Institut für Informatik
der Technischen Universität München

Softwarekartographie: Modelle und Methoden zur systematischen Visualisierung von Anwendungslandschaften

André Wittenburg

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Martin Bichler
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. Florian Matthes
	2. Univ.-Prof. Dr. Helmut Krcmar

Die Dissertation wurde am 24.07.2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 19.11.2007 angenommen.

Zusammenfassung

Die Langlebigkeit und die steigende Anzahl von Anwendungssystemen in Unternehmen führen zu immer komplexeren Anwendungslandschaften, die von den Unternehmen zunehmend als vernetztes System betrachtet werden, um Strategien zu verwirklichen und Projekte zu planen. In dieser Arbeit wird das Konzept der Softwarekartographie als Hilfsmittel zur Beschreibung, Bewertung und Gestaltung dieser Anwendungslandschaften vorgestellt. Es werden intuitive und verständliche Sichten auf relevante Merkmale für Anwendungslandschaften eingeführt, welche die Dokumentation, Planung und Weiterentwicklung von Anwendungslandschaften unterstützen.

Auf Basis einer explorativen Analyse von in der Praxis verbreiteten Konzepten zur Beschreibung und zur Gestaltung von Anwendungslandschaften (u. a. bei BMW Group, Deutsche Post, HVB, Münchener Rück, Siemens, T-Com), wird in dieser Arbeit eine graphische Modellierungssprache und eine Methode zu ihrer Anwendung vorgestellt. Die Managementprozesse, um eine langfristige, zielgerichtete Planung und Steuerung der Anwendungslandschaft zu ermöglichen, werden beschrieben sowie die Verwendung der Modellierungssprache und der Methode in diesen Prozessen erläutert.

Ausgehend von den Modellen und der Methode werden die grundlegenden Anforderungen an eine geeignete Werkzeugunterstützung zur Softwarekartographie beschrieben und Defizite existierender kommerzieller Werkzeuge aufgezeigt. Insbesondere der Zusammenhang von semantischen und symbolischen Modellen für Anwendungslandschaften wird in diesem Kontext verdeutlicht, so dass eine enge Kopplung zwischen den semantischen Informationen in einem Repository und den Symbolen einer Softwarekarte entsteht.

Abstract

The long life-time and the increasing number of application systems used by enterprises lead to complex application landscapes. Enterprises start to consider these application landscapes as highly connected systems used to implement strategies and to plan projects. This work introduces Software Cartography, which provides support for documenting, evaluating, and planning application landscapes. Intuitive graphical views visualizing relevant attributes of application landscapes are introduced to support the management of application landscapes.

Based on an explorative analysis of concepts for documenting and planning application landscapes (with BMW Group, Deutsche Post, HVB, Munich Re, Siemens, T-Com et al.), a modeling language is developed and an associated methodology for the use of software maps is presented. The management processes for a long-term and goal-oriented planning and controlling of the application landscape are defined and the usage of the developed modeling language in these processes is presented.

Starting with a modeling language including its models and method, requirements for tool support are identified and shortcomings of commercial tools are described. The relationship between semantic and symbolic models of application landscapes is explained to clarify the need for a tight coupling between the semantic information in a repository and the symbols in a software map visualizing an application landscape.

Danksagung

Diese Arbeit ist während meiner Anstellung am Lehrstuhl für Informatik 19 (sebis) der Technischen Universität München entstanden. An dieser Stelle möchte ich all denen danken, die mich während meiner Promotion unterstützt haben.

Herrn Prof. Dr. Florian Matthes danke ich für die Möglichkeit an seinem Lehrstuhl promovieren zu dürfen und dafür, dass er es mir stets ermöglicht hat, das Forschungsprojekt *Softwarekartographie* mitzugestalten. Herrn Prof. Dr. Helmut Krcmar danke ich für die zahlreichen Gespräche, die wir geführt haben, und für die Übernahme des Zweitgutachtens dieser Arbeit.

Ein besonderer Dank gilt Herrn Prof. Dr. Ernst Denert, der den Projektantrag für das Forschungsprojekt *Softwarekartographie* befürwortet und dessen Stiftung diese Arbeit gefördert hat. Er hat mit seinem persönlichen Engagement und durch die Vermittlung wertvoller Industriekontakte zum Erfolg des Projektes beigetragen.

Dem ganzen Softwarekartographieteam, welchem ich angehört habe, gilt mein besonderer Dank für unzählige Stunden, die wir mit Diskussionen, Artikelschreiben, Studien und Reviews verbracht haben. Dieser Dank geht an Sabine Buckl, Alexander M. Ernst, Josef Lankes und Christian M. Schweda.

„Meinen“ Studenten, deren Arbeiten ich betreuen durfte, Nico Beyer, Katharina Brendebach, Sabine Buckl, Christian M. Schweda, Peggy Sekatzek, Ronny Spiegel und Astrid Stangler danke ich für ihre Beiträge zu dieser Arbeit und auch für die Geduld, die sie bei meinen Korrekturen aufgebracht haben.

Dr. Gerald Schröder und Dr. Ulrike Steffens danke ich für die Durchsicht dieser Arbeit und die zahlreichen Hinweise, die sie mir gegeben haben.

Die Ideen zu dieser Arbeit sind über die Jahre in vielen Workshops und Gesprächen mit Menschen entstanden, die sich für das Architekturmanagement begeistern können. Sie haben mir einen Einblick in ihre Arbeitsweisen gewährt und ich durfte meine Ideen an ihren spiegeln. Dieser Dank geht an alle Projektpartner und Interessierte der Softwarekartographie.

Meinen herzlichster Dank gilt meinen Eltern für ihre vorbehaltlose Unterstützung und den Rückhalt, den sie mir stets gegeben haben. Vielen Dank!

Garching b. München, Juli 2007

André Wittenburg

Förderung durch die Ernst Denert-Stiftung

Diese Arbeit wurde gefördert durch die Ernst Denert-Stiftung für Software-Engineering, Mitglied im Stifterverband für die Deutsche Wissenschaft.

Inhaltsverzeichnis

Abbildungsverzeichnis

KAPITEL 1

Einführung und Überblick

Inhaltsverzeichnis

Die Entwicklung der Informationstechnologie hat seit ihrem Bestehen dazu beigetragen, dass in heutigen Unternehmen zahlreiche betriebliche Anwendungssysteme im Einsatz sind, die mittels verschiedenster Technologien entwickelt wurden, über verschiedene Schnittstellen miteinander verbunden sind und unterschiedliche Geschäftsprozesse unterstützen. Dabei tragen die Anwendungssysteme direkt oder indirekt zum Geschäftserfolg bei und werden entlang der Wertschöpfungskette nicht nur für Unterstützungsfunktionen (*Support Activities*) [Po85], wie Personalwesen oder Rechnungswesen, eingesetzt, sondern ebenso zur Unterstützung der Primärfunktionen (*Pimary Activities*), wie Einkauf, Produktion oder Vertrieb.

Nicht nur die Zahl der Anwendungssysteme hat dazu geführt, dass Unternehmen ihre Anwendungslandschaft zunehmend als Ganzes betrachten, sondern auch das gestiegene Interesse an einer verbesserten Ausrichtung der Informationstechnologie (IT) an dem Geschäft des Unternehmens [Ro05b, Lu05, LM04]. Eine Methode inklusive geeigneter Modelle, um Anwendungslandschaften zu beschreiben und zu gestalten, die nicht nur von einzelnen Unternehmen genutzt wird, ist jedoch bisher nicht prominent hervorgetreten.

Die vorliegende Arbeit entwickelt eine Modellierungssprache für Anwendungslandschaften und ordnet diese in die Managementprozesse für Anwendungslandschaften ein. Eine Bestandsaufnahme des Status quo zur Beschreibung und zur Gestaltung von Anwendungslandschaften zeigt die Anforderungen an die Modellierungssprache, welche die verschiedenen Ansätze unter Berücksichtigung existierender Methoden der Informatik, insbesondere der Architekturdokumentation vereinigt.

1.1. Motivation

Im Software-Engineering wurden zahlreiche Modellierungssprachen für einzelne Anwendungssysteme entwickelt, jedoch fehlt bisher eine Modellierungssprache, welche sich der Herausforderung zur Dokumentation, Planung und Gestaltung der gesamten Anwendungslandschaft mit geeigneten Methoden und Modellen annimmt.

Diese Arbeit fokussiert somit nicht auf einzelne Anwendungssysteme, wie es beispielsweise die *Unified Modeling Language* (UML) [OM05d] tut, sondern auf die Anwendungslandschaft bestehend aus hunderten oder tausenden Anwendungssystemen.

Des Weiteren werden im Gegensatz zu anderen Modellierungssprachen die Anwendungslandschaft und die Anwendungssysteme, aus denen sich die Anwendungslandschaft zusammensetzt, nicht als starr betrachtet, sondern es werden ebenso die Veränderungsprozesse in Unternehmen, die in Projekten eine Veränderung der Anwendungslandschaft hervorrufen, berücksichtigt. Dies stellt einen Unterschied zu anderen Modellierungssprachen wie beispielsweise ArchiMate [La04] dar, welche zwar die Modellierung der Unternehmensarchitektur ermöglicht und u. a. Organisationseinheiten, Geschäftsprozesse und Infrastrukturelemente in Betracht zieht, jedoch Elemente, welche die Unternehmensarchitektur verändern, nicht berücksichtigt.

Eine Modellierungssprache für Anwendungslandschaften muss zahlreiche Interessen an dem zu betrachtenden System adressieren und intuitive graphische Sichten für diese Interessen zur Verfügung stellen. Bisher vorliegenden Modellierungssprachen fehlt es an diesen Sichten auf die Anwendungslandschaft, um zahlreiche Stakeholder zu unterstützen und sowohl eine kurz- und mittelfristige Planung der Anwendungslandschaft zu ermöglichen, als auch langfristige Planungen, die entlang von Strategien und Zielen entwickelt werden.

1.2. Beiträge

Die vorliegende Arbeit entwickelt eine Modellierungssprache, um komplexe Anwendungslandschaften systematisch zu beschreiben und zu gestalten. Als Beschreibungsmittel werden so genannte Softwarekarten eingeführt, die bestimmte Interessen an dem betrachteten System – der Anwendungslandschaft – adressieren. Diese Softwarekarten sind graphische Repräsentationen der Anwendungslandschaft oder Teile dieser und besitzen einen Aufbau analog zu Karten in der thematischen Kartographie [HGM02, Sl05].

Die verschiedenen Softwarekarten werden hinsichtlich der verwendeten Gestaltungsprinzipien analysiert und mittels Typen kategorisiert, so dass die Konstruktionsprinzipien jedes einzelnen Typs erkennbar werden. Ergänzend werden existierende Modellierungssprachen, wie beispielsweise die *Unified Modeling Language* (UML) [OM05d], den Anforderungen gegenübergestellt sowie die Defizite dieser Sprachen zur Modellierung von Anwendungslandschaften aufgezeigt.

Aufbauend auf den Konstruktionsprinzipien werden objektorientierte Modelle vorgestellt, um zum einen die dargestellte Information (semantisches Modell) und zum anderen die graphischen Visualisierungselemente (symbolisches Modell) zu beschreiben. Eine enge Kopplung zwischen den Modellen, die analog zu einer Modelltransformation zu verstehen ist, ermöglicht die korrekte Visualisierung von Informationen. Dadurch werden auf der einen Seite Defizite von *starren* Modellierungssprachen, die sich nicht an die sich verändernden Interessen der Stakeholder eines Systems anpassen können, vermieden und auf der anderen Seite wird *gemalten* Architekturdokumentationen entgegengewirkt, die keine definierte Syntax und Semantik besitzen.

Neben der Entwicklung der Modellierungssprache mit dem Konzept der Softwarekarte wird aufgezeigt, wie sich das Management der Anwendungslandschaft existierender IT-Managementprozesse im Unternehmen bedient und das Management der Unternehmensarchitektur hierbei als *Klebstoff* fungiert. Die Anwendung von Softwarekarten beim Management von Anwendungslandschaften zeigt, wie Softwarekarten die Gestaltung von Anwendungslandschaften beeinflussen und sowohl im operativen Betrieb als auch in planerischen und strategischen Prozessen eingesetzt werden. Die Softwarekartographie stellt somit Methoden und Modelle zum Management der Anwendungslandschaft bereit.

1.3. Initiale Begriffsdefinitionen

Die zentralen Begriffe dieser Arbeit *Anwendungslandschaft*, *Unternehmensarchitektur*, *Architektur* und *Softwarekarte* sind im allgemeinen Begriffsapparat der Informatik und der Wirtschaftsinformatik bisher nicht durchgehend vorhanden bzw. werden unterschiedlich benutzt. Um für die Arbeit eine erste Grundlage für den verwendeten Begriffsapparat zu legen, werden die Begriffe an dieser Stelle bereits eingeführt.

Der Begriff *Anwendungslandschaft* wird für diese Arbeit wie folgt definiert:

> **Anwendungslandschaft :** Die Anwendungslandschaft ist die Gesamtheit der betrieblichen Anwendungssysteme inklusive der Kommunikationsbeziehungen zwischen den Anwendungssystemen in einem Unternehmen.

