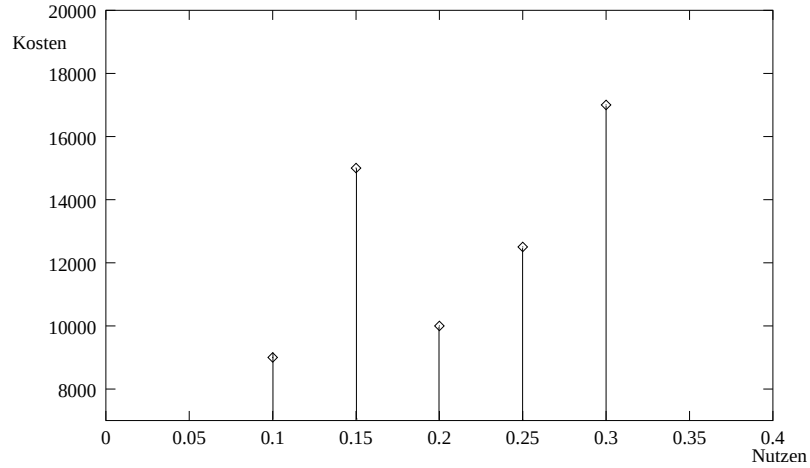

Abbildung 23: Ein Beispiel eines Kosten/Nutzen Diagramms

Anhand der gewonnenen Daten lässt sich nun das Verhältnis von subjektivem Nutzen zu den geschätzten Kosten bestimmen:

$$\text{rel. Nutzen} = \frac{\text{subjektiver Nutzen}}{\text{geschätzte Kosten}}$$

Ausgewählt werden könnte dann beispielsweise die Alternative mit dem höchsten relativen Nutzen.

Werden die in Abbildung 23 verwendeten Daten betrachtet, ergibt sich die Tabelle 3.

	A_1	A_2	A_3	A_4	A_5
Subjektiver Nutzen	0.2	0.25	0.1	0.15	0.3
Geschätzte Kosten	10000	12500	9000	15000	17000
rel. Nutzen	$0,00002$	$0,00002$	$0,000011$	$0,00001$	$0,000018$

Tabelle 3: Kosten/Nutzen Vergleich

Die Daten würden bei der oben angegebenen Entscheidungsregel dafür sprechen, die erste Alternative auszuwählen, da sie bei gleichem Nutzen-Kosten-Verhältnis wie Alternative zwei billiger ist. Da die angegebene Entscheidungsregel auch nur eine Heuristik ist, könnte jedoch auch Alternative zwei und Alternative fünf noch einmal genauer betrachtet werden.

9 Kritik und Ausblick

Diese Arbeit beschäftigte sich mit der Erstellung eines Prozesses für die im Konfigurationsmanagement anfallenden Aufgaben. Dabei wurde weniger das Änderungs- und Versionsmanagement betrachtet, sondern der Schwerpunkt lag auf den Aufgaben, die aus dem Auswahlmanagement entstehen. Es wurden Vorschläge gemacht, wie eine Suche, eine Kostenabschätzung und eine Auswahl aus unterschiedlichen Konfigurationen realisiert werden könnte.

Zentraler Gedanke war dabei der Wechsel der Sichtweise weg von der *Auswahl einer einzelnen Komponente* hin zur *Auswahl ganzer Konfigurationen*. Dieser Wechsel ist aufgrund der Problematik der Abhängigkeiten der Komponenten untereinander und der Tatsache, dass die Summe der lokalen Optima nicht gleich dem globalen Optimum ist, begründet worden.

Dabei tritt allerdings die Frage auf, wie die Informationen über die einzelnen Komponenten und die Informationen über die Art und Weise, wie die Komponenten zu neuen Bauteilen zusammengesetzt werden, aggregiert werden können, um eine Entscheidungsgrundlage zu bilden. Hier wären weitere aus der Praxis zu gewinnende Erfahrungen notwendig.

