2 Einführung

2.1 Anwendungsgebiete linearer und gemischt-ganzzahliger Modelle

Ein lineares Modell besitzt folgende Grundform:

$$\min/\max \; z = c^T x$$

$$\text{so dass} \quad Ax \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b$$

$$\text{und} \quad lb \leq x \leq ub$$

Zur Lösung eines solchen Modell muss ein Lösungsvektor *x* ermittelt werden, der einen Zielfunktionswert *z* maximiert oder minimiert, unter den Nebenbedingungen eines (Un-)Gleichungssystems, das sich aus der Matrix *A* und dem Vektor *b* ergibt.

Eine Methode zur Lösung solcher Modelle ist das 1947 von George. B. Dantzig entwickelte Simplex-Verfahren (zur Entstehungsgeschichte vgl. [Dantzig02]), das eigentlich bereits 1939 von Kantorovich ([Kantov60]) beschrieben wurde, aber damals aufgrund der politischen Verhältnisse unbeachtet blieb. Die einzelnen Schritte des Verfahrens sowie Details und Erweiterungen finden sich bei Dantzig selbst (vgl. [Dantzig66] oder [Dantzig97]) sowie in den meisten einführenden OR-Lehrbüchern (z.B. [SuhlL06], [Vanderbei01] oder [Domschke05]).

In der Praxis kommt es jedoch häufig vor, dass fraktionale Lösungswerte eines Modells keinen Sinn ergeben. Beispielsweise wird man einer Modellempfehlung, dass der Einsatz von 1,3 Flugzeugträgen optimal sei, in der Realität nicht nachkommen können. Wie [Gomory91] beschreibt, waren es in den 1950er-Jahren, ähnlich wie beim Simplex, militärische Fragestellungen und Modelle, die die Einbeziehung von ganzzahligen Werten in die Modellierung erforderlich machten. Sind alle oder einige der Entscheidungsvariablen des Modells ganzzahlig, so spricht man von IP- bzw. MIP-Modellen. Mithilfe ganzzahliger Variablen in Form von Binärvariablen lassen sich komplexe logische Bedingungen formulieren, und nicht-lineare Zusammenhänge können linearisiert werden (vgl. [Padberg00]). MIP-Modelle[1] bilden eine eigene Klasse mit eigenen, oft sehr rechenintensiven Lösungsalgorithmen, wie *Branch & Bound* oder *Branch & Cut*. Grundlegende Algorithmen zu MIP-Optimierungsverfahren findet man in den meisten OR-Lehrbüchern, wie z.B. in [SuhlL06], [Hillier01] oder [Dürr02], detail-

1 Wenn in dieser Arbeit von MIP-Modellen die Rede ist, so schließt dies die reinen ganzzahligen Modelle (IP) mit ein.

lierte und stärker theoretisierende Betrachtungen bietet indessen der Klassiker von Nemhauser und Wolsey ([Nemhauser88]).
Auch wenn die Möglichkeit der Formulierung lediglich linearer Zusammenhänge bei LP- und MIP-Modellen auf den ersten Blick einschränkend wirken mag, so eröffnet die rechnergestützte Optimierung dieser Modelle dennoch in der Praxis eine große Zahl von Anwendungsmöglichkeiten:

Ernährungsplanung. Einer der klassischen Anwendungsfälle der linearen Programmierung sind so genannte *Diet Problems*. Dabei geht es darum, einen kostenminimalen Ernährungsplan aus verschiedenen Lebensmitteln zusammenzustellen, so dass bestimmte Mindestmengen an Nährstoffen und Vitaminen enthalten sind. Dantzig selbst beschreibt ein solches Diet Problem inklusive seiner Selbstversuche augenzwinkernd in [Dantzig90]. Ein größeres Modell für die Futterzusammenstellung in der Rinderzucht wird dargestellt in [McGillivray84] (zusammengefasst in [Bell99, S. 119-120]).

Flugplanung. [SuhlL95] beschreibt den Einsatz von Optimierungsmethoden in der Flugplanerstellung und Flugrotationsplanung, wo kostenminimale bzw. profitmaximale Pläne unter Berücksichtigung diverser Restriktionen, wie vorhandene Flughafen-Slots, Flugzeugtypen, Passagierzahlen etc., erstellt werden, ähnlich auch [Desrosiers00] für die Flugrouten- und Personaleinsatzplanung einer Charterfluggesellschaft. Der Optimierung der Crew-Einsatzplanung, insbesondere im Fall von Störungen des ursprünglichen Einsatzplans, widmet sich [Gou05].

Verschnittoptimierung. Bereits Anfang der 1960er-Jahre beschrieben Gilmore und Gomory die Anwendung der linearen Optimierung unter Anwendung eines Column-Generation Algorithmus für die Verschnittoptimierung in der Papierindustrie ([Gilmore61]), ein Ansatz, der mehrfach aufgegriffen und u.a. von [Farley88] verbessert bzw. generalisiert wurde.

Gaspipelineoptimierung. Im gaswirtschaftlichen Bereich existieren viele Einsatzmöglichkeiten linearer Programmierung. Oft ist es dabei allerdings erforderlich, nicht-lineare, insbesondere quadratische Zusammenhänge (Druck, Brennwerte etc.) zu linearisieren. Als Beispiel hierfür können etwa die von [Aalto05] beschriebenen Echtzeitoptimierungsprobleme im Finnischen Ferngasnetz angeführt werden.

Finanzplanung. Im Bereich der Finanzierungs- und Investitionsentscheidungsunterstützung findet man ebenfalls eine Reihe von Problemen, die sich mit linearer Programmierung lösen lassen. So beschreibt etwa [Cornuéjols06, S. 47 ff.], wie sich über ein LP-Modell ein optimales kurzfristiges Cashflow-Matching bestimmen lässt. Dabei geht es um die optimale Finanzierung des Cashflows eines Unternehmens aus unterschiedlichen Quellen (Anleihen, Wechsel, Kredite etc.). Ein weiteres Beispiel ist die Aufdeckung von Arbitragemöglichkeiten mithilfe eines LP-Modells (vgl. [Cornuéjols06, S. 69 ff.]).

Losgrößenplanung. Unter Losgröße versteht man die Anzahl der Teile, die geschlossen durch eine Fertigung laufen. Die Bestimmung einer optimalen Lösgröße dient der Minimierung der Produktionskosten unter Berücksichtigung teils konträrer Faktoren wie z.B. Lagerkosten, degressive Produktionsfixkosten, Kapitalbindung, Durchlaufzeit, Verderb etc. In modernen Produktionsplanungs- und -steuerungssystemen (PPS) kann die Bestimmung der optimalen Losgröße über ein MIP-Modell erfolgen (vgl. beispielsweise [Jacobsen98]).

Sortimentsplanung. Bei der operativen Sortimentsplanung im Einzelhandel wird anhand bestimmter Regeln eine optimale Zusammenstellung und Anordnung von Waren in den Verkaufsregalen bestimmt. Ein Beispiel ist das auf der Optimierungsengine MOPS basierende System *HerCon* der Herlitz AG (vgl. [Suhl01], [Herrn00], [Syring03]).

Raffinerieplanung. In der Erdöl verarbeitenden Industrie gibt es, wie in anderen Zweigen der chemischen Industrie auch, eine Reihe von Anwendungsmöglichkeiten für lineare und gemischt-ganzzahlige Optimierung, seien es Mischungsprobleme (z.B. [Dantzig97, S. 26 ff.]), Probleme der Raffinerie internen Logistik (vgl. [Lee96]), oder Raffinerie bezogene Scheduling-Probleme (vgl. [Kelly03]). Eine Reihe weiterer Anwendungen aus dem Raffineriebereich wird in [Sahdev06] aufgeführt.

Die genannten Beispiele umfassen nur einen kleineren Ausschnitt aus dem breiten Anwendungsbereich der linearen und gemischt-ganzzahligen Optimierung. Eine Anzahl weiterer Anwendungsmöglichkeiten findet sich (meist in Form von Fallstudien) in [Bell99] oder in Form vereinfachter Beispiele in [Williams85]. Als Fachzeitschrift widmet sich *Interfaces* hauptsächlich praktischen Anwendungsfällen für OR-Methoden.

Allen Praxisanwendungen gemein ist, dass die Modelle nicht im leeren Raum stehen, sondern innerhalb Entscheidungsunterstützender Systeme (EUS) generiert und gelöst werden, wobei die Gestalt solcher Systeme vom einfachen Spreadsheet bis zu komplexen, problemspezifi-

schen Applikationen reichen kann. In der Regel lösen EUS die Modelle nicht selbst, sondern benutzen eingebundene Optimierungsengines als Fremdmodule. Die Einbindung hat Einfluss auf die Art und Weise der Modellgenerierung, Lösungsprozesssteuerung und Modellanalyse und betrifft dadurch auch die Software architektonische Qualität des EUS.
Gleichwohl sind die Probleme der Einbindung und die Frage der Beschaffenheit der Schnittstellen zur Optimierungssoftware bisher kaum systematisch dargestellt worden. Ebenso weisen viele Optimierungsengines, insbesondere solche aus dem akademischen und semikommerziellen Bereich, größere Schwächen bei ihren Einbettungsmöglichkeiten und Schnittstellenkonzepten auf. Dies bildet Motivation und Ausgangspunkt für die im Folgenden vorgestellten Überlegungen, Konzepte und Implementationen.