Unter einem *Anwendungssystem* wird ein System, das in Software implementiert ist, verstanden und Teil eines Informationssystems [WK94] ist. Analog zu Krcmar [Kr05, S. 25] wird der Mensch als Teil des Informationssystems, einem soziotechnischen System, nicht zum Anwendungssystem hinzugezählt und die Infrastruktur – bei Krcmar die Hardware –

ist ebenso nicht Teil des Anwendungssystems. Die Infrastruktur wird vom Anwendungssystem genutzt und ist somit ein Serviceerbringer aus Sicht des Anwendungssystems, aber nicht ein Teil dessen.

Mit dem Zusatz „[...] inklusive der Kommunikationsbeziehungen“ wird in der Definition des Begriffs *Anwendungslandschaft* ergänzt, dass auch die Verbindungen zwischen Anwendungssystemen zur Anwendungslandschaft gehören. Das Wort *betrieblich* schränkt zusätzlich die Anwendungssysteme dahingehend ein, dass sie einen betrieblichen Charakter besitzen müssen, der im Folgenden so zu verstehen ist, dass ein *betriebliches Anwendungssystem* mindestens einen Geschäftsprozess des Unternehmens unterstützen muss[1].

Ein weiterer im Kontext der Anwendungslandschaft häufig verwendeter Begriff ist die *IT-Landschaft*, welcher für Anwendungslandschaft und Infrastruktur zusammen verwendet wird und somit die Hardware und weitere Systeme, wie beispielsweise Datenbankmanagementsysteme, Transaktionsmonitore oder Applikationsserver, mit einschließt.

Wird der Fokus der Betrachtung ausgehend von der Anwendungslandschaft erweitert und werden nicht nur die Anwendungssysteme betrachtet, sondern alle Elemente eines Unternehmens, die einen Einfluss auf die Anwendungslandschaft haben, so gelangt man zu einer Architektursicht, die mit Unternehmensarchitektur[2] bezeichnet wird.

> **Unternehmensarchitektur :** Die Unternehmensarchitektur (engl. *Enterprise Architecture*) ist die kohärente und ganzheitliche Architektur eines Unternehmens, die nicht nur die Informationstechnologie sondern ebenso betriebswirtschaftliche Elemente umfasst. Dabei umfasst die Architektur nicht nur die einzelnen Elemente des Unternehmens selbst, wie beispielsweise die Organisationsstruktur, die Geschäftsprozesse, die Anwendungssysteme und die Infrastrukturelemente, sondern auch ihre Verbindungen und Querschnittselemente, wie Strategien & Ziele, Anforderungen & Projekte, Richtlinien & Muster sowie Kennzahlen & Metriken.

Das steigende Interesse an dem Thema Unternehmensarchitektur und ihres Managements wurde von Langenberg und Wegmann [LW04] durch die steigende Anzahl von Publikationen in diesem Bereich belegt.

Der Begriff *Architektur*, der in der Definition für *Unternehmensarchitektur* steckt, ist ein weiterer zentraler Terminus in dieser Arbeit. Für diesen Begriff wird auf eine existierende Definition aus dem IEEE Std 1471-2000 [IE00] zurückgegriffen. *Architektur* wird in dieser Arbeit wie folgt verstanden.

[1] Im folgenden sind die Begriffe *betriebliches Anwendungssystem* und *Anwendungssystem* als synonym zu verstehen. Die Arbeit betrachtet ausschließlich *betriebliche Anwendungssysteme*.

[2] Im deutschen Sprachgebrauch wird ebenso der Begriff *Enterprise-Architektur* als Synonym für *Unternehmensarchitektur* verwendet. Für eine Herleitung des Begriffs *Unternehmensarchitektur* wird auf Abschnitt 2.2 verwiesen.

> **Architektur :** The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. [IE00]

Architektur ist somit der fundamentale Aufbau eines Systems bestehend aus seinen Komponenten, ihren Beziehungen zueinander, der Umgebung und den Prinzipien und Richtlinien, die das Design und die Entwicklung beeinflusst haben. Eine detaillierte Diskussion des Begriffs folgt in Abschnitt 3.1.

Abschließend für die initialen Begriffsdefinitionen wird der Begriff der *Softwarekarte* eingeführt, der die Obermenge der in dieser Arbeit zu entwickelnden graphischen Modellen zur Architekturdokumentationen von Anwendungslandschaften darstellt und einen schichtenartigen Aufbau besitzt:

> **Softwarekarte :** Eine Softwarekarte ist ein graphisches Modell zur Architekturdokumentation der Anwendungslandschaft oder von Ausschnitten dieser. Eine Softwarekarte setzt sich zusammen aus einem Kartengrund und den auf dem Kartengrund aufbauenden Schichten, die verschiedene Merkmale visualisieren.

1.4. Aufbau

Als Grundlage wird die Softwarekartographie in ihren Anwendungsbereich, das Management von Anwendungslandschaften, in Kapitel 2 eingeordnet. Es werden verschiedene Managementprozesse, z. B. das Management der Unternehmensarchitektur, das IT-Architekturmanagement oder das Projektportfoliomanagement, vorgestellt, um eine Abgrenzung verschiedener Managementprozesse für Anwendungslandschaften vorzunehmen, den Zusammenhang der Prozesse darzustellen und die Stellung der Softwarekartographie zu verdeutlichen.

Neben der Vorstellung der Managementprozesse werden verschiedene Frameworks für Unternehmensarchitekturen betrachtet, um einen Vergleich zwischen bekannten und neuartigen Ansätzen herzustellen.

Anknüpfend an die Vorstellung der IT-Managementprozesse und dem damit verbundenen Einsatzgebiet für die Visualisierungen in der Softwarekartographie, werden in Kapitel 3 diese Visualisierungen aufgegriffen, systematisch analysiert und kategorisiert.

Nach einer Einführung in das Thema *Architekturdokumentation* und in den Aufbau von Modellierungssprachen werden in der Praxis verbreitete Visualisierungen zum Management von Anwendungslandschaften und weiterführende Anforderungen an diese Visualisierungen betrachtet, um auf Basis dieser Analyse die Softwarekartentypen als Klassifikationsmittel einzuführen. Eine Einführung zur thematischen Kartographie legt notwendige Grundlagen für die Softwarekartentypen und verdeutlicht zusätzlich die Wahl des Namens *Softwarekartographie*.

Aufbauend auf den Softwarekartentypen werden in Abschnitt 4 die semantischen und symbolischen Modelle von Softwarekarten eingeführt und ihr Zusammenhang dargelegt. Wesentliche Strukturen eines Informationsmodells, als Metamodell für das semantische Modell einer Softwarekarte, werden diskutiert, um auf der Diskussion aufbauend einen Verbesserungsvorschlag auf der Basis von Mustern vorzustellen.

Das Visualisierungsmodell, als Metamodell für symbolische Modelle, beschreibt, wie Softwarekarten objektorientiert beschrieben werden und mehrdeutige Interpretationen von Softwarekarten mittels einer engen Kopplung an die semantischen Informationen vermieden werden. Abschließend wird in Kapitel 4 die Softwarekartographie existierenden Modellierungssprachen gegenübergestellt und von diesen abgegrenzt.

In Kapitel 5 werden verschiedene Anwendungsfälle für Softwarekarten erarbeitet und Gestaltungsprinzipien für Anwendungslandschaften eingeführt. Hieraus abgeleitet werden Anforderungen an eine geeignete Werkzeugunterstützung zum Management von Anwendungslandschaften, welche in einer Studie über Werkzeuge zum Management von Unternehmensarchitekturen eingeflossen sind. Abschließend wird in Kapitel 5 ein Werkzeug für die Softwarekartographie vorgestellt, welches auf einer Modelltransformation zur Kopplung semantischer und symbolischer Modelle basiert und sich zur Generierung von Softwarekarten eignet.

Den Abschluss der Arbeit bildet eine bewertende Zusammenfassung und ein Ausblick in Kapitel 6.

KAPITEL 2

Management von Anwendungslandschaften

Inhaltsverzeichnis

Der Begriff *Management* wird sowohl in der Literatur als auch umgangssprachlich vielfältig und unterschiedlich verwendet. Vom etymologischen Ursprung des Wortes Management, dem lateinischen Wort *manus* für Hand, bis zum Management-Begriff nach Taylor,

der als Vater der Managementwissenschaft gilt [Br99, En06b], hat der Begriff bereits eine deutliche Wandlung vollzogen. Heute wird der angloamerikanische Begriff Management im betriebswirtschaftlichen Sprachgebrauch im Sinne von *Leitung* oder *Führung* verwendet [Br99, Ga04]. Weitere Definitionen und Diskussionen finden sich u. a. für die Wirtschaftsinformatik bei Krcmar [Kr05], der den Begriff insbesondere im Kontext des Informationsmanagements diskutiert.

Unter dem Management von Anwendungslandschaften, welches eine Aufgabe im Rahmen des Managements der Informationstechnologie, also des IT-Managements ist, wird in dieser Arbeit das *Führen* der Anwendungslandschaft im funktionalen Sinne (vgl. Krcmar [Kr05, S. 23]) verstanden. Tätigkeiten beim Management von Anwendungslandschaften sind beispielsweise das Fällen von Entscheidungen, die nicht nur interne Veränderungen eines Anwendungssystems hervorrufen, sondern Auswirkungen auf andere Teile einer Anwendungslandschaft haben oder angrenzende Bereiche wie Geschäftsprozesse, Services etc. betreffen.

Im Fokus dieser Arbeit stehen Modelle und Methoden zur Beschreibung, Bewertung und Gestaltung von Anwendungslandschaften, die das Management (von Anwendungslandschaften) unterstützen. Diese Modelle und Methoden sind als Zulieferer bzw. Unterstützungsfunktionen zum Management von Anwendungslandschaften zu verstehen. Eine Managemententscheidung – also eine Führungsentscheidung – wird aus einer *Intuition* des Managers heraus getroffen und/oder wird durch Entscheidungshilfen abgestützt. Die beschriebenen Modelle und Methoden dienen als Entscheidungshilfen.

Im folgenden Abschnitt 2.1 wird eine Methode zum Management von Anwendungslandschaften vorgestellt, die ebenso die Themen Strategie- und Zielemanagement, Anforderungsmanagement, Projektportfoliomanagement, Synchronisationsmanagement, Management der Unternehmensarchitektur (engl. *Enterprise Architecture Management*) sowie IT-Architekturmanagement einbezieht.

Die Rolle des Themas IT-Governance, welches die Strukturierung der einzelnen Managementprozesse und die Rollen innerhalb der Prozesse adressiert, wird nach der Einführung der Managementprozesse in Abschnitt 2.2 dargestellt.

Die Anzahl verwandter Ansätze zum Management von Anwendungslandschaften, die in der Literatur beschrieben oder von Werkzeugherstellern implementiert wurden, ist bisher vergleichsweise gering, jedoch ist in jüngster Vergangenheit eine Steigerung erkennbar. Abschnitt 2.3 diskutiert insbesondere die verwandten Ansätze des Werkzeugherstellers alfabet AG [al06] und von Keller [KNS92] sowie die Frameworks von Zachman [Za87] und Scheer [Sc96]. In der Praxis ist die Zahl der Vorträge in diesem Themenbereich ebenso angestiegen, so skizzieren Struck [St05b] von der Deutschen Bahn und Weber [We06] von Detecon Consulting verwandte Ansätze, dies jedoch nur in Form von Vortragsunterlagen.

2.1. Managementprozesse für Anwendungslandschaften

Das Management von Anwendungslandschaften findet nicht autark statt, sondern im Kontext von anderen Managementprozessen. Die Anwendungssysteme in einer Anwendungslandschaft unterstützen beispielsweise die Geschäftsprozesse eines Unternehmens (*Supporter*), oder sie ermöglichen erst die Umsetzung eines Geschäftsprozesses (*Enabler*). Der Managementprozess dieser Geschäftsprozesse, dessen Zweck die Entwicklung und Umsetzung von Geschäftsprozessen zur Verwirklichung von Geschäftszielen ist, wird unter Geschäftsprozessmanagement subsumiert[1].

Neben diesem skizzierten Prozess existieren weitere Managementprozesse, welche die Anwendungslandschaft beeinflussen oder Veränderungen in ihrem Umfeld hervorrufen. Im Folgenden werden die Managementprozesse vorgestellt, in denen das Management der Anwendungslandschaft eingebettet ist[2].

Zielsetzung der folgenden Einführung verschiedener Prozesse und deren Aktivitäten ist das Beschreiben einer *Plattform*, die verschiedene Managementprozesse integriert und die IT-Governance unterstützt. Unter dem Begriff *Plattform* ist ein virtueller Verbund von Methoden und Werkzeugen zu verstehen, wobei offen bleibt, ob eine IT-gestützte Implementierung in einem einzelnen Werkzeug oder mittels mehrerer Werkzeuge erfolgt. Werden mehrere Werkzeuge eingesetzt, so sind die Schnittstellen zwischen den Werkzeugen, die sich in der Integration der Managementprozesse widerspiegeln, von besonderer Bedeutung.

2.1.1. Motivation für die Plattform

Große Unternehmen nutzen bis zu mehrere tausend betriebliche Anwendungssysteme, die unterschiedliche Geschäftsprozesse unterstützen und an verschiedenen Standorten weltweit eingesetzt und betrieben werden. Dabei ist die Anwendungslandschaft nicht starr, sondern verändert sich stetig durch neue oder geänderte Anforderungen, die aus dem Geschäft oder aus der IT selbst resultieren. IT-Projekte setzen diese Anforderungen um und verändern somit die Anwendungslandschaft.