Aus diesen Gründen empfiehlt es sich den vorgestellten Prozess abschließend in einer Fallstudie, etwa im Rahmens eines Projektseminars, zu evaluieren. Die dabei auftretenden Erfahrungen wären in den Prozess einzuarbeiten, um ihn weiter zu verbessern. Insbesondere die Zielhierarchie sollte empirisch überprüft werden, indem die Entscheider befragt werden, ob sie mit den empfohlenen Konfigurationen einverstanden und mit den vorgeschlagenen Zielvariablen zufrieden waren.

Besonderes Interesse liegt hier auf der Frage nach der Vollständigkeit und der Frage, ob alle Ziele weit genug operationalisiert sind, um eine entsprechende Datenbasis aus der Konfiguration gewinnen zu können. Außerdem wäre von Interesse, welche Zeitspannen für die Auswertungen benötigt werden, da ein hoher Zeitverbrauch zu hohen Kosten nach sich zieht und darüber hinaus dazu führt, dass der Prozess nicht angewendet wird. In diesem Fall ist die Zielhierarchie weiter einzuschränken.

Dabei ist nach wie vor zu beachten, dass die komponentenorientierte Anwendungsentwicklung sowohl in der Theorie als auch in der Praxis noch immer ein sehr neues Gebiet der Informatik als auch der Wirtschaftsinformatik ist. Insbesondere die Verfügbarkeit entsprechend großer Mengen an Komponenten, aus denen auf der Suche nach einer zu realisierenden Konfiguration ausgewählt werden kann, ist aktuell nicht gegeben. Für die Durchführung der Fallstudie könnte daher eine Testumgebung bereitgestellt werden, in der beispielsweise die Komponenten eines begrenzten Anwendungsfeldes zur Verfügung gestellt werden. Andere Arbeiten haben auf diese Art und Weise ebenfalls ihre Testfälle durchgeführt.

Weiterer Schwachpunkt in der aktuellen Situation stellt die geringe Anzahl an Arbeiten über Konfigurationen von Komponenten dar. Insbesondere Vorhersagemodelle für Eigenschaften von Konfigurationen auf der Basis einer Komponentenspezifikation sind nicht verfügbar und müssen daher durch erfahrene Entwickler kompensiert werden.

Die Schwierigkeiten fangen bei diesem Ansatz allerdings schon damit an, dass ein allgemein akzeptierter Spezifikationsrahmen für die einheitliche Beschreibung von Komponenten sich noch nicht durchgesetzt hat und es ist auch noch nicht abzusehen, ob sich überhaupt jemals einer etablieren wird. Ein solcher Rahmen wird jedoch meist als Voraussetzung für

eine weitreichende Etablierung von Komponentenmarktplätzen angesehen, die dann die Quelle für den hier vorgestellten Suchprozess darstellen.

Darüber hinaus ist sicher auch die Frage interessant, ob Unternehmen überhaupt dazu bereit sind, ihre Entwicklungsprozesse in der hier vorgestellten Form zu systematisieren. Der Gedanke, Software in einem Prozess ingenieurmäßig zu entwickeln hat sich auch noch nicht vollständig in der Praxis durchgesetzt, obwohl das Software Engineering eine seit längerer Zeit verfolgte Richtung der praktischen Informatik ist. Insbesondere die Tatsache, dass Qualitätssicherungsprozesse immer noch mühsam in die Entwicklungsprozesse integriert werden, zeigt, wie lange es dauert, bis sich systematische Vorgehensweisen etablieren.

Auf der anderen Seite zeigen Aktivitäten wie die, die Microsoft bei der Umstellung von Navision auf eine komponentenorientierte Basis durchführt, dass durchaus in der Industrie der Glaube an die Vorteile der Komponentenorientierung vorhanden ist. Daher wird es interessant zu beobachten sein, wie sich dieser Zweig der Informatik in der näheren Zukunft entwickeln wird.