2.2 Architektur Entscheidungsunterstützender Systeme (EUS)

2.2.1 Entscheidungsunterstützende Systeme

Das Fällen von (rationalen) Entscheidungen ist eine wichtige Managementtätigkeit, die sich durch alle Ebenen der Unternehmung zieht. Die Vertreter der Entscheidungsorientierten Betriebswirtschaftslehre (z.B. [Heinen76], [Heinen91]) betrachten Entscheiden sogar als *die* zentrale Managementaufgabe schlechthin, weshalb Entscheidungen auch die eigentlichen Quellen und Ursachen für Kosten, Erfolg und Liquidität sind (für eine entscheidungsorientierte Sicht des Rechnungswesens vgl. [Riebel92]). Dabei können die zugrunde liegenden Entscheidungsprozesse als „Transformationen von Information in Aktionen“ ([Holten97, S. 3]) aufgefasst werden. Information als Rohmaterial dieses Transformationsprozesses ist ein wichtiger Wettbewerbsfaktor, und Informationsasymetrien zwischen Unternehmen können deren Erfolg oder Misserfolg begründen (vgl. [Picot93]).
Allerdings muss die Information in einem für den Entscheidungsträger erfassbaren Zustand, als „zweckbezogenes Wissen“ ([Wittmann59, S. 14]), vorliegen, um als Grundlage für Entscheidungen und damit Aktionen zu dienen. Entscheidungsunterstützende Systeme übernehmen hierbei die Aufgabe der Informationsveredlung durch Verdichtung, Umwandlung oder Berechnung.
Entscheidungsunterstützende Systeme existieren in großer Anzahl und Verschiedenheit und bilden eine Teilmenge der betrieblichen Informationssysteme. In der Literatur findet man zuweilen die Skizzierung einer Informationssystempyramide (z.B. [Jahnke93]), derzufolge Entscheidungsunterstützende Systeme (EUS) nur für die mittlere Managementebene gedacht sind, während die Versorgung der Führungskräfte der oberen Führungsebene durch spezielle *Führungsinformationssysteme* (*FIS*, engl. *Executive Information Systems*/*EIS*) erfolgt. Für die

operativen Managementebenen sieht das Informationssystempyramidenmodell *Management Informationssysteme* (*MIS*) und für die unterste operative Ebene so genannte *Transaktionsverarbeitungssysteme* (*TVS*, engl. *Transaction Processing Systems*/*TPS*) vor. Einer solchen Einschränkung des Begriffes EUS nur auf Systeme für die mittlere Managementebene soll hier ausdrücklich nicht gefolgt werden, denn im Sinne einer entscheidungsorientierten Sicht des Unternehmens werden auf allen Ebenen Entscheidungen getroffen, nur dass der Aggregationsgrad der zugrunde liegenden Informationen und die Reichweite der Entscheidungen je nach Ebene unterschiedlich sind. Wenn in dieser Arbeit daher von Entscheidungsunterstützenden Systemen die Rede ist, so kann es sich um Applikationen der operativen Ebene handeln (z.B. Verschnittoptimierungssysteme), aber auch um Anwendungen, die Entscheidungen der oberen Führungsebene betreffen (z.B. Lagerstandortsplanung). Typischerweise hat der Output eines EUS einen Empfehlungscharakter für Entscheidungen (z.B. „Verschnittmuster x ist kostenminimal - deshalb sollte so geschnitten werden.“), was für den Output anderer betrieblicher Informationssysteme nicht notwendigerweise gilt (z.B. Output eines datenverdichtenden Systems: „Der Gesamtumsatz aller Filialen beträgt x Euro.“).

Betriebliche Informationssysteme können datenorientiert, modellorientiert oder wissensbasiert sein. Datenorientierte Systeme beruhen häufig auf Konzepten wie *Data Mining* (vgl. [Han06]) oder *Online-Analytical-Processing (OLAP)* (vgl. [Clausen98]) und beziehen ihre Ausgangsdaten aus einem *Data-Warehouse* (vgl. [Gray98]). Wissensbasierte Systeme haben ihre Wurzeln im Bereich der Künstlichen Intelligenz (KI) und der Experten-Systeme, daher nennt man die Kombination aus *Decision Support Systems* und *Expert Systems* auch *Expert Support Systems* ([Holten97, S. 8). Modellorientierte Systeme generieren ihre entscheidungsunterstützenden Aussagen auf der Grundlage von Modellen. Dies können entscheidungstheoretisch basierte Modelle (vgl. etwa [Laux98]), Simulationsmodelle (vgl. [SuhlL06, Kap. 9]) oder mathematische Optimierungsmodelle sein, wobei sich diese Arbeit auf Letztere fokussiert.

Was die Architektur Entscheidungsunterstützender Systeme, bzw. allgemeiner betrieblicher Informationssysteme anbelangt, so bietet die Literatur der Wirtschaftsinformatik mehrere Architekturvorschläge, die sich zuweilen weniger durch Software architektonische Details, als durch die Betrachtungsperspektive unterscheiden. So ist das Modell der *ganzheitlichen Informationssystems-Architektur* von Krcmar ([Krcmar00, S. 30-31]) stärker organisationswissenschaftlich ausgerichtet, während die Sichtweise Scheers (vgl. [Scheer92]) im Modell *ARIS (Architektur Integrierter Informationssysteme)* stärker technisch ist. Da diese Modelle aber aufgrund ihrer ganzheitlichen Perspektive und ihres hohen Abstraktionsniveaus hier wenig brauchbar sind, soll in einer eigenen, grob an [SuhlL06, S. 90] angelehnten Darstellung das

Architekturschema eines typischen optimierungsmodellbasierten EUS skizziert werden (Abbildung 1).

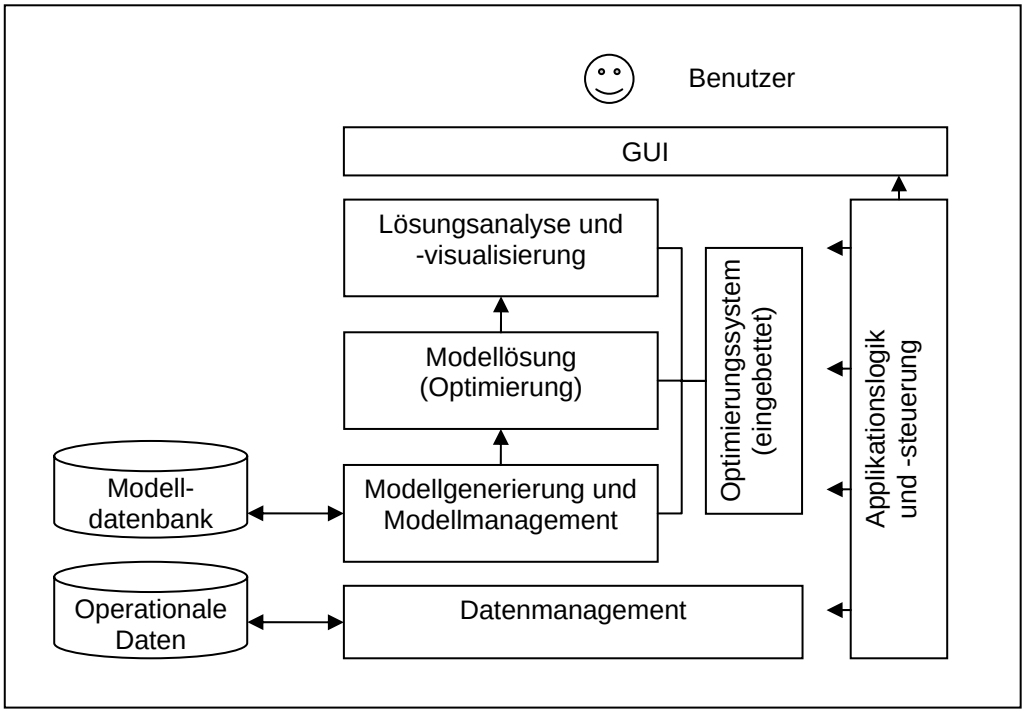

Abbildung 1: Architekturschema optimierungsmodellbasiertes EUS

Die Architektur gliedert sich in unterschiedliche Schichten und Module und enthält ein eingebettetes Optimierungssystem. Die unterste Schicht bildet das Datenmanagement, dessen Hauptaufgabe es ist, das EUS mit operationalen Ausgangsdaten zu versorgen, hierzu müssen beispielsweise Schnittstellen zu einem evtl. vorhandenen betrieblichen Data-Warehouse und sonstigen Datenbanken existieren. Auch kann es sein, dass das EUS eine eigene Datenhaltung als Zwischenspeicher besitzt, die von der Datenmanagementschicht verwaltet wird. Die Dienste der Datenmanagementschicht stehen allen anderen Modulen bidirektional zur Verfügung, so dass nicht nur Daten zur Modellgenerierung extrahiert werden können, sondern beispielsweise auch das Zurückschreiben von Optimierungsergebnissen und -analysen möglich ist. Die Applikationslogik steuert und koordiniert als verbindende Schicht alle Module des EUS. Der Benutzer interagiert mit dem System meist nur über eine grafische Benutzeroberfläche (GUI). Viele Prozesse innerhalb des EUS sind für ihn unsichtbar – unter Umständen weiß er nicht einmal, dass im Inneren des EUS ein Optimierungsmodell gelöst wird. Das eingebettete Optimierungssystem unterstützt mindestens den Prozess der Modellösung, kann darüber hinaus aber auch Funktionalitäten für Modellgenerierung und Lösungsanalyse bereitstellen. Während Module zur Modellösung (Solver) aufgrund der hohen Komplexität fast

immer Produkte von spezialisierten Fremdanbietern sind, werden die Module zur Modellgenerierung und zur Lösungsanalyse normalerweise als proprietäre Bausteine realisiert, die aber evtl. vom Optimierungssystem angebotene Hilfsfunktionalitäten nutzen.
Die beschriebene Architektur ist zwar eine typische, aber keineswegs die einzig mögliche. Ein EUS kann beispielsweise auch nur aus einem Spreadsheet mit entsprechender Solveranbindung (z.B. *CLIPMOPS* vgl. [Suhl06a]) bestehen. Unterschiedliche EUS-Architekturen bedingen allerdings auch unterschiedliche Einbettungsmechanismen und Schnittstellen. Optimierungssysteme müssen daher so konzipiert sein, dass sie in unterschiedlichen Einbettungsszenarien benutzt werden können.