Die Größe der Anwendungslandschaft und die Anzahl von IT-Projekten verlangt nach einem geeigneten Steuerungsinstrument, um Anforderungen einander gegenüberstellen zu können, die Transparenz der IT zu erhöhen, die Wiederverwendung existierender Lösungen zu vereinfachen sowie Komplexität zu managen. Im Falle der Projektpartner des Forschungsprojektes Softwarekartographie kann eine derartige Plattform jedoch nicht auf der sprichwörtlichen *grünen Wiese* entstehen, sondern muss existierende und teilweise durch Softwaresysteme unterstützte (IT)-Managementprozesse berücksichtigen und einbinden.

[1] Im Englischen ist der korrespondierende Begriff das *Business Process Management* kurz BPM.
[2] Die Abschnitte 2.1.1 bis 2.1.9 basieren auf Wittenburg et al. [Wi07] und Fischer et al. [FMW05].

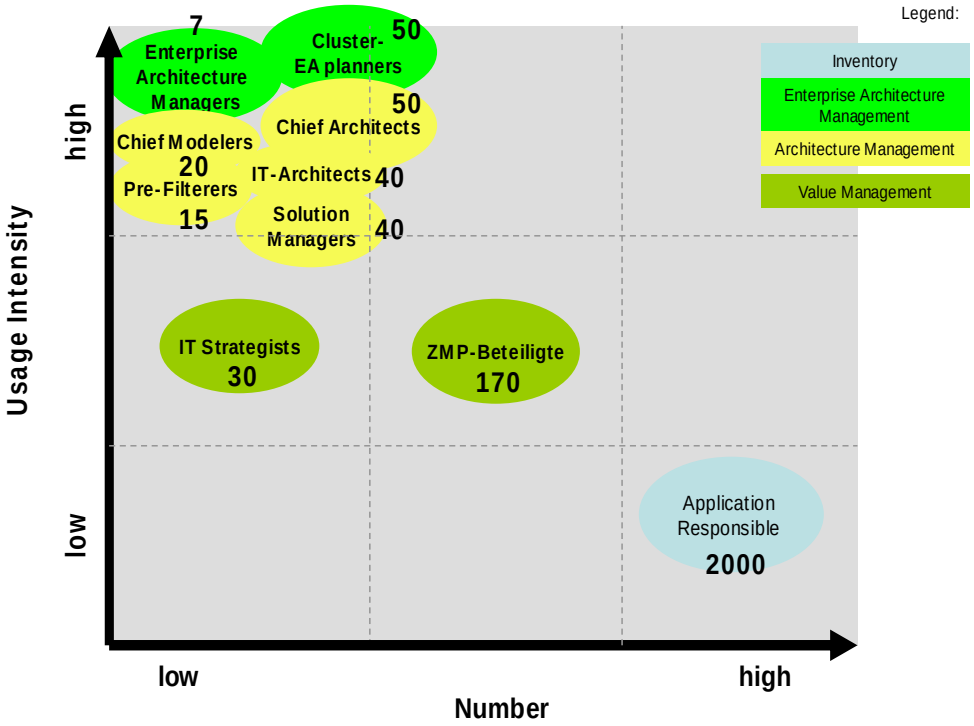


Abbildung 2.1.: Benutzerstruktur einer Plattform zum Management der Anwendungslandschaft [Le05]

Die Strukturierung der Prozesse ergibt sich zum einen durch das Separieren nach funktionalen Gesichtspunkten (*Separation of Concerns*) und zum anderen durch die Rollen, die in die Prozesse involviert sind.

Die Benutzerstruktur einer derartigen Plattform wird von Lentrodt [Le05] derart skizziert, dass es innerhalb der BMW Group ca. 200 Nutzer mit hoher Nutzungsintensität, weitere ca. 200 Nutzer mit mittlerer Nutzungsintensität und ca. 2000 Nutzer mit geringer Nutzungsintensität gibt (siehe Abbildung 2.1). Diese Größenordnungen zeigen, dass eine IT-Umsetzung der Plattform eine Menge von ca. 2400 Nutzern unterstützen muss, die mit unterschiedlichen Profilen, in unterschiedlichen Organisationseinheiten, und unterschiedlicher Nutzungsintensität gemeinsam auf der Plattform arbeiten.

Ziel der Plattform ist es, die gesamte Anwendungslandschaft zu dokumentieren, die Weiterentwicklung selbiger zu planen, Verbesserungspotential in der Anwendungslandschaft zu identifizieren und die gegenseitige Ausrichtung von Geschäft und IT zu optimieren. Typische Fragen, die mittels der Plattform beantwortet werden sollen, sind die folgenden:

- Welche Anwendungssysteme werden genutzt und wo werden sie betrieben?
- Wo sind Schwachstellen und ungenutzte Potentiale?

- Wo und wie werden Anwendungssysteme verändert und wie sieht die Anwendungslandschaft nach der Veränderung aus?
- Wie wird die Anwendungslandschaft kurzfristig und langfristig aussehen?
- Wie kann die Anwendungslandschaft an Anforderungen richtig angepasst werden?
- Wie soll sich die Anwendungslandschaft entwickeln?
- Wie optimiere ich die gegenseitige Ausrichtung von Geschäft und IT?

Die Abschnitte 2.1.2 bis 2.1.8 beschreiben die einzelnen Managementprozesse und wie diese miteinander integriert werden. Losgelöst von den Prozessen und ihrer Integration ist bei der folgenden Vorstellung die Organisationsstruktur des Unternehmens. Die BMW Group besitzt beispielsweise hinsichtlich der IT eine föderale Aufbauorganisation, bestehend aus zentralen und dezentralen IT-Abteilungen[3], wobei die dezentralen IT-Abteilungen den unterschiedlichen Geschäftsbereichen, wie Forschung, Entwicklung, Vertrieb etc., zugeordnet sind. Ähnliche Strukturen, die teilweise produktgetrieben sind, lassen sich beispielsweise bei der Allianz (Allianz Lebensversicherung, Euler Hermes, Dresdner Bank, Fireman's Fund etc.), der Deutsche Post World Net (Brief, Express & Logistik, Finanz Dienstleistungen) oder auch Gillette[4] (Grooming, Batteries/Flashlights, Oral Care etc.) erkennen. Die zentralen IT-Abteilungen übernehmen typischerweise Querschnittsfunktionen wie Betrieb, Support, IT-Strategie etc., wobei es hier zur Doppelung der Funktionen kommen kann, so dass sich sowohl die zentrale als auch die dezentralen IT-Abteilungen beispielsweise dem Thema IT-Strategie widmen.

In der Plattform wird die Organisationsstruktur zur Definition von Rollen und Berechtigungen abgebildet, die sich aus der Gestaltung der IT-Governance (siehe Abschnitt 2.2) ergibt. Die Entscheidung, ob beispielsweise ein Szenario für ein Projektportfolio (siehe Abschnitt 2.1.3) genehmigt wird und wer diese Genehmigung erteilen darf, ist bestimmt durch die Verantwortlichkeiten in der Organisationsstruktur.

Einen Überblick über die im Folgenden eingeführten Prozesse zum Management von Anwendungslandschaften gibt die Abbildung 2.2. Nachdem die einzelnen Managementprozesse sukzessiv eingeführt wurden, wird in Abschnitt 2.1.8 die Integration der Prozesse und die Interaktion mit dem IT-Projektlebenszyklus detailliert beschrieben. Als gegeben für die folgenden Managementprozesse wird ein existierendes Vorgehensmodell für IT-Projekte vorausgesetzt.

2.1.2. Anforderungsmanagement

Der Prozess für das Anforderungsmanagement ist der Einstiegspunkt für neue IT-Maß nahmen, die in einem oder mehreren IT-Projekten resultieren können. Neue Maßnahmen

[3] Das Modell von dezentralen und zentralen IT-Abteilungen in der IT-Aufbauorganisation wird auch als koordiniertes Modell bezeichnet (siehe [BES04, BBB03]). Die Zuständigkeiten und die Trennung von *Demand* und *Supply* ergänzen das Modell [Ma06a].

[4] Gillette gehört seit 2005 zur Procter & Gamble Company.

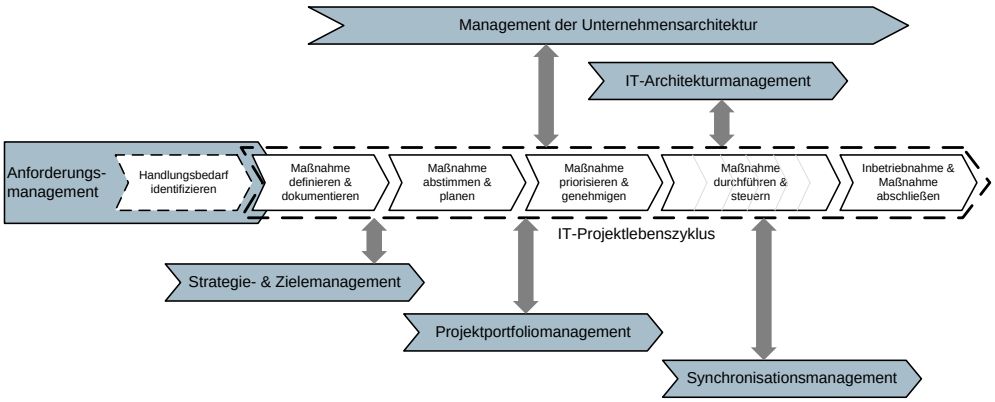


Abbildung 2.2.: Prozesse zum Management von Anwendungslandschaften

müssen in diesem Managementprozess in einer einheitlichen Form erfasst und dokumentiert werden. Beispiele für eine derartige Maßnahme sind die Einführung von RFID in einer Warenhauskette, das Auslagern der Zahlungsverkehrsabwicklung bei einer Bank oder auch die Migration eines Datenbanksystems.

Wie Abbildung 2.2 zu entnehmen ist, besteht eine Verzahnung mit dem IT-Projektlebens zyklus (Phase *Maßnahme definieren & dokumentieren*). Hervorzuheben ist, dass die Funktionalität dieses Prozesses nicht mit dem Anforderungsmanagement bzw. *Requirements Engineering* im Software-Engineering zu verwechseln ist. Es ist nicht Aufgabe dieses Prozesses, eine umfassende Anforderungsanalyse durchzuführen, die u. a. zu einer *Software Requirements Specification* nach dem IEEE 830-1998 [IE98] führt, sondern es ist das Ziel, die Portfolioentscheidung vorzubereiten bzw. zu ermöglichen.

Die Dokumentation einer Maßnahme findet semi-strukturiert in einer textuellen Form statt, wobei neben der eigentlichen Maßnahmenbeschreibung auch ein Ansprechpartner angegeben werden muss. Zusätzlich wird eine Verknüpfung zu den Elementen des Prozesses Strategie- und Zielemanagement (siehe Abschnitt 2.1.4) hergestellt, indem hinterlegt werden muss, welche Strategien und Ziele mit der Anforderung verfolgt bzw. unterstützt werden. Dieses Verknüpfen ermöglicht zum Zeitpunkt des Erstellens des kommenden Projektportfolios sowohl eine zusätzliche Bewertungsmöglichkeit als auch eine Nachvollziehbarkeit, welches einen wesentlichen Vorteil nach dem Umsetzen einer Anforderung ergibt. Es ist somit möglich, nachzuvollziehen, warum eine Anwendungssystemversion entstanden ist und welche Ziele mit der Anforderung verbunden waren.

Ist des Weiteren zum Zeitpunkt der Eingabe der Anforderung bereits bekannt, welche Elemente der Unternehmensarchitektur (z. B. ein betriebliches Anwendungssystem oder ein Infrastrukturelement) verändert werden, so ist dies ebenso zu dokumentieren. Zusätzlich erlaubt es die Methode, eine Anforderung zu bewerten, wobei die Bewertungskriterien für alle Anforderungen gleichartig sind. Als Beispiele für derartige Kriterien seien Verhältniszahlen zwischen strategischem vs. operativem Einfluss und problemlösend vs. problemvorbeugend genannt.

Das Sammeln dieser Informationen ermöglicht es dem Nutzer ähnliche Anforderungen,

die auch von verschiedenen Organisationseinheiten eingestellt worden sein können, zu identifizieren. Abfragen an die Plattform suchen beispielsweise nach gleichen assoziierten Architekturelementen oder Zielen und identifizieren diese, so dass ähnliche Anforderungen gebündelt werden. Wurden auf diesem Wege alle Anforderungen evaluiert, ähnliche Anforderungen identifiziert und gesammelt, kann der Status einer Anforderung auf *dokumentiert* gesetzt werden. Ein Komitee (z. B. ein Lenkungskreis) kann abschließend über Anforderungen entscheiden und diese entweder *ablehnen*, *annehmen* oder *zurück delegieren*. Wird eine Anforderung angenommen, so übernimmt das Projektportfoliomanagement die Anforderung.