Insbesondere falls die Unterstützung mit Methoden und Werkzeugen weiter voran getrieben wird und zusätzlich immer wieder neue Technologien, wie beispielsweise die WebService Technologie, die eine komponentenorientierte Wiederverwendung übers Internet ermöglicht, entstehen, ist die Hoffnung auf eine Durchsetzung der Komponentenorientierung auf breiter Front mehr als berechtigt.

Für die hier vorgelegte Arbeit wären besonders ein spezialisiertes Komponentenrepository und ein darauf aufbauender Marktplatz für Softwarekomponenten interessant. Das erste Arbeiten wie das CompoNex Projekt der TU Darmstadt, das einen Marktplatz für Softwarekomponenten realisiert hat, auch in der Industrie auf entsprechendes Interesse stoßen, zeigt, dass das Interesse an der neuen Technologie und Denkweise nach wie vor vorhanden ist.

Literatur

Abts, C.; B. W. Boehm und E. B. Clark (2000): *COCOTS: a COTS software integration cost model - model overview and preliminary data findings*. In *Proceedings of the European Software Control and Metrics (Escom)* S. 325 – 333. München.
URL http://dspace.dial.pipex.com/town/drive/gcd54/conference2000/abts.pdf.

Abts, C. M. und B. Boehm (1997): *COTS Software Integration Cost Modeling Study*. USC Center for Software Engineering, University of Southern California.
URL http://sunset.usc.edu/research/COCOTS/docs/USAFReport.pdf.

Ackermann, J. (2002): *Spezifikation des Parametrisierungsspielraums von Fachkomponenten - Erste Überlegungen*. In K. Turowski, Hrsg., *Modellierung und Spezifikation von Fachkomponenten. 3. Workshop* S. 17 – 68. Nürnberg.

Ackermann, J. (2003): *Zur Spezifikation der Parameter von Fachkomponenten*. In K. Turowski, Hrsg., *Tagungsband 5. Workshop Komponentenorientierte betriebliche Anwendungssysteme* S. 47 – 154. Universität Augsburg.

Ackermann, J.; F. Brinkop; S. Conrad; P. Fettke; A. Frick; E. Glistau; H. Jaekel; O. Kotlar; P. Loos; H. Mrech; E. Ortner; U. Raape; S. Overhage; S. Sahm; A. Schmieten; T. Teschke und K. Turowski (2002): *Vereinheitlichte Spezifikation von Fachkomponenten*. Gesellschaft für Informatik (GI), Arbeitskreis 5.10.3.
URL http://www.fachkomponenten.de.

Alves, C. und J. Castro (2001): *CRE: A Systematic Method for COTS Components Selection*.
URL http://www.cs.ucl.ac.uk/staff/C.Alves/SBES'01_Alves.pdf.

Alves, C. und A. Finkelstein (2002): *Challenges in COTS Decision-Making: A Goal-Driven Requirements Engineering Perspective*. University College London, UK.
URL http://www.cs.ucl.ac.uk/staff/C.Alves/seke'02_Alves.pdf.

Basili, V. R.; G. Caldiera und G. Cantone (1992): *A Reference Architecture for the Component Factory*. ACM Transactions on Software Engineering and Methodology **1** (1), S. 53 – 80.

Basili, V. R.; G. Caldiera und H. D. Rombach (1999): *Experience Factory*. In J. J. Marciniak, Hrsg., *Encyclopedia of Software Engineering* S. 469 – 476. Wiley-Interscience.

Becker, S. und S. Overhage (2003): *Stücklistenbasiertes Komponenten-Konfigurationsmanagement*. In K. Turowski, Hrsg., *Tagungsband 5. Workshop Komponentenorientierte betriebliche Anwendungssysteme* S. 17 – 32. Universität Augsburg.