2.2.2 Integration von Optimierungsfunktionalität in EUS

Ein Entscheidungsunterstützendes System, das Informationen auf Basis der Lösung linearer und gemischt-ganzzahliger Modelle aufbereitet, muss diese Modelle einem Optimierer übergeben, wobei prinzipiell unterschiedliche Wege bzw. Schnittstellen existieren. Abbildung 2 stellt diese unterschiedlichen Integrationsmöglichkeiten für Optimierer dar und schematisiert die Benutzerinteraktion.

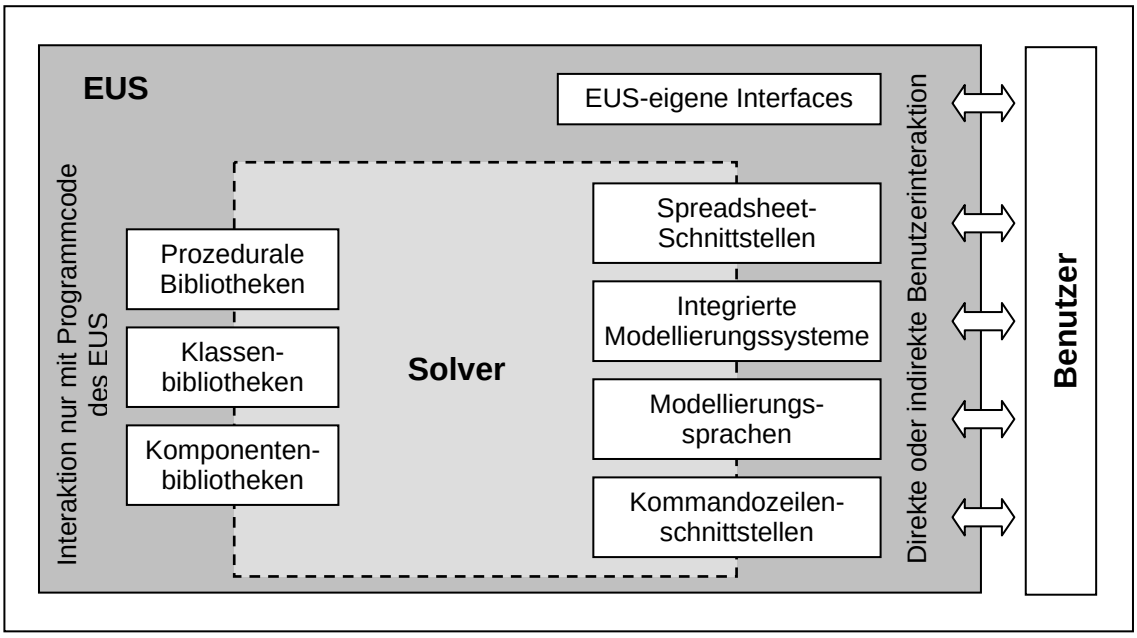

Abbildung 2: Solvereinbindung in EUS

Die einfachste und älteste Möglichkeit, mit einem Solver zu arbeiten, ist eine **Kommandozeilenschnittstelle**. Fast alle Optimierer verfügen über diese Möglichkeit, sei es in Form des Aufrufs einer ausführbaren Datei, die ggf. unter Angabe von Parametern unmittelbar mit dem Start den Optimierungslauf beginnt, oder sei es in Form einer textbasierten Dialogschnittstel-

le, über die nicht nur die Optimierung gestartet wird, sondern auch z.B. diverse Einstellungen für den Solver vorgenommen werden können. Der Benutzer arbeitet entweder direkt mit einer solchen Schnittstelle, dann hat das EUS lediglich Hintergrundfunktion (z.B. vorherige Aufbereitung von MPS-Dateien etc.), oder das EUS benutzt selbst die Kommandozeilenschnittstelle, indem es beispielsweise Batchdateien erzeugt und diese zur Ausführung bringt. Die Benutzung einer Kommandozeilenschnittstelle dürfte jedoch für die meisten EUS – insbesondere für diejenigen neueren Datums – der Ausnahmefall sein, da die Einsatzmöglichkeiten einer solchen Schnittstelle im Vergleich zu anderen sehr begrenzt sind.

Eine weitere Möglichkeit, einen Solver innerhalb eines Entscheidungsunterstützenden Systems zu benutzen, ist der Weg über **Modellierungssprachen**[1]. Auch hier kann der Benutzer entweder direkt mit der Modellierungssprache arbeiten, oder die Benutzung erfolgt indirekt über das EUS. Im ersten Falle wird, wie z.B. bei AMPL (vgl. [Fourer03a]), mit einer textbasierten Dialogschnittstelle gearbeitet, in der Statements der Modellierungssprache und Steuerungsanweisungen direkt eingegeben werden können. Im zweiten Fall, der für EUS der Normalfall ist, übernimmt das Entscheidungsunterstützende System die Steuerung und die Kommunikation mit der Modellgenerierungssprache, wozu insbesondere die kommerziellen Optimierungspakete spezifische Schnittstellenbibliotheken anbieten. Beispiele sind die Einbettungsbibliotheken für *XPRESS-MOSEL* ([Dash06d]) oder für *OPL* ([ILOG06e]).

Eine dritte Möglichkeit der Solverinteraktion ist die Benutzung eines **Integrierten Modellierungssystems**. Derartige Systeme verfügen über grafische Benutzeroberflächen und beinhalten meist eine Modellgenerierungssprache. Sie unterstützen Modellverwaltung und Datenbankanbindungen und verfügen über Funktionalitäten zur grafischen oder reportbasierten Aufbereitung der Optimierungsergebnisse. Dabei interagiert der Benutzer direkt mit der Oberfläche des Modellierungssystems, ein vorhandenes EUS kann eventuell im Hintergrund z.B. zur Datenaufbereitung und -lieferung wirken. Beispiele sind die kommerziellen Umgebungen *XPRESS-IVE* ([Dash06c]) und *OPL-Studio* ([ILOG06d]). Ein anderer Typus der integrierten Modellierungssysteme stellt dem Entwickler eine Vielzahl von Softwarebausteinen (inkl. GUI, Reporting etc.) zur Konstruktion eines EUS mit mathematischer Modellierungsfunktionalität zur Verfügung, so dass das resultierende Entscheidungsunterstützende System ein Konstrukt auf Basis des Modellierungssystems ist. Herausragendes Beispiel hierfür ist AIMMS ([Paragon06]).

Für kleinere Modelle bieten sich auf **Tabellenkalkulationsprogrammen** basierende EUS an. Eine Vielzahl von Anwendungsmöglichkeiten spreadsheetbasierter Modellierung und Ent-

[1] Für einen einführenden Überblick zum Thema Modellierungssprachen siehe [Fragnière02].

scheidungsanalyse findet sich beispielsweise in [Ragsdale04]. Im Fall des marktbeherrschenden Microsoft Produkts *Excel* wird dabei die Optimierungsengine oft in Form eines Add-Ins eingebunden, und weitere EUS-Funktionalitäten sind meist in der objektbasierten Scriptingsprache *Visual Basic for Applications* (*VBA*) implementiert. Eine ausführliche Darstellung der Verwendung von VBA in Entscheidungsunterstützenden Systemen findet man in [Albright06]. Vorteil dieses Ansatzes ist, dass eine Reihe von Grundfunktionalitäten bereits aus dem Tabellenkalkulationspaket vorhanden ist und sich weitere Funktionalitäten durch den Rapid Application Development Charakter der Sprache VBA sehr schnell implementieren lassen (zu Rapid Application Development unter VB vgl. [McMahon00]). Nicht zuletzt aus Performancegründen eignet sich eine spreadsheetbasierte Schnittstelle zu Entscheidungsunterstützenden Systemen aber in der Regel nur für kleinere Modelle. Ein Beispiel für einen Optimierer mit Excel-Add-In-Schnittstelle ist *CLIPMOPS* ([Suhl06b]).