2.1.3. Projektportfoliomanagement

Das Projektportfoliomanagement, welches in diesem Fall das IT-Projektportfolio und nicht ein (IT-) Produktportfolio betrifft, soll das optimale Portfolio von IT-Projekten zusammenstellen. Die Portfolioanalyse, die ihre Wurzeln bei Markowitz in der Finanzwirtschaft [Ga04] hat, sucht hierbei mittels einer Analysemethode das Optimum im Gesamtkontext des Unternehmens. Hierzu werden bei diesem Ansatz zunächst mehrere Anforderungen in Projektanträge zusammengefasst und detailliert dokumentiert. Die Dokumentation umfasst hierbei neben Arbeitspaketen, Meilensteinen etc. auch Kennzahlen der Projektanträge, wie beispielsweise Projektrisiko, Projektkosten und Projektlaufzeit. Diese Informationen sind typischerweise ebenso Teil einer Projektdokumentation entlang eines Projektvorgehensmodells (*Maßnahme definieren & dokumentieren*), so dass diese im Projektportfoliomanagement wiederverwendet werden. Ziel ist es auch hier, analog zum Anforderungsmanagement, einen Gesamtüberblick über alle Projektanträge zu erhalten, um diese im Gesamtkontext zu analysieren und zu evaluieren.

Eine Analysemethode im IT-Projektportfolio ist beispielsweise die Abhängigkeitsanalyse, die anhand dokumentierter Projektanträge Abhängigkeiten zwischen IT-Projekten identifiziert. In Organisationen, die über dezentrale IT-Abteilungen und ein großes IT-Projektportfolio verfügen, ermöglichen Abhängigkeitsanalysen beispielsweise, Projekte zu identifizieren, die gleiche betriebliche Anwendungssysteme in ihrem Projektkontext haben. Dabei sollte beispielsweise ein Projekt, welches eine Schnittstelle eines betrieblichen Anwendungssystems in seinem Projektkontext anbinden will, mit einem anderen Projekt, welches diese Schnittstelle modifiziert, entsprechend synchronisiert werden.

Bei den Evaluierungsmethoden werden des Weiteren Matrix-artige Darstellungen, im Folgenden als *Portfoliomatrix* bezeichnet, verwendet. Diese Darstellungen setzen verschiedene Merkmale mit einem mindestens ordinalen Skalenniveau zueinander in Beziehung. Abbildung 2.3 zeigt eine beispielhafte Portfoliomatrix, die in den Spalten den ökonomischen Wert (*Economic*) und in den Zeilen den strategischen Wert (*Strategic*) darstellt. Zusätzlich wird das Projektvolumen mittels der Größe eines Kreises und das Risiko mittels der Füllfarbe eines Kreises visualisiert.

Bei der Evaluierung des Portfolios werden Projekte, die sich in der Matrix in Abbildung 2.3 weit rechts oben befinden, und somit einen hohen ökonomischen und hohen

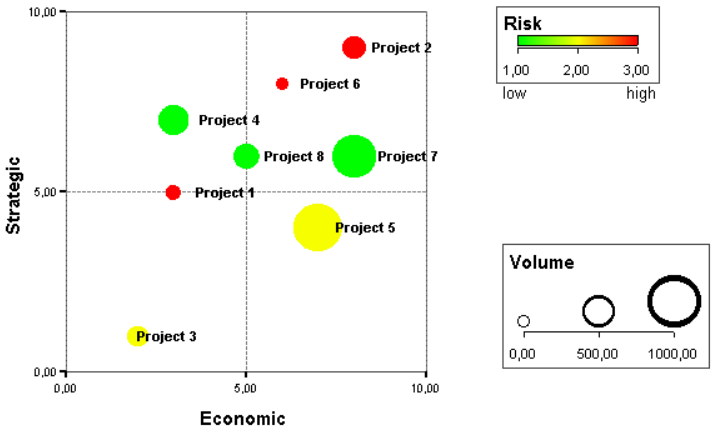


Abbildung 2.3.: Portfoliomatrix zur Bewertung eines Szenarios

strategischen Wert besitzen, genauer betrachtet. Bei der Findung des optimalen Portfolios müssen jedoch zusätzlich das Risiko und die Projektvolumina berücksichtigt werden, wobei das Gesamtbudget für IT-Projekte in der Planungsperiode die Obergrenze für die Summe der Projektvolumina darstellt. Wie eine Quantifizierung von Risiken möglich ist, zeigen beispielsweise Wehrmann et al. [WHS06].

Da das Verhältnis von ökonomischem zu strategischem Wert und weitere Merkmale von Interesse sind, existieren verschiedenste Variationen von Portfoliomatrizen, zu denen auch die prominenten Matrizen der BCG und von McKinsey zählen [Ga04]. Niemann [Ni05] zeigt Beispiele für derartige Portfoliomatrizen und deren Anwendung im Projektportfoliomanagement.

Weitere Variationen im Portfoliomanagement ergeben sich aus der Gegenüberstellung verschiedener Szenarios für Portfolios, die sich aus den Abhängigkeitsanalysen unter Berücksichtigung des Planungsbudgets ergeben können. Gibt es verschiedene gültige Kombinationen von Portfolios, so werden diese mit den selben Methoden evaluiert, um abschließend zu *einem* gültigen Portfolio zu gelangen.

Das Resultat dieses Planungsprozesses, implementiert durch die Prozesse Anforderungsmanagement und Portfoliomanagement, ist ein Portfolio von genehmigten IT-Projekten für eine Planungsperiode, welches in der Analyse eine Verbindung zum Strategie- & Zielemanagement (siehe Abschnitt 2.1.4), zum Synchronisationsmanagement (siehe Abschnitt 2.1.6) und zum Management der Unternehmensarchitektur (siehe Abschnitt 2.1.7) aufgebaut hat.

2.1.4. Strategie- & Zielemanagement

Nach Clausewitz [Cl80, S. 271] grenzt sich Strategie gegenüber Taktik durch den längerfristigeren Charakter ab: „Es ist also nach unserer Einteilung die Taktik die Lehre vom Gebrauch der Streitkräfte im Gefecht, die Strategie die Lehre vom Gebrauch der Gefechte zum Zweck des Krieges.“[5] Übersetzt auf die IT ergibt sich, dass eine strategische Entscheidung einen langfristigen Charakter besitzt. Es kommt hinzu, dass die IT-Strategie typischerweise aus der Geschäftsstrategie abgeleitet wird bzw. werden sollte [BES04, Ke06, Kr05]. Das Ausformulieren einer Strategie erfolgt hierbei zumeist informell und mittels natürlicher Sprache.

Um das Umsetzen einer Strategie zu bewerten bzw. zu beobachten, werden typischerweise Ziele einer Strategie zugeordnet, die wiederum eine messbare Größe darstellen. Der Prozess Strategie- & Zielemanagement übernimmt diese Aufgabe und ermöglicht es, Strategien zu dokumentieren, mit Zielen zu verbinden und sowohl Strategien als auch Ziele mit den Anforderungen und Projektanträgen des Anforderungsmanagements bzw. Portfoliomanagements zu assoziieren. Ziel dieses Prozesses ist es, Projekte an Strategien und Zielen auszurichten, um so die Zielerreichung zu dokumentieren und messbar zu machen.

Positive Seiteneffekte beim Verknüpfen von Anforderungen und Projektanträgen mit Strategien und Zielen sind Transparenz und Nachvollziehbarkeit. Auch nach Abschluss eines Projektes können die Strategien und Ziele identifiziert werden, die ein Projektantrag verfolgt.

Ein Ansatz zur Dokumentation, Kontrolle und Evaluierung von Strategien und Zielen basiert auf dem Konzept der *Balanced Scorecards* von Kaplan et al. [KN91]. Die zugehörigen Metriken zur Evaluierung von Zielen basieren auf unternehmensspezifischen Performanzindikatoren und Kosten/Nutzen-Analysen. Beispiele für derartige Metriken sind Kapitalwerte, Risikowerte oder auch strategischer Nutzen. Diese Metriken werden beim Projektportfoliomanagement (siehe Abschnitt 2.1.3) berücksichtigt, um zu einer Priorisierung innerhalb des Projektportfolios zu gelangen.

2.1.5. IT-Architekturmanagement

Da sich die Plattform nicht auf einzelne Anwendungssysteme konzentriert, sondern auf die Anwendungslandschaft als Ganzes, werden alle Entscheidungen im Kontext einer übergreifenden IT-Architektur getätigt. Das IT-Architekturmanagement entwickelt hierzu Leitlinien und Unternehmensstandards, wobei es sich in diesem Fall bei den Architekturelementen um einzelne Anwendungssysteme handelt.

Werden beispielsweise über tausend Anwendungssysteme betrieben, die verschiedene Datenbankmanagementsysteme, Applikationsserver, Webserver etc. nutzen, so entsteht eine Heterogenität, die nach geeigneten Managementmethoden verlangt, um diese zu steuern. Um einen *Wildwuchs*, bei dem jedes Anwendungssystem unterschiedliche Technologien

[5] Die Definition von Strategie durch Clausewitz hat nicht an Bedeutung verloren und wird beispielsweise auch vom Strategieinstitut der Boston Consulting Group [OGB03] verwendet.

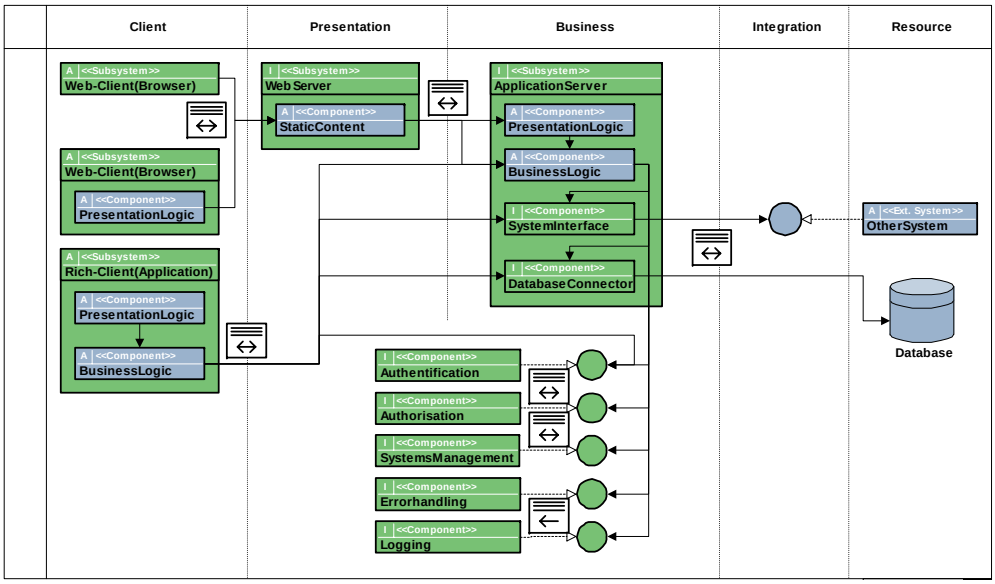


Abbildung 2.4.: Musterarchitektur *Online Transaction Processing* [St03]

einsetzt, zu vermeiden, wurden beispielsweise innerhalb der BMW Group so genannte Musterarchitekturen und -lösungen in einem Referenzmodell entwickelt [St03]. Ein weiterer Grund für die Entwicklung dieser Bibliothek von IT-Architekturen ist das Dokumentieren von Lösungen bzw. Lösungsvorschlägen, die sich bewährt haben. Basis eines Referenzmodells für IT-Architekturen sind beispielsweise die Muster von IBM [IB04] und SUN Microssystems [SU01], die für betriebliche Anwendungssysteme verschiedene Bausteine beschreiben.

Ein Beispiel für eine Musterarchitektur findet sich in Abbildung 2.4[6]. Die Abbildung zeigt eine Musterarchitektur für eine *Online Transaction Processing Application* bestehend aus den Stufen *Client*, *Presentation*, *Business*, *Integration* und *Resource*. Es handelt sich bei den einzelnen Subsystemen bzw. Komponenten (z. B. *Web Server*) in der Stufe (*Presentation*) um abstrakte Bausteine, die produktneutral sind. Eine Instantiierung dieser Musterarchitektur, als Musterlösung bezeichnet, ergänzt eine Musterarchitektur mit konkreten, einsetzbaren Produkten. Am Beispiel des Webservers kann dies ein Apache HTTP-Server 2.0 sein.

Für eine Musterarchitektur kann es mehrere Musterlösungen geben, die verschiedene Produkte bzw. Produktversionen enthält, wobei die Zahl der Musterlösungen zu einer Musterarchitektur gering gehalten wird, um den oben diskutierten Wildwuchs zu vermeiden. Ein weiterer Vorteil dieser Bibliothek liegt in der Übergabe zum IT-Betrieb, der eine dokumentierte Musterlösung unterstützt (siehe auch Abschnitt 4.2.3.2).

Neben der Sicht aus Abbildung 2.4 existieren weitere Sichten auf eine Musterarchitektur und Musterlösung, die verschiedene Interessen adressieren. Abbildung 2.4 zeigt eine

[6] Die Notation für Musterarchitekturen und Musterlösungen ist von der BMW Group entwickelt worden und entspricht keiner standardisierten Norm (siehe auch [St03]).

funktionale Sicht (*Functional Perspective*), die um weitere Sichten, wie beispielsweise eine *Deployment Perspective*, ergänzt wird [St03]. Weitere Beispiele für derartige Bibliotheken von Unternehmensstandards für Architekturen finden sich auch unter dem Namen *Softwarearchitekturbebauungsplan* bei der HVB Information Services [HV04].