Bertoa, M. F. und A. Vallecillo (2002): *Quality Attributes for COTS Components*. 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002).
URL http://alarcos.inf-cr.uclm.es/qaoose2002/docs/QAOOSE-Ber-Val.pdf.

Bryson, N. (1996): *Group Decision-Making And The Analytic Hierarchy Process: Exploring The Consensus-Relevant Information Content*. Computers Ops Res. **23** (1), S. 27 – 35.

Burgues, X.; C. Estay; X. Franch; J. A. Pastor und C. Quer (2002): *Combined Selection of COTS Components*. In J. C. Dean und A. Gravel, Hrsg., *COTS-Based Software Systems, First International Conference, ICCBSS 2002, Orlando, FL, USA, February 4-6, 2002, Proceedings* Band 2255 von *Lecture Notes in Computer Science* S. 54 – 64. Springer Verlag, Berlin, u.a.

Carmone, F. J.; A. Kara und S. H. Zanakis (1997): *A Monte Carlo investigation of incomplete pairwise comparison matrices in AHP*. European Journal of Operational Research **102**, S. 538 – 553.

Comella-Dorda, S.; J. C. Dean; E. Morris und P. Oberndorf (2002): *A Process for COTS Software Product Evaluation*. In J. C. Dean und A. Gravel, Hrsg., *COTS-Based Software Systems, First International Conference, ICCBSS 2002, Orlando, FL, USA, February 4-6, 2002, Proceedings* Band 2255 von *Lecture Notes in Computer Science* S. 86 – 96. Springer Verlag, Berlin, u.a.

ComponentManager (2003): *Homepage des Select Component Managers*. URL http://www.selectbs.com/products/products/component_manager.htm.

ComponentSource (2003): *Homepage des ComponentSource Komponentenmarktplatzes*. URL http://www.componentsource.com.

Dröschel, W. und M. Wiemers (1999): *Das V-Modell 97*. Oldenbourg.

Eisenführ, F. und M. Weber (2003): *Rationales Entscheiden*. 4. Auflage, Springer Verlag, Berlin, u.a.

Endres, A. (2003): *Softwarequalität aus Nutzersicht und ihre wirtschaftliche Bewertung*. Informatik Spektrum **26** (1), S. 20 – 25.

Eschinger, M.; F. Hieber und M. von Blohn (2001): *Variantenstücklisten als Hilfsmittel beim Einsatz von EJB-Fachkomponenten für die individuelle Konfiguration von Anwendungssystemen*. Seminararbeit an der Technischen Universität Darmstadt, FG Wirtschaftsinformatik I.

ExpertChoice (2003): *Homepage von ExpertChoice*. URL http://www.expertchoice.com.

Fettke, P.; P. Loos und B. Viehweger (2003): *Komponentenmarktplätze - Bestandsaufnahme und Typologie*. In K. Turowski, Hrsg., *Tagungsband 5. Workshop Komponentenorientierte betriebliche Anwendungssysteme* S. 1 – 15. Universität Augsburg.

Finan, J. S. und W. J. Hurley (1999): *Transitive calibration of the AHP verbal scale*. European Journal of Operational Research **112**, S. 367 – 372.

Forman, E. H. und S. I. Gass (2001): *The Analytic Hierarchy Process - An Exposition*. INFORMS **49** (4), S. 469 – 486.

Forman, E. H. und M. A. Selly (2001): *Decision By Objectives - How To Convince Others That You Are Right*. World Scientific Publishing Co. Pte. Ltd., Singapore u.a.

Frakes, W. B. und T. P. Pole (1994): *An Empirical Study of Representation Methods for Reusable Software Components*. IEEE Transactions on Software Engineering **20** (5), S. 617 – 630.

GartnerGroup (2003): *Homepage der Gartner Group - Informationen zu Decision Tools*. URL http://regionals4.gartner.com/4_decision_tools/measurement/decision_tools/decision_tools.html.