Bei den drei übrigen, in Abbildung 2 links dargestellten Schnittstellentypen handelt es sich um Interfaces, die nur aus dem Programmcode des EUS angesprochen werden. Fast alle Solver verfügen über **prozedurale Schnittstellen** in Form von statischen oder dynamischen Callable Libraries zur Einbindung in andere Programme (s. Abschnitt 3.1). Daneben besitzen einige Optimierer auch objektorientierte Schnittstellen in Form von **Klassen- oder Komponentenbibliotheken** (s. Abschnitt 3.2). Ein interessanter Ansatz in diesem Zusammenhang ist auch das *Open Solver Interface* (*OSI*, [OSI06]), das eine objektorientierte Middleware zwischen Anwendung und Optimierungsengine bildet. Das uniforme Interface in Form einer Open-Source-Klassenbibliothek ermöglicht die Benutzung einer Anzahl unterschiedlicher Solver, ohne dass die Client-Anwendung eine spezifische Anpassung an die jeweilige Solverschnittstelle erfahren müsste.

Solver und Schnittstellen müssen ein kohärentes, aber flexibles System bilden, das dem Benutzer und Entwickler den situativ bestmöglichen Zugang zu einem Optimierer ermöglicht. Die Erstellung eines Konzepts für ein solches integriertes Schnittstellensystem – hier am Beispiel des Optimierers MOPS – ist eines der Ziele dieser Arbeit und soll in Kapitel 5 mit konkreten Inhalten gefüllt werden.

2.2.3 Umfeldfunktionalitäten eines Optimierungssystems

Ein modernes Optimierungssystem besteht nicht nur aus einer leistungsfähigen Optimierungsengine im Kern, sondern bietet noch eine Reihe weiterer Funktionalitäten, welche die Einbettung in eine Applikation erleichtern. Diese Funktionalitäten, die hier als „Umfeldfunktionalitäten“ bezeichnet werden sollen, umfassen die in Abbildung 3 genannten Bereiche. So verfügen einige Optimierungsbibliotheken (z.B. die *ILOG Concert Technology Library*) etwa über spezifische Klassen für Arrays und andere Datenstrukturen, die die Datenverwaltung in einem Modellgenerator erleichtern sollen. Einige Systeme bieten Anbindungsunterstützung für externe Applikationen, wie z.B. die Excel-Anbindung der *MPL OptiMax* Bibliothek ([Maximal06]), oder sie erleichtern bestimmte, in Optimierungsanwendungen häufig benötigte I/O-Aufgaben, wie etwa die XML- und CSV-Reader-Klassen der *Concert Technology Library*. Ebenso findet man Unterstützung für Datenbankanbindungen und für die Ausgabe von Daten und Lösungsergebnissen (Reporting). Das Optimierungssystem AIMMS ([Bisschop04]) geht bei der Ausdehnung der Umfeldfunktionalitäten sogar so weit, einen eigenen GUI-Builder für die Gestaltung der grafischen Benutzeroberfläche zur Verfügung zu stellen.

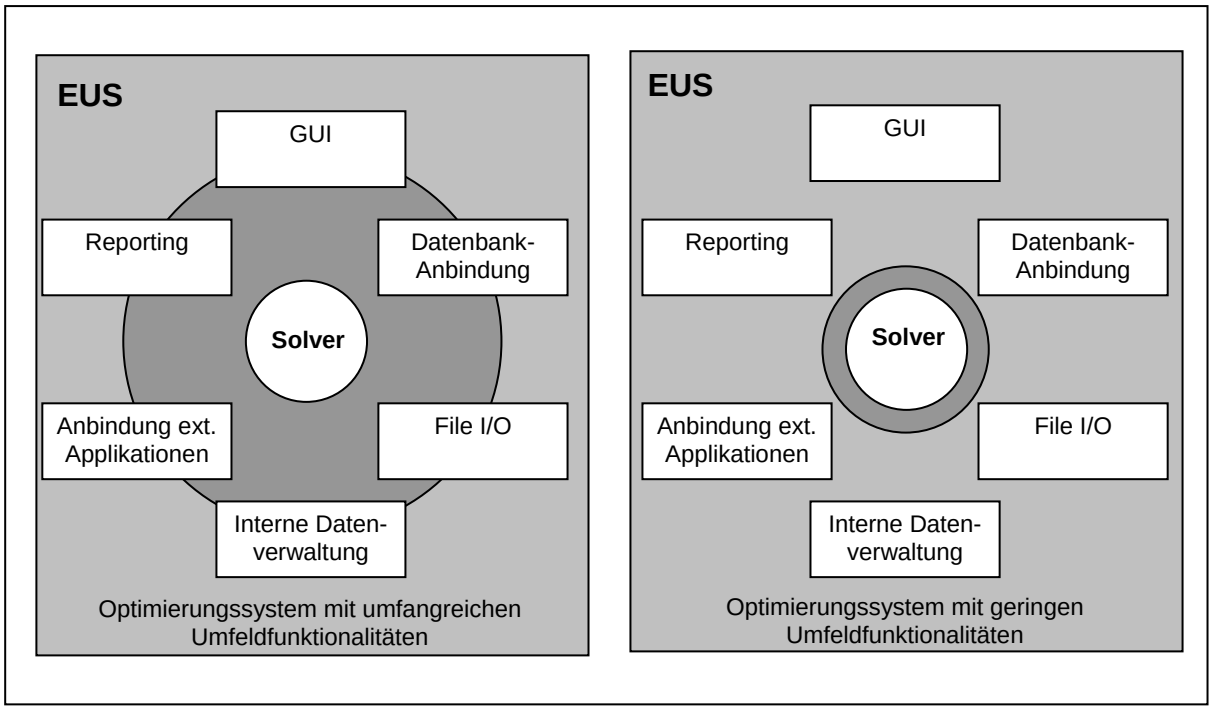

Abbildung 3: Umfeldfunktionalitäten eines Optimierungssystems

In dieser Arbeit wird die Auffassung vertreten, dass die Leistungsfähigkeit der Umfeldfunktionalitäten und der Schnittstellen einen ähnlich großen Einfluss auf den – insbesondere kommerziellen – Erfolg eines Optimierungssystems hat wie die numerische Performance des

Solvers. Diese Ansicht ist keineswegs neu und wurde bereits vor über 25 Jahren unter anderem in [Krabek80, S. 20-21] geäußert:

> *"We do not feel that the linear programming user's most pressing need over the next few years is for a new optimizer that runs twice as fast on a machine that costs half as much […] The process required to manage the data, formulate and build the model, report on and analyze the results costs far more, and is much more of a barrier to effective use of LP, than the cost/performance of the optimizer. […]*
> *In the future we feel that the payoff will be much greater if we devote substantial efforts to those areas that will increase the productivity of the modeller and improve his ability to control the modelling process and to understand and present the results obtained from the model".*

Die Möglichkeit, durch gutes Schnittstellendesign und mächtige Umfeldfunktionen größere Produktivitätszuwächse in der Entwicklung optimierungsbasierter EUS zu erzielen, ist eine maßgebliche Motivation für diese Arbeit. Aus wissenschaftlicher Sicht ist dieses Thema bisher verhältnismäßig wenig behandelt worden, möglicherweise, weil ein stark mathematisch ausgerichteter Teil der OR-Welt schon früh die Entwicklung von Matrix- und Reportgeneratoren als „not academically respectable" ([Rice79, S. 23]) abgetan hatte und auch später die Entwicklung von Schnittstellen und Umfeldfunktionalitäten eher kommerziellen Anbietern überließ. Nichtsdestotrotz ist eine systematische Ausarbeitung des Themas, insbesondere aus der Sicht des Wirtschaftsinformatikers, aufgrund der sich eröffnenden Produktivitätspotenziale relevant.

2.2.4 Schnittstellenproblematik

Welche der in 2.2.2 genannten Schnittstellen zur Integration eines LP-/MIP-Solvers in ein Entscheidungsunterstützendes System benutzt wird, hängt von unterschiedlichen Kriterien ab:

- Verfügbarkeit

  Dieses an sich triviale und selbstverständliche Kriterium besagt, dass der eingesetzte Optimierer auch über die gewünschte Schnittstelle verfügen muss. Ist das nicht der Fall, kann ggf. eine Interface-Middleware, wie das *Open Solver Interface* ([OSI06]), eingesetzt werden.

- Programmiersprache des EUS

  Die Sprache, in der der modellgenerierende und -verwaltende Teil des EUS geschrieben ist, hat ganz maßgeblichen Einfluss auf die in Frage kommenden Solverinterfaces. So ist klar, dass objektorientierte Klassen- oder Komponentenschnittstellen nicht einsetzbar sind, wenn die Sprache des Modellgenerators nicht objektorientiert ist. Ebenso können Klassenbibliotheken, die es für C++ und Java gibt, nicht aus

Interpretersprachen wie Visual Basic 6 benutzt werden. Was Komponentenbibliotheken anbelangt, so ist es zwar über bestimme Brückentechnologien mit kleineren Einschränkungen möglich, einen COM-Server durch einem .NET-Client und umgekehrt zu benutzen (vgl. [Troelsen02]), jedoch wird man aus Konsistenzgründen in der Regel eine dem Basissystem entsprechende Komponententechnologie wählen. Die Benutzung prozeduraler Interfaces aus objektorientierten Basissprachen ist hingegen möglich und zuweilen sogar unumgänglich, da bei manchen Optimierungssystemen nur die prozeduralen Bibliotheken das komplette Funktionalitätsspektrum des Solvers bieten (etwa bei XPRESS-MP, s. Kapitel 3).