2.1.6. Synchronisationsmanagement

Im Gegensatz zu den Managementprozessen in den Abschnitten 2.1.2 bis 2.1.5 steuert das Synchronisationsmanagement das dynamische Verhalten. Da Projekte sich gegenseitig beeinflussen können und zeitliche Verschiebungen in Projektplänen entstehen, bedarf es einer Steuerung der zeitlichen Abhängigkeiten von Projekten. Im Gegensatz zum Multi-Projektmanagement oder Programmmanagement ist es nicht die Aufgabe des Synchronisationsmanagements, eine gemeinsame Ressourcenplanung durchzuführen (Multi-Projektmanagement) oder ein gemeinsames und übergeordnetes Ziel mehrerer Projekte zu verfolgen (Programmmanagement). Das Synchronisationsmanagement muss stattdessen die zeitliche Abfolge von Projekten aufgrund fachlicher und technischer Abhängigkeiten planen und steuern.

Eine Aufgabe im Rahmen des Synchronisationsmanagement ist das frühzeitige Identifizieren von Projekten, die ihrem Zeitplan nicht folgen, und das Abstimmen dieser mit anderen als abhängig registrierten Projekten. Dies muss in Abstimmung mit dem Projektportfoliomanagement erfolgen, um die Budgetierung ggf. anzupassen. Eine weitere Aufgabe dieses Managementprozesses ist die Planung des Ausrollens (der *Roll-Out*) von neuen oder geänderten Applikationen an mehreren Standorten, welches unter der Maßgabe einer möglichst geringen Unterbrechung von Produktionsprozessen nicht immer gleichzeitig erfolgen kann.

Ein Werkzeug für das Synchronisationsmanagement verwendet die aus dem Projektmanagement bekannten Gantt-Diagramme, wobei diese mit der bekannten Erweiterung zur Darstellung von Vor- und Nachbedingungen durch Linien zwischen den Balken ergänzt sind.

2.1.7. Management der Unternehmensarchitektur

Für das Management von Anwendungslandschaften bildet der Prozess zum Management der Unternehmensarchitektur den *Klebstoff* für den Zusammenhalt mit den anderen, beteiligten Prozessen und führt zu einem integrierten Gesamtprozess. Bevor die Details des Zusammenspiels in Abschnitt 2.1.8 folgen, wird zunächst die Funktionsweise des Prozesses zum Management der Unternehmensarchitektur beschrieben.

Die Unternehmensarchitektur[7] als ganzheitliche und kohärente Architektursicht ermöglicht das Zusammenspiel verschiedener Elemente zu erkennen, die eigene Managementprozesse besitzen. Geschäftsprozesse, Infrastrukturelemente, Projekte, IT-Architekturen, Strategien etc. sind in einer Unternehmensarchitektur miteinander in Beziehung gesetzt

[7] Für die Herleitung des Begriffs *Unternehmensarchitektur* wird auf Abschnitt 2.2 verwiesen.

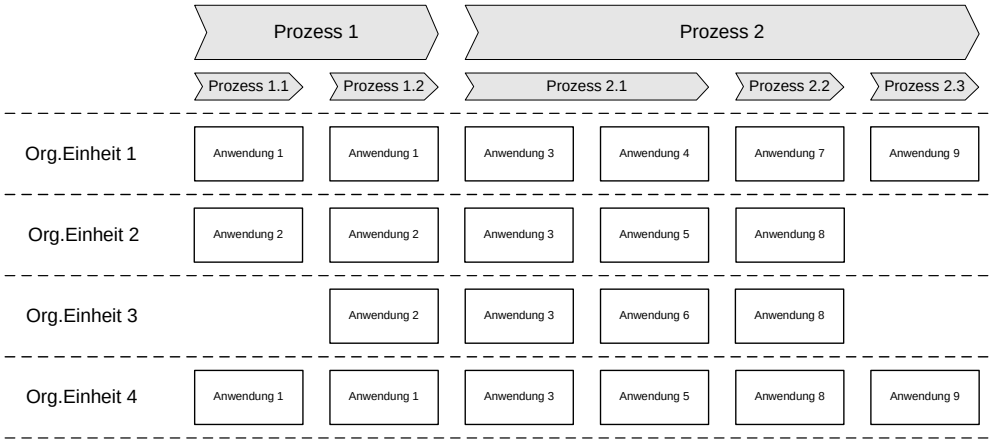


Abbildung 2.5.: Exemplarische Softwarekarte zur Darstellung der Geschäftsprozessunterstützung durch Anwendungssysteme bei verschiedenen Organisationseinheiten

und werden mittels einer Architekturdokumentation bestehend aus verschiedenen Sichten und Modellen beschrieben. Somit erlaubt es die Unternehmensarchitektur, Wechselwirkungen und Beziehungen zwischen den Elementen zu erkennen, die in den einzelnen Dokumentationen der anderen Managementprozesse nicht erkennbar sind.

Das Management der Unternehmensarchitektur besitzt somit die Aufgabe, das Zusammenspiel dieser verschiedenen Elemente eines Unternehmens auf einer Architekturebene zu dokumentieren, zu verwalten und zu steuern. Dieser Prozess ist eng verzahnt mit den anderen Prozessen und nutzt zum einen Informationen der einzelnen Prozesse und liefert zum anderen Informationen für diese Prozesse (siehe Abschnitt 2.1.8). Beim Anforderungsmanagement (siehe Abschnitt 2.1.2) werden beispielsweise die von der Anforderung betroffenen Elemente der Unternehmensarchitektur dokumentiert.

Als Werkzeug in dem Prozess zum Management der Unternehmensarchitektur werden Darstellungen der Unternehmensarchitektur verwendet, die in Abschnitt 3.4.1 kategorisiert und hinsichtlich des Aufbaus detailliert beschrieben werden. Zur Einführung in derartige Darstellungen ist in Abbildung 2.5 eine exemplarische Softwarekarte dargestellt, welche die Unterstützung von Geschäftsprozessen bei unterschiedlichen Organisationseinheiten durch unterschiedliche betriebliche Anwendungssysteme zeigt.

Abbildung 2.5 ist zu entnehmen, dass das betriebliche Anwendungssystem *Anwendung 1* den Geschäftsprozess *Prozess 1.1* unterstützt und von den Organisationseinheiten *Org.Einheit 1* und *Org.Einheit 4* genutzt wird. Ebenso ist zu erkennen, dass die *Org.Einheit 2* für denselben Geschäftsprozess ein anderes Anwendungssystem (*Anwendung 2*) zur Unterstützung einsetzt und die *Org.Einheit 3* kein Anwendungssystem für diesen Prozess verwendet.

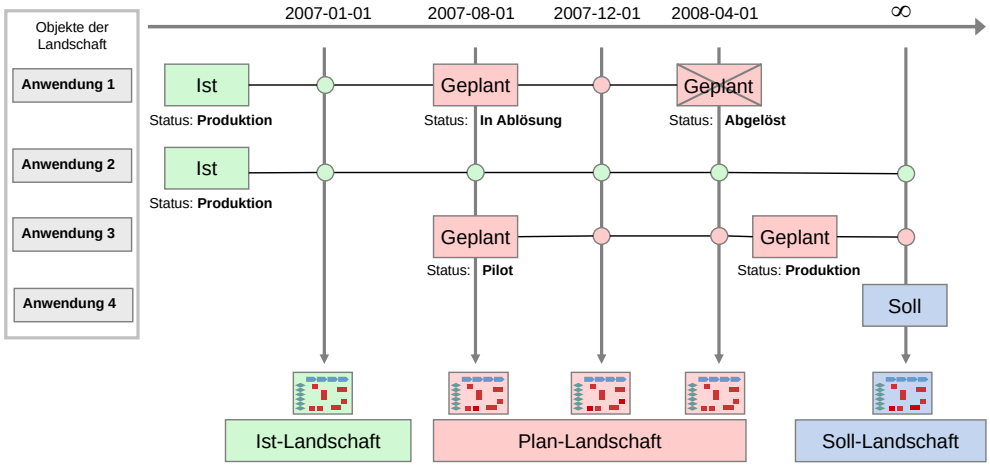


Abbildung 2.6.: Entwicklung von Objekten in Ist-, Plan- und Soll-Landschaften

Die Darstellungsart in Abbildung 2.5 wird zur Darstellung der Ist-Landschaft, mehrerer Plan-Landschaften und der Soll-Landschaft genutzt:

- Bei der Ist-Landschaft handelt es sich um den Status quo zum Zeitpunkt der Analyse.
- Eine Plan-Landschaft beschreibt einen Zustand in der Zukunft, der sich durch geplante oder bereits laufende Projekte zu einem bestimmten Zeitpunkt ergibt, wobei diese Projekte die Ist-Landschaft in die Plan-Landschaft überführen. Durch die zeitliche Abfolge von Projekten und deren Abschlussdatum existieren mehrere Plan-Landschaften, je nach Zeitpunkt des Plan-Zustandes, z. B. 2007-08-01.
- Die Soll-Landschaft ist ein angestrebter Ziel-Zustand, der sich aus den IT-Strategien ableitet und genutzt wird, um beispielsweise neue Anforderungen mit der abzustimmen.

Abbildung 2.6 zeigt den Zusammenhang von Ist-, Plan- und Soll-Landschaften und die verschiedenen Möglichkeiten der Entwicklung von Objekten der Landschaft. Das Objekt *Anwendung 1* ist beispielsweise am 1. Januar 2007, dem Datum der Ist-Landschaft, im Status *Produktion* und verändert sich in der Plan-Landschaft über den Status *in Ablösung* am 1. August 2007 bis hin zu *abgelöst* am 1. April 2008. Die anderen drei Beispiele für die Anwendungssysteme *Anwendung 2*, *Anwendung 3* und *Anwendung 4* zeigen weitere Möglichkeiten für die Entwicklung von Objekten in den Ist-, Plan- und Soll-Landschaften.

Der Nutzen derartiger Betrachtungen von Ist-, Plan- und Soll-Landschaften liegt zum einen in der erhöhten Transparenz, die nicht nur den Status quo sondern auch geplante Zustände betrifft, und zum anderen in der Verbesserung der Planung der Unternehmensarchitektur. Plan-Landschaften werden mit der Soll-Landschaft verglichen, um festzustellen, ob der mit den Plan-Landschaften eingeschlagene Weg der Weiterentwicklung der Landschaft mit den Vorgaben der Soll-Landschaft konform ist.

Ebenso erhöht sich die Planungssicherheit für Projekte, da diese nicht auf dem Status quo aufsetzen müssen, sondern bereits Veränderungen der Unternehmensarchitektur im projekteigenen Planungshorizont berücksichtigen. Projekte, die beispielsweise ein geplantes Projektende nach dem 1. April 2008 haben (vgl. Abbildung 2.6), erkennen anhand der Plan-Landschaft, dass *Anwendung 1* bereits abgelöst und eine geplante Schnittstellenanbindung an diese Applikation nicht sinnvoll ist, sondern stattdessen der Nachfolger anzubinden ist.

Weitere Anwendungsgebiete von Darstellungen wie in Abbildung 2.5 beziehen sich auf die Planung von Veränderungen hinsichtlich der Geschäftsprozessunterstützung bei verschiedenen Organisationseinheiten. Hier wird zwischen den Operationen Integration, Entkoppelung und Einführung unterschieden:

Integration bezeichnet hierbei das Ausdehnen der Geschäftsprozessunterstützung durch ein Anwendungssystem (horizontale Integration) oder das Erhöhen der Anzahl von Nutzern von einem Anwendungssystem für den selben Geschäftsprozess (vertikale Integration)[8].

Entkoppelung unterscheidet zwischen vertikaler bzw. horizontaler Entkoppelung und ist das Gegenteil zur Integration.

Einführung ist das Füllen eines *weißen Flecks* in der Geschäftsprozessunterstützung durch die Einführung einer Prozessunterstützung für die entsprechende Organisationseinheit durch ein Anwendungssystem.

Vertikale Integration kann beispielsweise in der Landschaft aus Abbildung 2.5 dadurch erreicht werden, dass das System *Anwendung 7* durch die *Anwendung 8* abgelöst wird, so dass alle Organisationseinheiten für den Geschäftsprozess *Prozess 2.2* dasselbe Anwendungssystem nutzen. Weitere Operationen, die Veränderungen in der Unternehmensarchitektur bedingen, werden in Abschnitt 5.1 vorgestellt.

2.1.8. Integration der Prozesse

Abbildung 2.7 zeigt das Zusammenspiel der einzelnen Prozesse, die in den Abschnitten 2.1.2 bis 2.1.7 beschrieben wurden, und dem IT-Projektlebenszyklus, welcher in der Mitte der Abbildung platziert ist. Die Interaktion der einzelnen Prozesse resultiert in einem kontinuierlichen Prozess, der verschiedene Rollen (Anwendungseigner, Projektleiter, Portfoliomanager etc.) einbezieht. Ziel der Darstellung in Abbildung 2.7 ist es nicht, jeden Prozess im Detail zu beschreiben, sondern das Zusammenspiel zu verdeutlichen.