Hahn, H. (2002): *Ein Modell zur Ermittlung der Reife des Softwaremarktes*. In K. Turowski, Hrsg., *Tagungsband 4. Workshop Komponentenorientierte betriebliche Anwendungssysteme* S. 75 – 85. Universität Augsburg.

Haines, G.; D. Carney und J. Foreman (1997): *Component-Based Software Development / COTS Integration*.
URL http://www.sei.cmu.edu/str/descriptions/cbsd_body.html.

Hansen, W. J. (2001): *A generic process and terminology for evaluating cots software*.
URL http://www.sei.cmu.edu/staff/wjh/Qesta.html.

Henninger, S. (1994): *Using Iterative Refinement to Find Reusable Software*. IEEE Software S. 48 – 59.

Jacobson, I.; G. Booch und J. Rumbaugh (1999): *The Unified Software Development Process*. Addison-Wesley.

Jilani, L. L.; R. Mili und A. Mili (1997): *Approximate Component Retrieval: An Academic Exercise or a Practical Concern?*. In *Proceedings of 8th Workshop on Institutionalizing Software Reuse (WISR8)*. Columbus, Ohio.
URL http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers/jilani/jilani.html.

Kiniry, J. R. (1999): *Leading to a Kind Description Language: Thoughts on Component Specification*. Caltech Technical Report CS-TR-99-04, Department of Computer Science, California Institute of Technology.
URL http://www.infospheres.caltech.edu/papers/cs-tr-99-04.pdf.

Kolisch, R. und K. Hempel (1996): *Auswahl von Standardsoftware, dargestellt am Beispiel von Programmen für das Projektmanagement*. Wirtschaftsinformatik **38** (4), S. 399 – 410.

Kontio, J. (1995): *OTSO: A Systematic Process for Reuseable Software Component Selection*. Institute for Advanced Computer Studies and Department of Computer Science. University of Maryland, Collage Park, USA.

Kontio, J. (1996): *A Case Study in Applying a Systematic Method for COTS Selection*. In *Proceedings: 18th International Conference on Software Engineering in Berlin*. IEEE.

Kontio, J.; G. Caldiera und V. Basili (1996): *Defining Factors, Goals and Criteria for Reuseable Component Evaluation*. Presented at the CASCON 96 conference, Toronto, Canada.

Krammer, A. und J. M. Zaha (2003): *Komponentenfindung in monolithischen betrieblichen Anwendungssystemen*. In K. Turowski, Hrsg., *Tagungsband 5. Workshop Komponentenorientierte betriebliche Anwendungssysteme* S. 155 – 166. Universität Augsburg.

Kunda, D. und L. Brooks (1999): *Case study: Identifying factors that support COTS component selection*. Requirenautics Quarterly, RESG of the British Computer Society **17**, S. 7 – 9.
URL http://wwwsel.iit.nrc.ca/projects/cots/icsewkshp/papers/kunda.pdf.

Lalanda, P. (1998): *A Control Model for the Dynamic Selection and Configuration of Software Components*. In J. Bosch und S. Mitchell, Hrsg., *Object-Oriented Technology, ECOOP'97 Workshop Reader, ECOOP'97 Workshops, Jyväskylä, Finland, June 9-13, 1997* Band 1357 von *Lecture Notes in Computer Science* S. 343 – 347. Springer Verlag, Berlin, u.a.

Larsson, M. und I. Crnkovic (2000): *Component Configuration Management*. In C. Szyperski, Hrsg., *Proceedings of WCOP 2000*.
URL http://www.mrtc.mdh.se/publications/0236.pdf.

Maiden, N. A. M.; H. Kim und C. Ncube (2002): *Rethinking Process Guidance for Selecting Software Components*. In J. C. Dean und A. Gravel, Hrsg., *COTS-Based Software Systems, First International Conference, ICCBSS 2002, Orlando, FL, USA, February 4-6, 2002, Proceedings* Band 2255 von *Lecture Notes in Computer Science* S. 151 – 164. Springer Verlag, Berlin, u.a.