- Modellgröße und -komplexität

  Insbesondere für kleinere Modelle, deren Erstellung einen gewissen Ad-hoc-Charakter aufweist, eigenen sich Spreadsheet-Interfaces. Integrierte Modellierungsumgebungen sind hingegen für kleinere und größere Modelle gleichermaßen geeignet. Im Fall sehr großer Modelle, die nicht über eine Modellierungssprache, sondern durch einen eigenen Modellgenerator erstellt werden, empfiehlt sich die Programmierung in einer objektorientierten Sprache und damit die Benutzung objektorientierter Solverinterfaces, was helfen kann, die Komplexität handhabbar zu machen.

- Performance der Modellgenerierung

  Normalerweise fällt die Laufzeit der Modellgenerierung im Verhältnis zur Laufzeit des Lösungsvorgangs kaum ins Gewicht, jedoch kann es sein, dass bei großen Modellen umfangreiche Vorab- oder Nebenberechnungen notwendig sind. In diesem Fall dürfte die Wahl der Programmiersprache für den Modellgenerator auf kompilierte Sprachen mit performantem Code wie C, C++ oder Fortran fallen, die prozedurale Solverbibliotheken oder Klassenbibliotheken benutzen können.

- Modellvariabilität

  Die Modifikation eines Modells ist schwieriger, wenn dieses durch einen Modellgenerator in einer imperativen Programmiersprache erstellt wird, als wenn dies durch eine deklarative algebraische Modellierungssprache geschieht (vgl. [Fourer83]). Daher wird bei Modellen, die sich häufiger strukturell ändern, die Wahl der Schnittstelle eher auf eine Modellierungssprache als auf Klassen- oder Komponentenbibliotheken fallen.

- Stabilität und Fehlerfreiheit

  Modellgeneratoren, die auf prozeduralen Schnittstellen aufbauen, haben tendenziell eine geringere Stabilität und eine höhere Fehleranfälligkeit als bei Benutzung objektorientierter Schnittstellen. Dies liegt einerseits am höheren Abstraktionsniveau objekt-

orientierter Schnittstellen, andererseits aber auch am Vorhandensein spezieller Mechanismen zur Sicherstellung der Typsicherheit. Noch höher sind Stabilität und Fehlersicherheit bei der Verwendung einer Modellierungssprache, sei es durch programmatische Einbettung des Modells (z.B. Laden eines vorkompilierten Modells, wie bei MOSEL, [Dash06d]) oder durch Einsatz eines integrierten Modellierungssystems.

- Kosten

  Zwar haben in vielen Fällen Modellgenerierungssprachen Vorteile gegenüber Matrixgeneratoren, jedoch ist der Einsatz von Modellgenerierungssprachen, eventuell in Verbindung mit einem integrierten Modellierungssystem, meist mit nicht unbeachtlichen Lizenzkosten verbunden. So kostet beispielsweise eine kommerzielle Lizenz für AMPL derzeit ca. 4.000 US$, und eine AIMMS Lizenz kann mit bis zu 17.500 US$ zu Buche schlagen ([Fourer05]). Die Wahl der geeigneten Optimierungssystemschnittstelle hat also nicht nur eine technische, sondern auch eine kaufmännische Dimension.

Diese Aufzählung von Entscheidungskriterien für die geeignete Form der Einbettung von Optimierungsfunktionalität in ein EUS lässt bereits deutlich werden, dass jede Art der Einbettung spezifische Vor- und Nachteile („Tradeoffs“) hat, die in der grob an [ILOG06b] angelehnten Abbildung 4 dargestellt sind. Betrachtet man als gegensätzliche Ansätze die prozedurale Matrixgenerierung auf der einen und die Modellgenerierung mittels einer algebraischen Modellierungssprache auf der anderen Seite, so sprechen viele Vorteile für den Modellierungssprachenansatz. Dies ist auch die Kernaussage eines Artikels von Robert Fourer ([Fourer83]), der die Verwendung von Modellierungssprachen zur Übersetzung des Modells aus einer algebraischen Notation („Modeler's Form“) in eine algorithmisch verwertbare Form („Algorithm's Form”) propagiert. Trotz des Alters des genannten Beitrags gelten nach wie vor die dort aufgeführten grundsätzlichen Nachteile von Matrixgeneratoren:

- Mangelhafte Verifizierbarkeit der Korrektheit des erzeugten Modells
- Schwierige Modifizierbarkeit des Modells
- Aufwändige Dokumentation von Modell und Modellgenerator
- Abhängigkeit des Modellgenerators von den Datenstrukturen des Solvers
- Komplexität

Fourer führt noch weitere Nachteile an. Diese betreffen jedoch nur spezifische Tools zur Matrixgenerierung, die teilweise heute nicht mehr existieren bzw. veraltet sind (z.B. *MODELER* [Modeler80], *UIMP* [Ellison82]). Als Antwort auf die festgestellte Überlegenheit von Model-

lierungssprachen stellt Fourer ein Grobkonzept für die Modellierungssprache *XML* (nicht zu verwechseln mit der *Extensible Markup Language*) vor. Dieses bereits in [Fourer78] erarbeitete Konzept bildete einen Ausgangspunkt für die spätere Entwicklung der Modellierungssprache *AMPL* (vgl. [Fourer90], [Fourer93], [Fourer03a]).

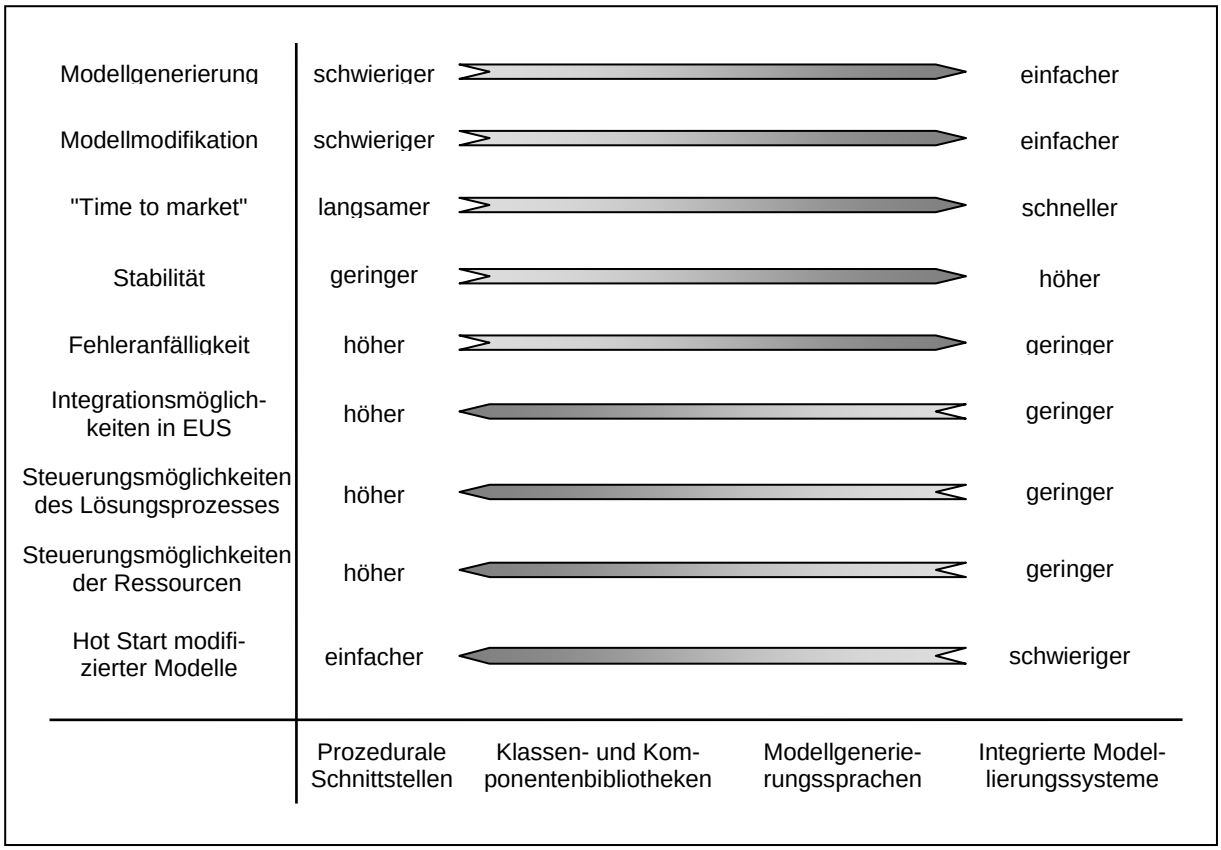

Abbildung 4: Tradeoffs bei Optimierungssystemschnittstellen

Betrachtet man die beiden konträren Möglichkeiten der Modellgenerierung „Prozeduraler Matrixgenerator“ versus „Algebraische Modellierungssprache“, dann stellt sich die Frage nach einer möglichen Synthese beider Ansätze, mit dem Ziel, die jeweils spezifischen Vorteile zu vereinen. Während der Hauptvorteil des Ersteren die nahtlose Einbettung in Standardprogrammiersprachen ist, besteht die Stärke der Modellierungssprachen in der robusten Modellgenerierung mit wenigen, an der algebraischen Notation orientierten Anweisungen. Modellierungssprachen sind jedoch geschlossene deklarative Sprachsysteme, die sich nicht ohne Weiteres in die verbreiteten imperativen Programmiersprachen, die zur Entwicklung eines EUS üblicherweise verwendet werden, integrieren lassen. Dabei scheitert eine Integration der deklarativen (=definitorischen) Modellierungssprachen in imperative (=operationale) Programmiersprachen wie C++, Fortran, VB etc. nicht nur am praktischen Mangel entsprechender Integrationsmechanismen, sondern auch an der konzeptionellen Verschiedenheit der kont-