Eine neue Anforderung wird zunächst im Anforderungsmanagement aufgenommen und bearbeitet, wobei die dokumentierten Anforderungen als IT-Maßnahmen, wie in Abschnitt 2.1.2 beschrieben, abschließend einheitlich bewertet werden. Wurde eine Anforderung *angenommen*, wird im nächsten Schritt ein Ticket erstellt, in dem die Maßnahme detaillierter beschrieben wird. Dies ist notwendig, da die nachfolgenden Schritte es erfordern, dass die Maßnahme im Maßnahmenticket zusätzlich die betroffenen Elemente der

[8] siehe hierzu auch Abschnitt 5.1

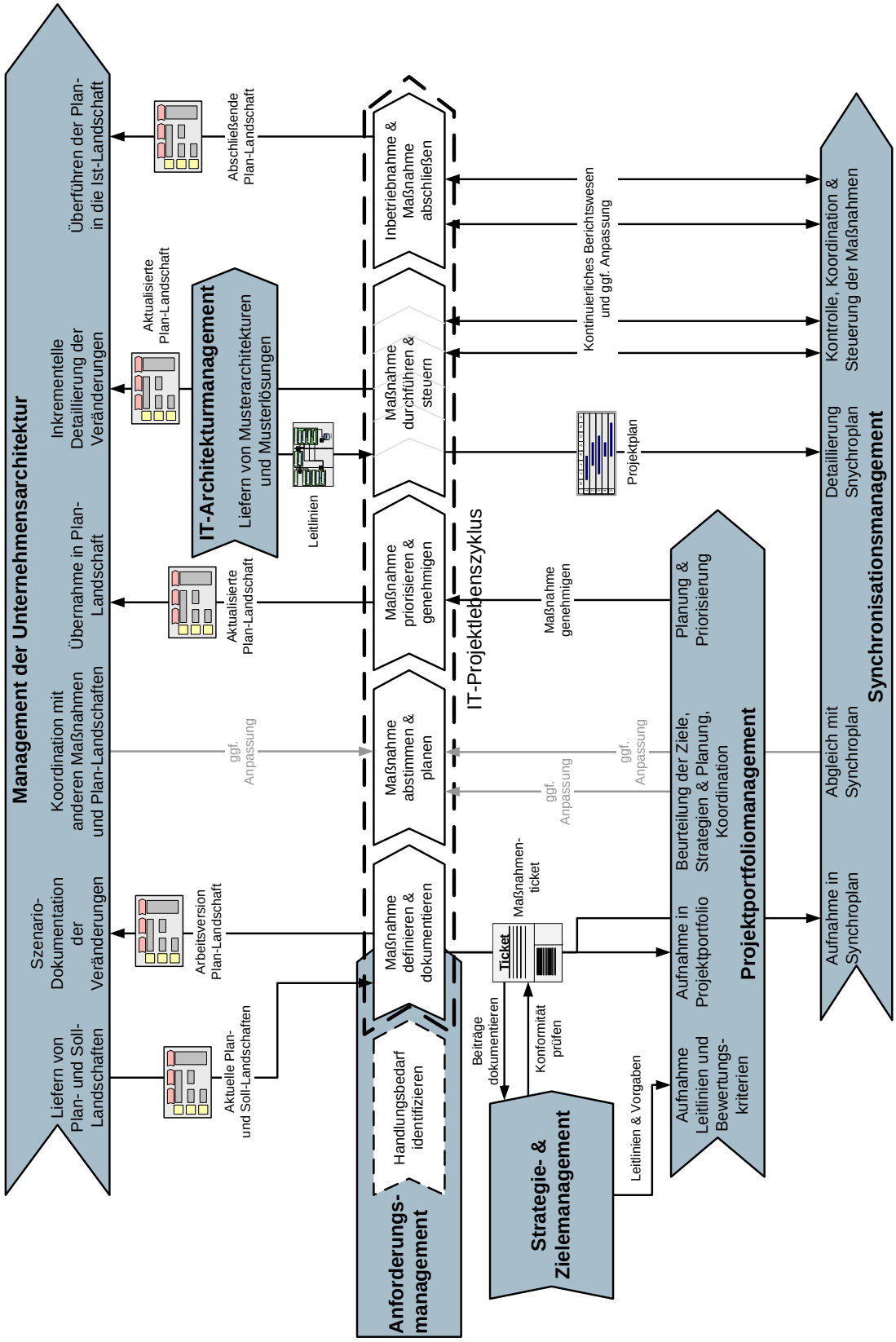


Abbildung 2.7.: Integration der Prozesse zum Management von Anwendungslandschaften

Plan- und Soll-Landschaften berücksichtigt, um die Anforderung im Projektportfolio- (siehe Abschnitt 2.1.3) und im Synchronisationsmanagement (siehe Abschnitt 2.1.6) entsprechend aufzunehmen. Zusätzlich ist die Anforderung mit den Strategien und Zielen (siehe Abschnitt 2.1.4) zu verbinden.

Im Projektportfoliomanagement wird anhand des Maßnahmentickets ein neuer Projektvorschlag erstellt, welcher, wie in Abschnitt 2.1.3 beschrieben, mit den anderen Projektvorschlägen im Projektportfolio verglichen und bewertet wird. Gleichzeitig wird der Projektvorschlag in den so genannten *Synchronisationsplan* des Synchronisationsmanagements aufgenommen und eine Arbeitsversion der Plan-Landschaft erstellt, welche die Veränderungen durch das vorgeschlagene Projekt enthält.

Entsprechend des IT-Projektlebenszyklus muss die Maßnahme anschließend abgestimmt und geplant werden (*Maßnahme abstimmen & planen*); dies insbesondere im Hinblick auf das existierende Portfolio und den Synchronisationsplan. Das Projektportfoliomanagement kann Änderungen in der Maßnahme verlangen, die beispielsweise aus Budgetgründen und/oder dem Bündeln von mehreren Maßnahmen resultieren, während das Synchronisationsmanagement Veränderungen an der Zeitplanung einer Maßnahme fordern kann, wenn es Abhängigkeiten (z. B. fachlicher Natur oder durch Ressourcenverfügbarkeit) gibt. Ebenso muss das Management der Unternehmensarchitektur prüfen, ob die Maßnahme mit den existierenden Plan-Landschaften und der Soll-Landschaft in Einklang steht, indem Zielvorgaben der Soll-Landschaft berücksichtigt werden und Abhängigkeiten in den Plan-Landschaften analysiert werden.

Ist eine Maßnahme abgestimmt und geplant, so folgt die Priorisierung und Genehmigung (*Maßnahme priorisieren & genehmigen*), wobei die Genehmigung auch eine Ablehnung der Maßnahme bedeuten kann. Die *positive* Genehmigung einer Maßnahme durch das Projektportfoliomanagement führt automatisch zu einer neuen Plan-Landschaft, in welche die Veränderungen der Maßnahme als genehmigt eingehen und somit für alle anderen Maßnahmen, die sich z. B. erst in der Definition (*Maßnahme definieren & dokumentieren*) oder der Abstimmung (*Maßnahme abstimmen & planen*) befinden, als genehmigt sichtbar sind.

Zu Beginn der Durchführung einer Maßnahme (*Maßnahme durchführen & steuern*) wird der detaillierte Projektplan erstellt, der eine Detaillierung des Synchronisationsplans beinhaltet. Das IT-Architekturmanagement (siehe Abschnitt 2.1.5) unterstützt die Umsetzung der Maßnahme zusätzlich mittels Leitlinien, die beispielsweise Einfluss auf die Softwarearchitektur und die verwendeten Komponenten haben. Während der Durchführung der Maßnahme erfolgt eine kontinuierliche Abstimmung (*Reporting*) zwischen dem Synchronisationsmanagement und der eigentlichen Maßnahme, um auf Veränderungen in der Projektdurchführung zu reagieren. Des Weiteren führt das Fortschreiten des Projektes zu Veränderungen der Unternehmensarchitektur, die ebenso vom Management der Unternehmensarchitektur aufgenommen und verarbeitet werden.

Vor dem Abschluss einer Maßnahme werden die Veränderungen derselben, die in der Plan-Landschaft dokumentiert sind, in die Ist-Landschaft übernommen und die Ergebnisse in den Betrieb überführt (*Inbetriebnahme & Maßnahme abschließen*).

Das Zusammenspiel dieser Managementprozesse resultiert in einem integrierten Managementprozess für Anwendungslandschaften, der das Management der Unternehmensarchitektur, mit dem IT-Projektlebenszyklus, dem Anforderungsmanagement, dem Strategie- & Zielemanagement, dem Projektportfoliomanagement und dem Synchronisationsmanagement verbindet.

2.1.9. Rolle eines Informationsmodells

Eine Plattform, deren Prozesse zum Management von Anwendungslandschaften in den Abschnitten 2.1.2 bis 2.1.7 beschrieben wurden, verwendet zahlreiche Informationen, die in den verschiedenen Managementprozessen entstehen und verarbeitet werden. Da es sich um eine integrierte Plattform handelt, die ein Zusammenspiel der Prozesse, wie in Abschnitt 2.1.8 dargestellt, bedingt, muss beispielsweise mittels referentieller Integrität sichergestellt werden, dass der Prozess auf konsistenten Daten arbeitet.

Das Informationsmodell soll diese referentielle Integrität sicherstellen und definiert auf einer konzeptuellen und strukturellen Ebene die *Elemente* des Modells, wobei der Begriff *Element* entsprechend der Meta Object Facility (MOF) [OM06d] verwendet wird und der Name der abstrakten Superklasse aller Modellelemente ist, welches beispielsweise Klassen, Attribute und Assoziationen einschließt. Die Beschreibung des Informationsmodells[9] kann beispielsweise mittels UML-Klassendiagrammen und einer zusätzlichen Beschreibung der Elemente in Form eines Glossars, das in natürlicher Sprache erstellt ist, erfolgen.

Für eine derartige Plattform wurde beispielsweise von der BMW Group ein Informationsmodell erstellt [Ha04], welches mittels objektorientierter Methoden mit UML-Klassendiagrammen dokumentiert wurde. Vergleichbare Modelle wurden ebenso von der HVB Systems[10] [Br05a], Krones [La05b] und Siemens [Si05] entwickelt, wobei die Zielsetzungen der einzelnen Modelle variieren (siehe Abschnitt 4.2.2).

Die Vielzahl von Modellen bei Anwendern, die durch weitere Modelle von Werkzeugherstellern (siehe Abschnitt 5.2), Beratungshäuser etc. ergänzt werden, deutet bereits daraufhin, dass derzeit kein allgemeingültiger Standard für ein derartiges Modell existiert, der alle Anforderungen abdeckt. Problematisch ist insbesondere der richtige Abstraktionsgrad, um zu gewährleisten, dass das Modell zwar alle Anforderungen abdeckt, dennoch nicht zu komplex und somit unverständlich sowie schwer wartbar wird. Als Beleg für diese These sei an dieser Stelle ein Zitat von Jaakko Riihinen aus einer E-Mail an einen Werkzeughersteller angeführt [Ri05]:

> [...] we have tried out several tools in this area[11], without much success. It seems to me the tool vendors do not understand the difference between modelling and drawing, which I think is of fundamental importance. Their metamodels are rather complex, but not integrated within themselves.

[9] Für eine weiterführende Diskussion zu Informationsmodellen und deren Definition wird auf Abschnitt 4.2 verwiesen.
[10] Die HVB Systems und die HVB Info wurden in 2006 zur HVB Information Services verschmolzen.
[11] Mit *area* ist hier der Bereich des Managements der Unternehmensarchitektur gemeint.

Aus obigem Zitat ist ebenso zu entnehmen, dass alleiniges *Malen* von Unternehmensarchitektur nicht zielführend ist, sondern Modellierungssprachen verwendet werden müssen. Diese müssen sowohl ein geeignetes Metamodell – das Informationsmodell – zur Verfügung stellen bzw. dessen Adaptierbarkeit ermöglichen, als auch eine Konsistenz zwischen den gezeichneten Modellen – dem *Gemalten* – und den dem Modell zugrundeliegenden Informationen gewährleisten.

Im Datenbank- und Software-Engineering sind derartige Werkzeuge bereits im Einsatz und finden sich bei der Datenmodellierung oder der Softwaremodellierung. Da sich jedoch insbesondere die funktionalen Anforderungen von Datenbank- und Software-Engineering-Werkzeugen deutlich von Werkzeugen für obige Problemstellung unterscheiden, ist ein Rückgriff auf diese Werkzeuge nur bedingt möglich (siehe Abschnitt 5.2).

2.2. Bezug zur IT-Governance

Nach der Einführung der Prozesse zum Management von Anwendungslandschaften in Abschnitt 2.1 werden im Folgenden die Begriffe und Aufgabenbereiche von IT-Governance, IT-Management, Management der Unternehmensarchitektur und Management der Anwendungslandschaft gegeneinander abgegrenzt, um abschließend eine Einordnung der Softwarekartographie zu ermöglichen.

Macharzina und Wolf [MW05] definieren die *Corporate Governance* als: „rechtlichen und faktischen Ordnungsrahmen für die Leitung und Überwachung eines Unternehmens“. Wird der Blickwinkel auf die Informationstechnologie konzentriert, so ist diese Definition übertragbar, denn auch hier existieren rechtliche Regularien, wie das Bundesdatenschutzgesetz, Basel II, Solvency II, SOX etc., die auf die Informationstechnologie einwirken und den Ordnungsrahmen bilden. Der faktische Ordnungsrahmen für die Leitung und Überwachung schlägt sich in der IT-Aufbauorganisation und der IT-Ablauforganisation nieder und definiert Rollen, Zuständigkeiten, Verantwortlichkeiten, und Abläufe.