McIlroy, M. D. (1968): *Mass Produced Software Components*. In P. Naur und B. Randell, Hrsg., *Software Engineering: Report on a Conference by the NATO Science Commitee* S. 138 – 150. NATO Scientific Affairs Division, Brussels.

Meling, R.; E. J. Montgomery; P. S. Ponnusamy; E. B. Wong und D. Mehandjiska (2000): *Storing and Retrieving Software Components: A Component Description Manager*. In *Proceedings of the Australian Software Engineering Congress (ASWEC) 2000*.
URL http://www.it.bond.edu.au/jmontgomery/pubs/aswec_02-2000.pdf.

Merad, S. und R. de Lemos (1999): *Solution Concepts for the Optimal Selection of Software Components*. In *Proceedings: International Workshop on Component-Based Software Engineering* S. 169 – 174.
URL http://www.sei.cmu.edu/cbs/icse99/papers/icse99-papers.pdf.

Mili, H.; F. Mili und A. Mili (1995): *Reusing Software: Issues and Research Directions*. IEEE Transactions on Software Engineering **21** (6), S. 528 – 561.

Millet, I. und T. L. Saaty (2000): *On the relativity measures - accommodating both rank preservation and rank reversals in the AHP*. European Journal of Operational Research **121**, S. 205 – 212.

Mustajoki, J. und R. P. Hämäläinen (2002): *WEB-HIPRE - A JAVA APPLET FOR AHP AND VALUE TREE ANALYSIS*. Systems Analysis Laboratory. Helsinki University of Technology.
URL http://www.hipre.hut.fi.

Ncube, C. und J. C. Dean (2002): *The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components*. In J. C. Dean und A. Gravel, Hrsg., *COTS-Based Software Systems, First International Conference, ICCBSS 2002, Orlando, FL, USA, February 4-6, 2002, Proceedings* Band 2255 von *Lecture Notes in Computer Science* S. 176 – 187. Springer Verlag, Berlin, u.a.

Ncube, C. und N. A. M. Maiden (1999): *PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm*.
URL http://www.sei.cmu.edu/cbs/icse99/papers/11/11.pdf.

Ochs, M.; D. Pfahl; G. Chrobok-Diening und B. Nothhelfer-Kolb (2000): *A COTS Acquisition Process: Definition and Application Experience*. ISERN Report 00-02.
URL http://www.iese.fhg.de/network/ISERN/pub/technical_reports/isern-00-02.pdf.

OMG (2001a): *The Common Object Request Broker: Architecture and Specification: Version 2.5*. OMG, Framingham.

OMG (2001b): *Unified Modeling Language Specification: Version 1.4*. OMG, Needham.

Ortner, E. (1997): *Methodenneutraler Fachentwurf: Zu den Grundlagen einer anwendungsorientierten Informatik*. Teubner, Stuttgart.

Ortner, E. (1998): *Ein Multipfad-Vorgehensmodell für die Entwicklung von Informationssystemen - dargestellt am Beispiel von Workflow-Management-Anwendungen*. Wirtschaftsinformatik **4** (40), S. 329 – 337.

Ortner, E. (1999a): *Repository Systems. Teil 1: Mehrstufigkeit und Entwicklungsumgebung*. Informatik Spektrum **22**, S. 235 – 251.

Ortner, E. (1999b): *Repository Systems. Teil 2: Aufbau und Betrieb eines Entwicklungsrepositoriums*. Informatik Spektrum **22**, S. 351 – 363.

Ortner, E.; K.-P. Lang und J. Kalkmann (1999): *Anwendungssystementwicklung mit Komponenten*. Information Management & Consulting **14** (2), S. 35 – 45.