rären Programmierparadigmen[1]. Zwar bieten mehrere Modellierungssprachen die Möglichkeit der Einbettung der Modelle in operationale Programmiersprachen, doch bleiben die Modelle in ihrer deklarativen Form dabei Fremdkörper: Das Modell wird separat in der jeweiligen Modellierungssprache entwickelt, als Plaintext oder in kompilierter Form (z.B. bei MOSEL) abgespeichert und dann zur Laufzeit des EUS von einer Funktion der Optimierungssystembibliothek eingelesen und verarbeitet. Die Formulierung des LP/-MIP-Modells ist also nicht Bestandteil des Programmcodes, wie es beispielsweise bei Constraint Logic-Modellen in der deklarativen Sprache PROLOG der Fall ist (vgl. [Sterling94], [Fourer02]).

Gleichwohl hat eine gewisse Annäherung der deklarativen an die prozedurale Sphäre stattgefunden, indem viele moderne Modellierungssprachen (z.B. AMPL, OPL, MOSEL s. Kapitel 3.4) durch diverse prozedurale Elemente zur Datenmanipulation und Lösungsprozesssteuerung angereichert wurden, so dass sie nicht mehr gänzlich, sondern nur noch von ihrem Grundcharakter her deklarativ sind.

Ein anderer Ansatz zur Überbrückung der Lücke zwischen prozeduralen Matrixgeneratoren und Modellierungssprachen ist der Einsatz objektorientierter Standardtechnologien in Form von Klassen- und Komponentenbibliotheken. Das höhere Abstraktionsniveau objektorientierter Programmierung in Verbindung mit syntaktischen Features (z.B. Operatorüberladung) erlaubt innerhalb einer imperativen, objektorientierten Standardprogrammiersprache die Annäherung an die deklarative Formulierung einer Modellierungssprache.

---

[1] Zu definitorischen und operationalen Paradigmen vgl. [Ambler92].
[Schichl04] und [Hürlimann98] nennen drei Sprachparadigmen, die üblicherweise unterschieden werden: Imperativ (C, Fortran etc.), Funktional (LISP) und Logic Programming (Prolog, Eclipse).
[Hürlimann98] skizziert die Vorteilhaftigkeit einer Multiparadigmen-Modellierungssprache, trennt aber auch dort bewusst die deklarativen von den operationalen Teilen.

Solversystem	Bibliotheken					Anbindung an Modellierungs-sprachen	Integrierte Modellierungs-umgebung
	Prozedural	Komponenten		Klassen			
		COM	.NET	C++	Java		
COIN (CLP etc.)	Ja	Nein	Nein	Ja	Nein	Ja	Nein
CPLEX	Ja	Nein	Ja	Ja	Ja	Ja	Ja
C-WHIZ/MIPIII	Ja	Nein	Nein	Nein	Nein	Ja	unbek.
FortMP	Ja	Nein	Nein	Nein	Nein	Ja	Ja
GAUSS	Ja	Nein	Nein	Nein	Nein	Ja	Ja
LINDO	Ja	Nein	Nein	Nein	Nein	Ja	Ja
MINOS	Ja	Nein	Nein	Nein	Nein	Ja	Nein
MOPS	Ja	Nein*	Nein*	Nein*	Nein	Nein*	Nein*
MOSEK	Ja	Nein	Ja	Ja	Ja	Ja	Nein
SoPlex	Nein	Nein	Nein	Ja	Nein	Ja	Nein
XPRESS	Ja	Nein	Ja	Ja	Ja	Ja	Ja

* In Konzeption, teilweise bereits entwickelt

Tabelle 1: Schnittstellen von Optimierungssystemen

Wie Tabelle 1 darstellt, existieren für andere Optimierer bereits einige derartige Schnittstellenentwürfe, meist in Form von Klassenbibliotheken für C++ oder Java. Komponentenbasierte Schnittstellenkonzepte sind hingegen relativ selten: Es gibt einige wenige Implementationen für das .NET Framework und fast keine auf COM basierenden Ansätze, wobei eine Ausnahme die *MPL Library* ([Maximal06]) bildet. Wegen dieser Lücke und der großen Bedeutung von Komponentenarchitekturen für die moderne Softwareentwicklung (vgl. z.B. [Maurer00]), liegt der Implementationsschwerpunkt dieser Arbeit auf komponentenbasierten Ansätzen zur Modellgenerierung.

Eine inhaltlich ähnliche Darstellung der Tradeoffs bei Benutzung unterschiedlicher Modellierungsschnittstellen findet sich bei [Koch04, S. 4] und ist in Abbildung 5 wiedergegeben. Dort wird der als „Effort“ bezeichnete Modellierungs- und Programmieraufwand dem erzielbaren „Effect“, d.h. der Flexibilität und Performance der Modellierung gegenübergestellt. Die mit geringstem Aufwand höchsten Effekte erreicht man mit als „Workbench“ bezeichneten Programmen, wie z.B. MATHEMATICA, mit denen man kleinere Modelle sehr schnell erstellen kann, allerdings ist deren Flexibilität im Vergleich zu einer Modellierungssprache begrenzt. Ein als „Do-it-yourself“ bezeichneter, direkt auf den Solverschnittstellen selbst erstellter Modellgenerator ist zwar äußerst effektiv, verlangt aber auch den größten Entwicklungsaufwand. Ein Mittelweg kann ein „Framework“ sein, das bereits diverse Algorithmen zur Modellierungsunterstützung mitbringt und dem Benutzer so erlaubt, sich auf die problemspezifischen Fragestellungen zu konzentrieren.

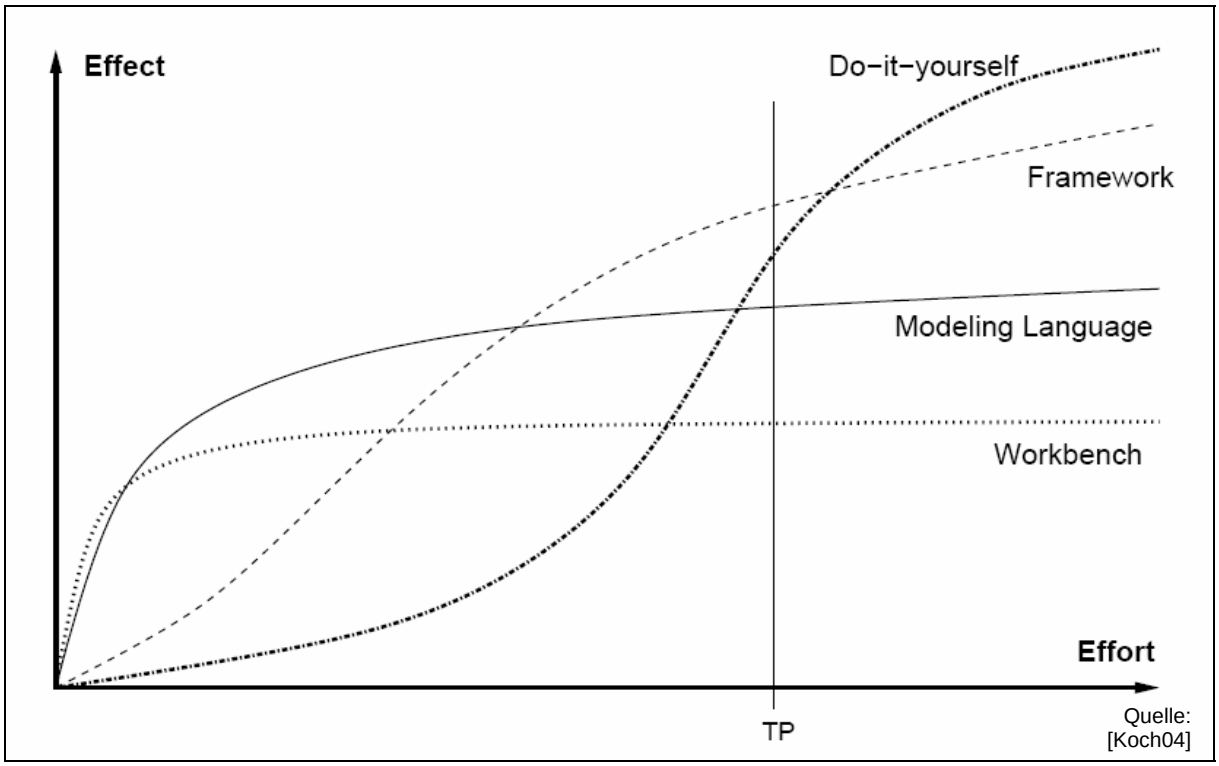

Abbildung 5: „Effort vs. Effect" bei Optimierungssystemschnittstellen

2.3 Ziele der Arbeit

Ausgehend von der zuvor beschriebenen Kernproblematik ergeben sich somit für die Arbeit folgende Hauptziele:

- Systematische Darstellung von vorhandenen Schnittstellenansätzen und Modellierungssprachen für LP-/MIP-Optimierungssysteme
- Konzeption einer Middleware mit einem kohärenten Schnittstellensystem zur Einbettung von Optimierungssoftware in Anwendungssysteme und deren teilweise Implementierung in den Bereichen:
    - Entwicklung einer COM-basierten Optimierungsobjektbibliothek für Rapid Application Development
    - Anbindung an verschiedene Modellierungssprachen (AMPL, MathProg, Lindo, MPL)
    - Entwicklung eines integrierten Modellierungssystems mit grafischer Benutzeroberfläche zur Benutzung dieser Sprachen und dessen Validierung am Beispiel des Optimierungssystems MOPS

Das erstgenannte Ziel, die systematische und ausführliche Darstellung von Schnittstellen zu Optimierungssystemen, ergibt sich aus der Tatsache, dass diesem Thema in der Literatur bisher kaum Bedeutung zugemessen wurde. So beschäftigt sich die OR-Literatur fast ausschließlich mit mathematischen und algorithmischen Problemen oder mit bestimmten Anwendungsfällen, fast nie jedoch mit den in der Praxis bedeutsamen Fragen der Modellgenerierung und der Einbettung von Optimierungssoftware in EUS. Dies zeigen einige Beispiele aus den wenigen Quellen, die sich überhaupt mit der Frage von Schnittstellen befassen: Beispielsweise gibt [Fourer05] in seiner Optimierungsproduktübersicht lediglich die Schnittstellentypen an, über die die dort gelisteten Optimierungssystem verfügen. In den Beiträgen der von [Voß02] herausgegebenen Aufsatzsammlung wird zwar auf die Schnittstellen der dort vorgestellten Optimierungsklassenbibliotheken eingegangen, jedoch fehlt zum einen eine systematisierende und vergleichende Darstellung, und zum anderen handelt es sich bei den vorgestellten Bibliotheken mehrheitlich um Softwarepakete für Constraint Programming, nicht-lineare Optimierung und für andere Problemklassen außerhalb des hier abgesteckten Bereichs der LP- und MIP-Modelle. Ähnliches gilt für die in [Fink02b] gegebene Übersicht. Für nichtkommerzielle MIP-Solver nimmt [Linderoth05] eine grobe Interfaceklassifizierung vor und beschreibt die Schnittstellen auch, stellt aber keinerlei Vergleiche oder Systematisierungen an.

Das zweite Hauptziel, die Entwicklung eines systematisch zusammenhängenden Schnittstellensystems, ergibt sich aus der Tatsache, dass es bisher für Optimierungssysteme, insbesondere für solche mit akademischem oder semi-kommerziellem Hintergrund, kaum derartige Konzepte gibt. Lediglich das kommerzielle System von ILOG ([ILOG06c]), sowie mit einigem Abstand XPRESS-MP (vgl. [Guéret02], [Dash06a], [Dash06b]) bieten ein komplexeres, zusammenhängendes Schnittstellensystem. Im nicht-kommerziellen Bereich kann im Wesentlichen nur die Initiative *COIN* ([COIN06]) genannt werden, die unter anderem versucht, mit dem *Open Solver Interface* (vgl. [LHeimer03], [OSI06]) und der Open-Source Modellierungssprache *FLOPC++* ([Hultberg06]) ein universelles Schnittstellensystem für mathematische Optimierungssoftware zu schaffen. Hierbei liegt der Fokus jedoch auf der Programmiersprache C++ und - anders als in der vorliegenden Arbeit - wird komponentenbasierten Ansätzen kaum Aufmerksamkeit geschenkt. Der hier exemplarisch für die Implementationen herangezogene Solver MOPS besitzt nur eine prozedurale Schnittstelle (DLL/LIB) mit einem recht übersichtlichen Funktionsumfang sowie eine Kommandozeilenschnittstelle in Form eines Executables. MOPS eignet sich daher auch aus praktischer Sicht gut als Exempel, da für dieses System noch einiger Entwicklungsbedarf im Interfacebereich besteht.

Diese Arbeit fokussiert die durchgeführte Implementierung der komponentenbasierten Optimierungssystemschnittstellen auf das *Component Object Model* (COM). Für diese sehr weit verbreitete und im Windows-Bereich immer noch dominante Komponententechnologie existieren kaum Optimierungsbibliotheken (siehe Tabelle 1), so dass sich hier aus konzeptioneller und praxisbezogener Sicht noch eine Lücke auftut, die durch die hier geleistete Entwicklungsarbeit geschlossen werden soll.

Die Entwicklung der Modellierungsoberfläche *MOPS Studio* motiviert sich aus dem praktischen Mangel an offenen und frei verfügbaren, modernen grafischen Modellierungsoberflächen, die in der Lehre einsetzbar sind. Zwar gibt es mit Produkten wie *OPL Studio* ([Heisig99], [ILOG06d]) oder *XPRESS-IVE* ([Dash06c]) bereits leistungsfähige, kommerzielle Oberflächen, doch sind diese häufig an einen bestimmten kommerziellen Optimierer gekoppelt und/oder nicht kostenfrei einsetzbar.

2.4 Abgrenzung

Operations Research wird fast immer nach der unterschiedlichen mathematischen Struktur der Modelle in die folgenden Teilgebiete aufgeteilt ([Dürr02, S. 20]): Lineare Optimierung, nichtlineare Optimierung, Netzplanprobleme, ganzzahlige und kombinatorische Optimierung, dynamische Optimierung und Simulationsverfahren. Davon sollen hier nur lineare und ge-

mischt-ganzzahlige Modelle betrachtet werden. Zwar unterstützen viele der im Rahmen dieser Arbeit erwähnten Optimierungssysteme auch nichtlineare Modelle oder es ergeben sich Berührungspunkte zum Constraint Programming (z.B. bei der *ILOG Concert Library* und *ILOG Solver*, s. Abschnitt 3.2.1.2), jedoch bilden LP/MIP-Modelle eine der wichtigsten Modellklassen des Operations Research und überdies ist MOPS, als Basis aller durchgeführten Implementationen, ein reiner LP/MIP-Optimierer. Daher werden, insbesondere bei der Beschreibung des State-of-the-art in Kapitel 3, Optimierungsfunktionalitäten jenseits von LP/MIP-Modellen nicht berücksichtigt.

Was die Softwareseite anbelangt, so beschränkt sich diese Arbeit ausschließlich auf Technologien für Personal Computer unter Windows. Wurden in führen Zeiten Optimierungsmodelle noch hauptsächlich auf Großrechnern gelöst, so haben sich seit dem Aufkommen der PCs deren Rechenleistungen so erheblich gesteigert, dass heute der größte Teil der Optimierungsaufgaben von PCs erledigt werden kann (vgl. z.B. [AtaSav05]). Suhl ([SuhlL06, S. 79]) nennt z.B. für MOPS Rechenzeitverkürzungen im Zeitraum 1991 bis 2004 von 612,4 Sekunden auf 0,6 Sekunden, wofür teils algorithmische Verbesserungen, teils Leistungssteigerungen der Hardware verantwortlich sind. Obwohl MOPS prinzipiell auch in Versionen für UNIX oder OS/390 und andere verfügbar ist, sind diese Implementation doch zu Gunsten der PC-Version immer mehr in den Hintergrund getreten. Aus diesem Grund soll sich hier auf Personal Computer beschränkt werden. Auf Personal Computern ist Windows mit über 90% Marktanteil ([Gonsalves03]) trotz wachsender Popularität von Linux noch immer beherrschend, so dass eine Fokussierung auf dieses Betriebssystem und seine Schnittstellenkonzepte und Frameworks naheliegt.

In Konkurrenz zu den hier behandelten Komponentenarchitekturen COM/DCOM und .NET steht vor allem CORBA (*Common Object Request Broker Architecture*, vgl. [Sayegh97]). CORBA ist zwar plattformunabhängig, und es existieren auch Implementationen für Windows, jedoch sind seine Domänen vor allem Java und der UNIX-Bereich, weshalb CORBA hier zwar kurz dargestellt wird, aber im Schnittstellenkonzept zunächst ausgeklammert bleibt.

Ebenfalls ausgeklammert bleiben soll der SAP-Bereich. Zwar integriert beispielsweise das SAP-Modul *Advanced Planner and Optimizer* (*SAP APO*) als externe Optimierungsengines die Produkte *ILOG CPLEX* für das *Supply Network Planning* (*SNP*) und *ILOG Solver* für *Production Planning and Detailed Scheduling* (*PP/DS*) ([SAPInfo02]), jedoch sind diese Integrationsfragen eher SAP-spezifisch und nicht genereller Natur.