Eigene Begriffsdefinitionen für *IT-Governance* finden sich beispielsweise bei Weill und Ross [WR04]

> IT governance: Specifying the decision rights and accountability framework to encourage desirable behavior in the use of IT.

und dem IT Governance Institute [IT05]

> IT governance is the responsibility of executives and the board of directors, and consists of the leadership, organisational structures and processes that ensure that the enterprise's IT sustains and extends the organisation's strategies and objectives.

Weill und Ross fokussieren sich in ihrer Definition insbesondere auf die Verteilung von Rollen, Rechten und Zuständigkeiten hinsichtlich der IT im Unternehmen, die derart verteilt werden müssen, dass das *gewünschte* Verhalten der IT erreicht wird. Das IT Governance Institut, mit dem Hintergrund der Auditierung[12], greift ebenso die Nachhaltigkeit der IT auf, ordnet die Zuständigkeit der IT-Governance in den höheren Managementebenen ein und stellt die IT-Governance deutlich in Bezug zu den Unternehmensstrategien und -zielen.

Beiden Definition gemein ist, dass die IT-Governance das IT-Management, also das Führen der IT, beeinflusst, indem der Ordnungsrahmen zur Leitung und Überwachung (vgl. Macharzina und Wolf) für die IT vorgegeben wird.

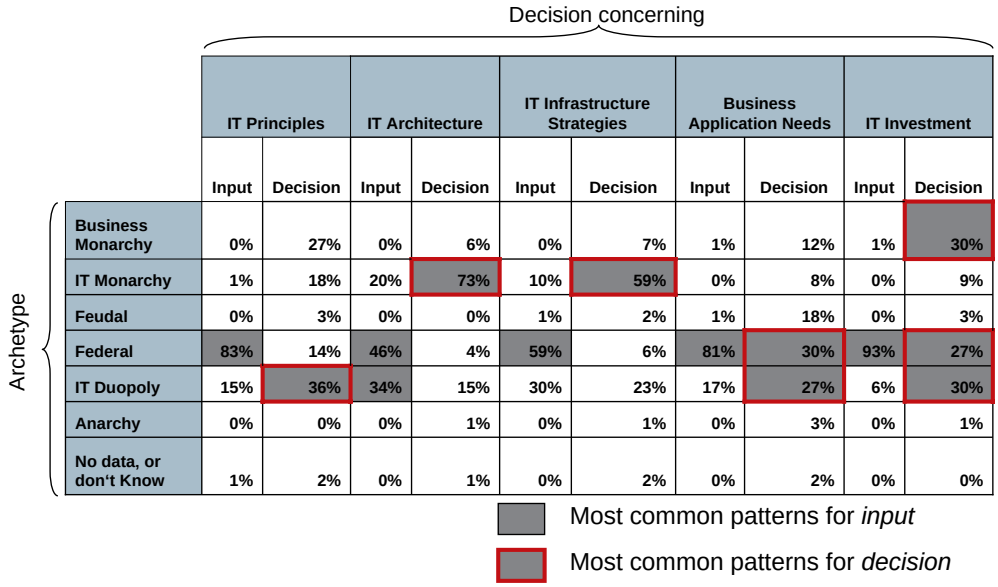


Abbildung 2.8.: Varianten der IT-Governance nach Weill und Ross [WR04]

In [WR04] identifizieren Weill und Ross des Weiteren verschiedene Entscheidungsfelder für die IT (z. B. IT-Prinzipien, IT-Architektur etc.) und stellen diese den ebenso identifizierten Arten von Entscheidungsgremien[13] (z. B. feudal, föderal) gegenüber (siehe Abbildung 2.8). Aus der Darstellung wird ersichtlich, dass Unternehmen die IT auf unterschiedliche Art und Weise steuern, wodurch an eine IT-Governance/Management-Plattform die Anforderung entsteht, hinsichtlich der Informationseingabe für Entscheidungen und dem tatsächlichen Entscheidungsträger flexibel zu sein. Eben dieser Anforderung kommt die in Abschnitt 2.1 beschriebene Plattform nach, indem zwar die Aufgaben im Sinne der Funktionalitäten der Prozesse definiert sind, jedoch die einzelnen Rollen und Rechte konfigurierbar bleiben.

Exemplarisch bedeutet dies, dass das Entscheidungsrecht, ein Projektportfolio zu genehmigen (siehe Abschnitt 2.1.3), von der IT-Governance definiert wird und das tatsächliche

[12] Das IT Governance Institute ist von der ISACA (Information Systems Audit and Control Association) gegründet worden.

[13] Bei Weill und Ross *Archetypes* genannt.

Treffen der Entscheidung die Aufgabe des IT-Managements im Prozess Projektportfoliomanagement ist.

Die Dokumentation der Unternehmensarchitektur dient dem Management als Unterstützungsfunktion für Entscheidungen, indem bestimmte Interessen, die für die Entscheidungsfindung relevant werden, von der Unternehmensarchitektur adressiert werden. Ross definiert die Unternehmensarchitektur als:

> organizing logic for business processes and IT infrastructure, reflecting the integration and standardization requirements of the firm's operating model. [Ro06]

Die Definition von Ross abstrahiert dabei von der konkreten Implementierung und Umsetzung einzelner Elemente, indem sich die Unternehmensarchitektur auf die *organisierende Logik* konzentriert und hierbei eine Reflexion des operationalen Modelle vornimmt.

Eine andere Definition für den Begriff der Unternehmensarchitektur findet sich bei Lankhorst, der sich in seiner Definition insbesondere auf die Prinzipien, Methoden und Modelle konzentriert:

> Enterprise architecture (EA) is a coherent whole of principles, methods and models that are used in the design and realisation of an enterprises organizational structure, business processes, information systems, and infrastructure. [La05a]

Da in der vorliegenden Arbeit die Unternehmensarchitektur nicht nur als statische Abbildung der Unternehmung verstanden werden soll, sondern als Managementwerkzeug, sind die Definitionen um planerische, respektive dynamische Aspekte zu erweitern. Dies führt, ergänzt um messbare Aspekte, zu der Definition aus Kapitel 1, welche die Dokumentation der Unternehmensarchitektur als modellhafte, kohärente und ganzheitliche Abbildung des Unternehmens versteht. Demzufolge wird das Management der Unternehmensarchitektur wie folgt definiert:

Management der Unternehmensarchitektur : Das Management der Unternehmensarchitektur ist ein kontinuierlicher und iterativer Prozess, der die existierende und geplante Unterstützung durch die IT und mit ihr verbundener Elemente im Unternehmen kontrolliert und verbessert. Der Prozess berücksichtigt hierbei die gesamte Unternehmensarchitektur und somit nicht nur die Informationstechnologie des Unternehmens, sondern ebenso die Geschäftsprozesse, Ziele, Strategien etc., um eine ganzheitliche und integrierte Sicht zu erhalten.

Ziel des Managements der Unternehmensarchitektur ist eine von Geschäft und IT gemeinsam entwickelte Sicht, die verwendet wird, um die gegenseitige Ausrichtung von Geschäft und IT zu optimieren.

Das Management der Unternehmensarchitektur ist ein wesentlicher Teil des Managements der Anwendungslandschaft, welche als Gesamtheit der betrieblichen Anwendungssysteme inklusive der Verbindungen zwischen Anwendungssystemen (siehe Kapitel 1)

verstanden wird. Die META Group [ME02] verwendet hierzu den Begriff des *Enterprise Application Portfolios*, welches als „collection of integrated information systems required to satisfy business needs“ definiert wird und führt hierzu weiter aus, dass nicht nur das existierende Portfolio sondern ebenso das geplante Portfolio und ein Migrationsplan Teil des *Enterprise Application Portfolios* sind.

Da jedoch das Management der Anwendungslandschaft nicht autark stattfindet, sondern im engen Bezug zu den anderen Elementen der Unternehmensarchitektur steht, verwenden die Modelle und Methoden zum Management der Anwendungslandschaft nicht nur Elemente der Anwendungslandschaft, sondern setzen die betrieblichen Anwendungssysteme beispielsweise in Bezug zu Geschäftsprozessen, Strategien oder Infrastrukturelementen. Eben diese Modelle und Methoden sind die von der Softwarekartographie zu entwickelnden bzw. entwickelten, um zur Beschreibung, Bewertung und Gestaltung von Anwendungslandschaften beizutragen.

Bevor in Kapitel 3 und 4 die Modelle und Modellierungsmethoden der Softwarekartographie beschrieben werden, soll im Folgenden ein Einblick in verwandte Ansätze zum Management von Unternehmensarchitekturen gegeben sowie Frameworks für Unternehmensarchitekturen vorgestellt werden. Das Thema Anwendung von Softwarekarten wird eigenständig in Kapitel 5 behandelt.

2.3. Analyse und Vergleich existierender Modelle und Frameworks

Für die in Abschnitt 2.1 vorgestellte Plattform mit ihren Prozessen zum Management von Anwendungslandschaften existieren zwei vergleichbare Ansätze, die im Folgenden vorgestellt werden sollen. Im einzelnen sind dies die Ansätze von Keller [Ke06] (siehe Abschnitt 2.3.1) und der alfabet AG [al06] (siehe Abschnitt 2.3.2), die nach einer Einführung mit dem Ansatz in Abschnitt 2.1 in Beziehung gesetzt werden. Den Ansätzen ist gemein, dass sie prozessorientiert sind und nicht nur einzelne Modelle für Unternehmensarchitekturen oder Informationssysteme behandeln.

Die Abschnitte 2.3.3 und 2.3.4 stellen je ein Framework für Unternehmensarchitekturen bzw. für die Architektur integrierter Informationssysteme vor, um den Modellen zur Beschreibung von Unternehmensarchitekturen bzw. Informationssystemen Rechnung zu tragen.

Neben den im Folgenden vorgestellten Ansätzen existiert eine Vielzahl weiterer Ansätze, z. B.

- das *The Open Group Architecture Framework* (TOGAF) [TOG02],
- die Modellierungssprache *ArchiMate* [La04],
- der Ansatz zur Verknüpfung von IT-Governance und Unternehmensarchitekturen von Niemann [Ni05],
- das *Department of Defense Architecture Framework* (DODAF) [Do04],
- das *Federal Enterprise Architecture Framework* (FEAF) [Of06] etc.

Die Auswahl der vorgestellten Ansätze beruht auf der Historie vieler Frameworks, die mit dem Zachman Framework begonnen hat, der Prominenz im deutschsprachigen Raum (ARIS) und der Vergleichbarkeit mit dem Ansatz aus Abschnitt 2.1 (Keller und alfabet AG). In späteren Abschnitten werden weitere Ansätze, wenn diese relevant sind, in den Kontext dieser Arbeit gestellt.

2.3.1. Prozesse der IT-Unternehmensarchitektur nach Keller

Keller beschreibt in [Ke06] sechs Prozesse für die IT-Unternehmensarchitektur, die der Abbildung 2.9 zu entnehmen sind. Keller definiert hierbei den Begriff der IT-Unternehmensarchitektur wie folgt:

> IT-Unternehmensarchitektur ist derjenige Teil der Unternehmensarchitektur (Enterprise Architecture), den die IT-Funktion in einem Unternehmen machen darf, ohne wegen Kompetenzüberschreitung von anderen Unternehmenseinheiten außerhalb der IT erfolgreich politisch attackiert zu werden. Im besten Fall sind IT-Unternehmensarchitektur und Unternehmensarchitektur identisch und einheitlich organisiert.

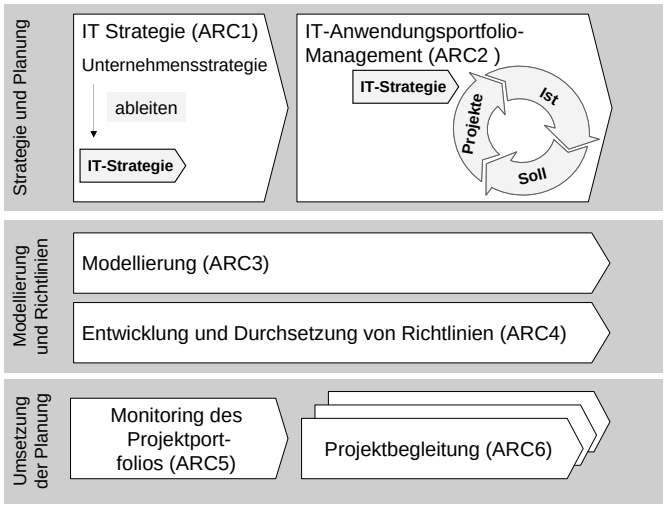


Abbildung 2.9.: Prozesse für die IT-Unternehmensarchitektur nach Keller [Ke06]

Ergänzt wird diese Definition von Keller durch: „In der Praxis wird die IT-Unternehmensarchitektur alle Gebiete der Unternehmensarchitektur abdecken bis auf die Definition der Geschäftsprozesse und die Aufbauorganisation der nicht-IT-Bereiche.“ Die obige Definition, von Keller selbst als pragmatisch bezeichnet [Ke06, S. 43], ist insbesondere im Kontext der Aufgaben eines IT-Unternehmensarchitekten entstanden.