Ostertag, E.; R. P. Diaz und C. Braun (1992): *Computing Similarity in a Reuse Library System: An AI-Based Approach*. ACM Transactions on Software Engineering and Methodology **1** (3), S. 205 – 228.

Overhage, S. (2002a): *Die Spezifikation - kritischer Erfolgsfaktor der Komponentenorientierung*. In K. Turowski, Hrsg., *Tagungsband 4. Workshop Komponentenorientierte betriebliche Anwendungssysteme* S. 1 – 17. Universität Augsburg.

Overhage, S. (2002b): *Komponentenkataloge auf Basis eines einheitlichen Spezifikationsrahmens - ein Implementierungsbericht*. In K. Turowski, Hrsg., *Modellierung und Spezifikation von Fachkomponenten. 3. Workshop* S. 1 – 16. Nürnberg.

Overhage, S. und P. Thomas (2003): *WS-Specification: Specifing Web Services using UDDI Improvements*. In A. B. Chaudhri; M. Jeckle; E. Rahm und R. Unland, Hrsg., *Web, Web-Services, and Database Systems* Band 2593 von *Lecture Notes in Computer Science* S. 100 – 119. Springer Verlag.

Phillips, B. C. und S. M. Polen (2002): *Add Decision Analysis to Your COTS Selection Process*. Software Technology Support Center Crosstalk.
URL http://www.software.org/pub/externalpapers/decisionAnalysis.pdf.

Prieto-Diaz, R. (1991): *Implementing Faceted Classification for Software Reuse*. Communications of the ACM **34** (5), S. 88 – 97.

Ritter, J. (2000): *Prozessorientierte Konfiguration komponentenbasierter Anwendungssysteme*. Dissertation, Universität Oldenburg, Fachbereich Informatik.
URL http://docserver.bis.uni-oldenburg.de/publikationen/dissertation/2000/ritpro00/ritpro00.html.

Ruhe, G. (2003): *Intelligent Support for Selection of COTS Products*. In A. B. Chaudhri; M. Jeckle; E. Rahm und R. Unland, Hrsg., *Web, Web-Services, and Database Systems* Band 2593 von *Lecture Notes in Computer Science* S. 34 – 45. Springer Verlag.
URL http://sern.ucalgary.ca/~ruhe/Research_Publications/Proceedings/nod7.pdf.

Saaty, T. L. (1980): *The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation*. McGraw-Hill, New york.

Saaty, T. L. (1994): *Highlights and critical points in the theory and application of the Analytic Hierarchy Process*. European Journal of Operational Research **74**, S. 426 – 447.

Schenkerman, S. (1994): *Avoiding rank reversal in AHP decision-support models*. European Journal of Operational Research **74**, S. 407 – 419.

Schneeweiß, C. (1999): *Einführung in die Produktionswirtschaft*. 7. Auflage, Springer Verlag, Berlin, u.a.

Shaw, M. und D. Garlan (1996): *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Englewood Cliffs, NJ.

Szyperski, C. (1998): *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Harlow.

Varadarajan, S.; A. Kumar; D. Gupta und P. Jalote (2002): ComponentXchange: An e-exchange for software components.
URL http://www.cse.iitk.ac.in/users/deepak/papers/ComponentXchange.pdf.

Vargas, L. G. (1990): *An overview of the Analytic Hierarchy Process and its applications*. European Journal of Operational Research **48**, S. 2 – 8.

Vargas, L. G. (1994): *Reply to Schenkerman's avoiding rank reversal in AHP decision-support models*. European Journal of Operational Research **74**, S. 420 – 425.

Wedekind, H. und T. Müller (1981): *Stücklistenorganisation bei einer großen Variantenzahl*. Angewandte Informatik **23** (9), S. 377 – 383.

Zimmermann, G. (1988): *Produktionsplanung variantenreicher Erzeugnisse mit EDV*. Springer Verlag, Berlin, u.a. Betriebs- und Wirtschaftsinformatik.