Eine letzte Abgrenzung gilt der Untersuchungsperspektive auf die Optimierungsengines: Abgesehen von ihrer Schnittstellenschicht werden Solver und ihre Algorithmen hier als „Black

Box“ betrachtet. Es wird also nicht untersucht, ob und wie bestimmte Optimierungsalgorithmen implementiert sind, sondern nur, wie sich ein Solver in sein Umsystem integriert.

2.5 Angewendete Forschungsmethoden

In dieser Arbeit wird von mehreren Forschungsmethoden Gebrauch gemacht:
Die primär verwendete Methode ist die Konstruktion von Prototypen. Dabei soll diese in der Wirtschaftsinformatik dominante Forschungsmethode (vgl. [König96] und [Braun04, S. 7]) auch unter dem Licht einiger Ideen des Ansatzes "Forschung durch Entwicklung" ([SzyperskiN79]) gesehen werden.
Der Ansatz „Forschung durch Entwicklung“ geht davon aus, dass in Praxis und Lehre ein „Bedarf an bewährten und relevanten substantiven und operativen konzeptionellen Aussagen […] zur Gestaltung rechnergestützter Informationssysteme“ besteht ([SzyperskiN79, S. 253]). Dabei betreffen „substantive Aussagen“ der angewandten Wissenschaften Objekte, etwa Rechner, während „operative Aussagen“ Aktionen betreffen, etwa den Prozess der Flugeinsatzplanung (zu „substantiv“ vs. „operativ“ vgl. [Bunge66]). Weitere Merkmale des Ansatzes „Forschung durch Entwicklung“ sind zum einen eine enge Zusammenarbeit des Forschenden mit Praktikern, zum andern aber auch die Grundhaltung, dass konzeptionelle Aussagen nicht beim Systementwurf enden, denn:

> *„Die logisch einwandfreie Darstellung eines Begründung- oder Funktionszusammenhangs genügt […] zu einer Bewährung deswegen nicht, weil damit über das tatsächliche Funktionieren eben noch nichts Zureichendes gesagt ist. Oder anders ausgedrückt: Die Aussage über das erwartete Funktionieren genügt nicht zum Erweis des tatsächlichen Funktionierens. Hier liegt der Grund, warum sich jede Erfindung oder Neukonstruktion durch das Funktionieren eines Prototyps bewähren muss.“ ([Tuchel68, S. 584], zitiert in [SzyperskiN79, S. 254-255])*

Dabei kommt die Vorstellung zum Ausdruck, „dass nicht nur bei empirischen oder deduktiven Forschungsvorhaben, sondern gerade auch in praktischen Einzelfällen (Erfahrungen, Entdeckungen) neues Wissen gefunden wird“ ([SzyperskiN79, S. 258]). Ein solcher praktischer Einzelfall ist das hier entwickelte Schnittstellensystem, und die daraus gewonnenen Erfahrungen werden im Rahmen dieser Arbeit wiedergegeben.
Das Konzept „Forschung durch Entwicklung“ beinhaltet zwar auch viele sozio-organisatorische Aspekte, die hier jedoch zu Gunsten einer technischen, konstruktionsorientierten Vorgehensweise in Form von Prototypenentwicklung ausgeklammert werden. Neben der Prototypenentwicklung müssen zur wissenschaftlichen Durchdringung allerdings auch Rahmenkonzepte und theoretische Hintergründe betrachtet werden.

Im Sinne einer mehrere Forschungsmethoden verwendenden Herangehensweise (*Multimethod Approach*, vgl. [Mingers01], [Esteves04]) werden neben der Prototypenentwicklung als Methoden noch Systemvergleich und Fallstudienanalyse einbezogen. Der Systemvergleich dient der systematisierenden Beschreibung vorhandener Optimierungssystemschnittstellen und hat Ähnlichkeit mit der in vielen Arbeiten initial vorgenommenen Literaturrecherche, nur dass hier die relevanten Optimierungssysteme direkt (primär) untersucht und beschrieben werden (Kapitel 3). Die Fallstudie zum „Projekt EPOS“ (Unterkapitel 5.1) dient der Illustration der Schnittstellenproblematik von Optimierungssystemen anhand eines konkreten Anwendungsfalls.

Alle drei verwendeten Methoden Prototypenkonstruktion, Systembeschreibung und Fallstudie bieten somit unterschiedliche Perspektiven auf den Untersuchungsgegenstand dieser Arbeit.

2.6 Aufbau der Arbeit

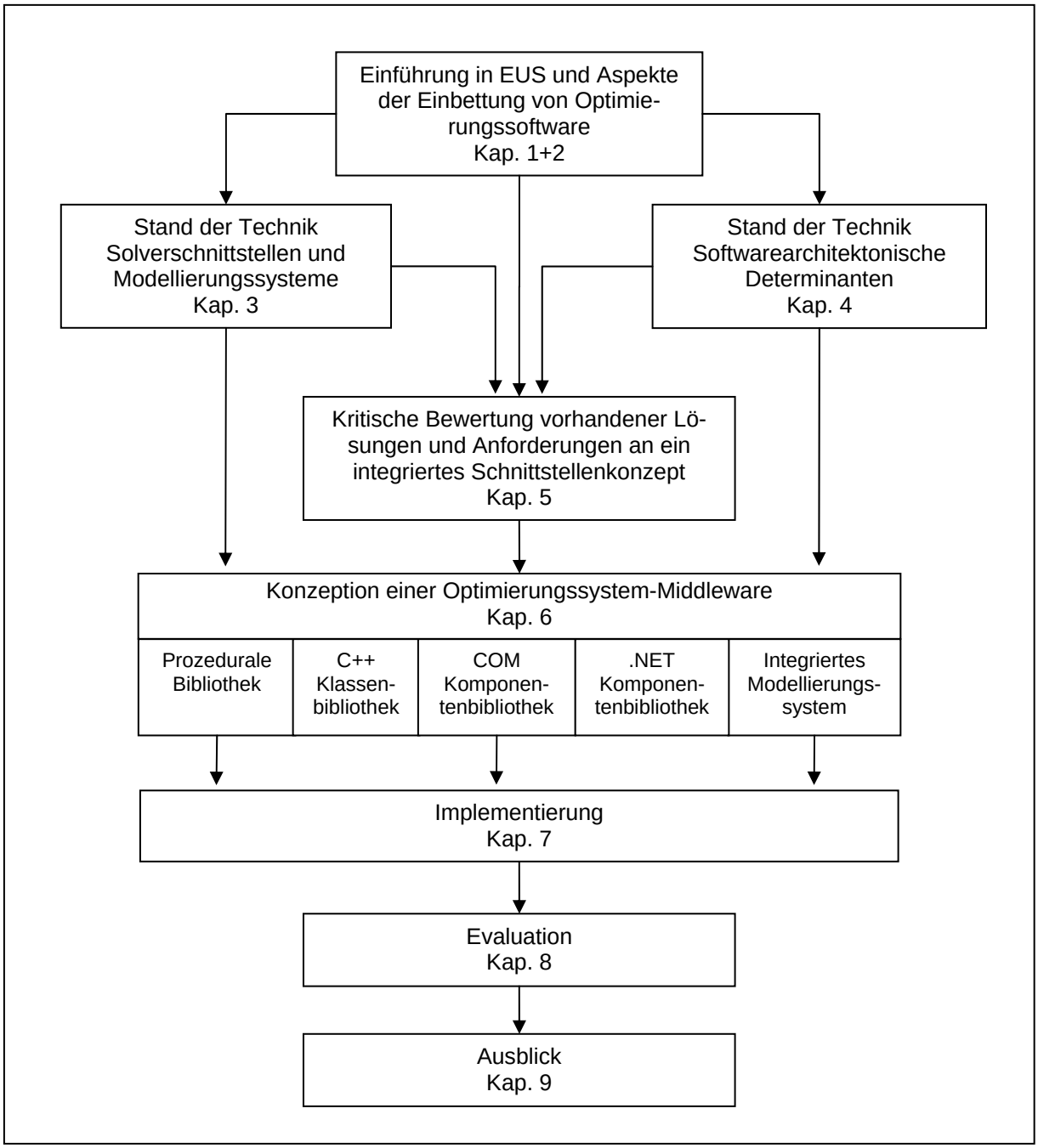

Abbildung 6: Aufbau der Arbeit

Der Aufbau der vorliegenden Arbeit ist in Abbildung 6 skizziert: Nach einer Einleitung wird in den Kapiteln 3 und 4 der Stand der Technik in Bezug auf Schnittstellen von Optimierungssystemen und Modellgenerierungssprachen sowie bezüglich softwarearchitektonischer Fragen beschrieben. Kapitel 5 unterwirft die bestehenden Konzepte einer kritischen Betrachtung und leitet zur Darstellung des eigenen Konzepts für ein Optimierungsmiddlewaresystem in Kapitel 6 über. Die Implementation der verwirklichten Teile des Konzepts (Prozedurale Bibliothek,

COM-Bibliotheken, MOPS Studio) ist in Kapitel 7 beschrieben, und in Kapitel 8 erfolgt eine kurze Evaluation dieser Bestandteile. Mit einem Ausblick auf die noch offenen Fragen und die noch ausstehenden Arbeiten wird in Kapitel 9 abgeschlossen.