Im Folgenden werden die Prozesse aus Abbildung 2.9 dem Ansatz aus Abschnitt 2.1 gegenübergestellt:

- Der Prozess *IT-Strategie* aus Abbildung 2.9 beschäftigt sich mit der Ableitung einer IT-Strategie aus der Unternehmensstrategie und ist nur bedingt mit dem Prozess

aus Abschnitt 2.1.4 vergleichbar. Der Prozess *Strategie- & Ziele-Management* aus Abschnitt 2.1.4 fokussiert auf die Dokumentation und Überwachung von Strategien, enthält jedoch keine Hinweise, wie aus der Unternehmensstrategie die IT-Strategie abzuleiten ist.

- Der Prozess *IT-Anwendungsportfoliomanagement* ist vergleichbar mit dem *Anforderungsmanagement* und dem *Projektportfoliomanagement* (siehe Abschnitte 2.1.2 und 2.1.3), wobei Keller in diesem Prozess bereits wiederkehrende Tätigkeiten definiert, die eine Bewertung des Portfolios der Anwendungssysteme erfordern. Beiden Ansätzen gemein ist, dass eine Analyse des Portfolios in einem Gesamtkontext über alle Anwendungssysteme bzw. Anforderungen oder Projektanträge erfolgen muss.
- Der Prozess *Modellierung* in Abbildung 2.9 ist bei Keller analog zu dem Prozess *Management der Unternehmensarchitektur*, wobei Keller hierzu ebenso die Softwarekarten vorschlägt, die Teil des Abschnitts 3.4.1 sind [LMW06]. Beide Ansätze verwenden globale Sichten auf die Unternehmensarchitektur und die Anwendungslandschaft, um ein konsistentes Gesamtbild zu erreichen.
- Der Prozess *Entwicklung und Durchsetzung von Richtlinien* ist nicht explizit in dem Ansatz aus Abschnitt 2.1 enthalten. Die *Richtlinien* nach Keller umfassen nicht einzelne Bereiche, wie beispielsweise IT-Architekturen (siehe Abschnitt 2.1.5), sondern beziehen sich auf allgemeine die IT betreffende Richtlinien. Im Vordergrund dieses Prozesses steht das Erkennen von Regelungsbedarf, das Entwerfen, Abstimmen, Kommunizieren und Durchsetzen von Richtlinien, welches bezüglich der Unternehmensarchitektur mittels einer Soll-Landschaft ermöglicht und bezüglich IT-Architekturen durch Musterarchitekturen und Musterlösungen abgebildet wird. Ein eigenständiger Prozess zur *Entwicklung und Durchsetzung von Richtlinien* kann nur dann Teil der in Abschnitt 2.1 dargestellten Plattform sein, wenn Elemente für Richtlinien definiert werden und die Überwachung auf Ebene der Soll-Landschaft, der Musterarchitekturen und der Musterlösungen erfolgt.
- Der Prozess *Monitoring des Projektportfolios*, welcher von Keller zur Kontrolle der Strategiekonformität, zur Vermeidung von funktionaler Redundanz etc. beschrieben wird, findet sich in allen Managementprozessen der Plattform aus Abschnitt 2.1 wieder. Die Strategiekonformität wird beispielsweise durch das Verknüpfen von Strategien mit Anforderungen und Projektvorschlägen ermöglicht und das Entstehen von funktionaler Redundanz durch den Einsatz von Softwarekarten überwacht.
- Der Prozess *Projektbegleitung* nach Keller wird von der Plattform aus Abschnitt 2.1 nicht berücksichtigt, da die *Projektbegleitung* nach Keller keine direkte Aufgabe einer Plattform ist, sondern die von IT-Unternehmensarchitekten, die in Projekten die Konformität mit der IT-Unternehmensarchitektur sicherstellen und zusätzlich Projekte bei der Umsetzung mit Blick auf die IT-Unternehmensarchitektur unterstützen. Durch die Konfiguration des Evaluierungsprozesses für Anforderungen und Projektvorschläge ist es jedoch möglich, die Freigabe ebenso an eine Zustimmung eines IT-Unternehmensarchitekten zu binden.

Zusammenfassend ergänzt Keller die Funktionen der in Abschnitt 2.1 vorgeschlagenen Plattform um eine Ausführungssicht, die Hinweise gibt und Vorgaben macht, wie die

Plattform zu bedienen ist. Die Funktionen der einzelnen Elemente sind vergleichbar, wobei sich die vorgestellte Plattform auf die Integration (existierender) Managementprozesse fokussiert. Ein wesentlicher Unterschied der beiden Ansätze ist, dass der Ansatz aus Abschnitt 2.1 das *Management der Unternehmensarchitektur* als Lieferant zu anderen Managementprozessen versteht und sich existierender Managementprozesse (z. B. Anforderungsmanagement, IT-Architekturmanagement) bedient.

2.3.2. Module von planningIT nach alfabet

Das Werkzeug planningIT der alfabet AG ist ein integriertes Planungswerkzeug zum IT-Management, dessen Zielsetzung es ist, den Prozess von der Anforderung bis zur Budgetierung zu unterstützen. Das Werkzeug baut auf einem Repository inklusive einem Informationsmodell (*Logical Inventory* nach alfabet) auf und stellt hierauf verschiedene Funktionalitäten zur Verfügung, die sich in sieben Module unterteilen (siehe Abbildung 2.10). Die Funktionen der einzelnen Module werden wie folgt beschrieben [al06]:

- Das *Business Demand Management Module* ermöglicht es, Ziele und Soll-Landschaften an den Anforderungen des Geschäfts auszurichten. Neue Anforderungen werden dokumentiert und von Analysten konsolidiert. Entscheidungsträger können Anforderungen ablehnen und hinsichtlich des Erfüllungsgrads und des Fortschritts begutachten.
- Das *Application Architecture Management Module* wird verwendet, um eine Anforderung mit Informationen zu der geplanten Umsetzung zu ergänzen. Veränderungen in der Architektur werden wie Kosten hinterlegt, so dass detaillierte Informationen zu einer Anforderung in diesem Modul abgebildet werden.
- Das *Enterprise Architecture Management Module* wird genutzt, um Standards und Richtlinien zu definieren und die Standardkonformität sicherzustellen.

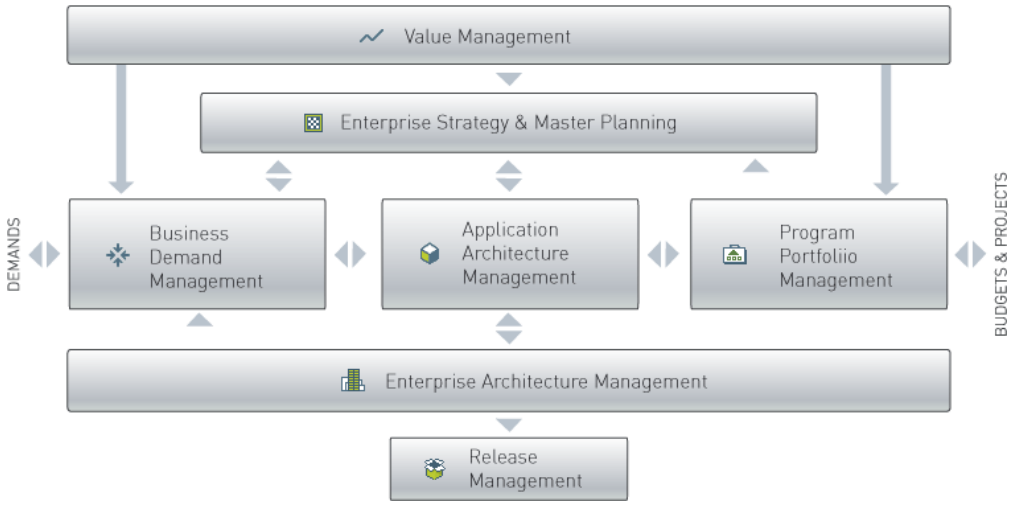


Abbildung 2.10.: Module von planningIT, alfabet AG [al06]

- Das *Enterprise Strategy* und *Master Planning Module* wird verwendet, um das Applikationsportfolio zu analysieren und zu bewerten (Redundanzen, Kosten etc.). Dieses Modul wird ebenso verwendet, um die Zielarchitektur zu konsolidieren und den IT-Masterplan[14] zu erstellen.
- Das *Program Portfolio Management Module* erlaubt es, Projektanträge zu priorisieren und Budgets, auf der Basis von Strategie- und Zielbewertungen, Ressourcenpriorisierung und Risiken, zu verteilen. Ergebnisse und erstellte Berichte werde von Entscheidungsträgern, Steuerungskomitees, Projektmanagern etc. begutachtet.
- Das *Value Management Module* ermöglicht die gegenseitige Ausrichtung von Geschäft und IT anhand von Strategien, welche die Zielarchitektur dirigieren. Ebenso werden Performanzmessungen hinsichtlich Zielen und Schlüsselindikatoren ermöglicht.
- Das *Release Management Module* wird verwendet, um operative Pläne aus den strategischen Plänen abzuleiten und die technische Ausrichtung zu verifizieren. Ebenso wird das Prüfen der Standardkonformität zukünftiger Versionen auf der physischen Infrastrukturebene ermöglicht.

Die Funktionalitäten der Module von planningIT besitzen deutliche Überschneidungen mit den Funktionalitäten der Plattform und den Prozessen zum Management der Anwendungslandschaft aus Abschnitt 2.1. Das planningIT-Modul *Business Demand Management* entspricht hinsichtlich der Funktionalität dem Prozess *Anforderungsmanagement* aus Abschnitt 2.1.2 und das planningIT-Modul *Program Portfolio Management Module* dem Prozess *Portfoliomanagement* aus Abschnitt 2.1.3.

Hinsichtlich der anderen Module gibt es keine eindeutige Deckungsgleichheit sondern Überschneidungen hinsichtlich der Funktionalität:

- Die Funktionalitäten des Prozesses *Strategie- & Zielemanagement* finden sich sowohl in dem planningIT-Modul *Enterprise Strategy* als auch in dem Modul *Value Management*. Das Verknüpfen von Strategien mit Anforderungen, Projektvorschlägen und Elementen der Unternehmensarchitektur wird von diesem Modul genauso wie die Bewertung und Zielüberwachung zur Verfügung gestellt.
- Eine vergleichbare Funktionalität, wie die des planningIT-Moduls *Release Management*, ist in der Plattform aus Abschnitt 2.1 in dem Prozess *Management der Unternehmensarchitektur* vorgesehen, welches neben anderen Ebenen auch die Infrastruktur betrachtet.
- Die Funktionalitäten von planningIT hinsichtlich der Module *Enterprise Architecture Management* und *Master Planning* sind in den Prozessen aus Abschnitt 2.1 anders gruppiert. Die Richtlinien auf der Ebene einzelner betrieblicher Anwendungssysteme und Infrastrukturelementen werden durch den Prozess *IT-Architekturmanagement* mittels Musterarchitekturen und -lösungen ermöglicht. Wird eine ganz-

[14] Im Sprachgebrauch von planningIT ist ein IT-Masterplan ein Modell, das die Gesamtheit der geplanten Geschäftsunterstützung durch die IT abbildet. Der IT-Masterplan besteht aus IT-Masterplankarten, die eine Notation analog zu der Softwarekarte aus Abbildung 2.5 verwenden.

heitliche Sicht verlangt, wie sie beispielsweise bei den IT-Masterplänen erfolgt, so sind diese Funktionalitäten im Prozess zum *Management der Unternehmensarchitektur* zu finden.

Abschließend ist festzuhalten, dass die Funktionalität von planningIT eng verwandt mit den Prozessen der Plattform aus Abschnitt 2.1 ist. In planningIT Version 1.0 nicht enthalten ist die Funktionalität des Prozesses *Synchronisationsmanagements* (siehe Abschnitt 2.1.6), da sich planningIT in dieser Version auf den Prozess von der Anforderung bis zur erstmaligen Budgetierung konzentriert. In planningIT Version 2.1[15] sind jedoch in diesem Bereich neue Funktionalitäten integriert, die auch Veränderungen während der Projektlaufzeit berücksichtigen und sich ggf. auch in Veränderungen der Budgetierung von Projekten widerspiegeln.

2.3.3. Enterprise Architecture Framework nach Zachman

Das Zachman Framework gilt als Vater vieler Frameworks für Unternehmensarchitekturen und wurde in seiner ursprünglichen Form 1987 als ein Framework für Architekturen von Informationssystemen vorgestellt [Za87]. Ausgangspunkt für Zachman ist die steigende Komplexität von Informationssystemen, die er nutzt, um verschiedene Sichten auf ein System zu motivieren, wobei die von ihm verwendete Analogie gleichartige Vorgehensweisen beim Bau von Häusern und auch Flugzeugen beschreibt. 1987 leitet Zachman hierzu ein Framework in Form einer Matrix, bestehend aus fünf Zeilen und drei Spalten, ab. Die Zeilen entsprechen den Sichten für verschiedene Interessenten an dem System und die Spalten definieren unterschiedliche Abstraktionsgrade, auf denen nur bestimmte Eigenschaften des Systems betrachtet werden. Die Matrix des Frameworks in der Version von 1987 besteht aus den Sichten (Zeilen)

- *Scope Description*,
- *Model of the Business*,
- *Model of the Information System*,
- *Technology Model* und
- *Detailed Description*

sowie den Abstraktionsgraden (Spalten)

- *Data Description (What?)*,
- *Process Description (How?)* und
- *Network Description (Where?)*.

Die Fragestellungen *Who?*, *When?* und *Why?* werden von Zachman in [Za87] zwar angesprochen, jedoch erst in der von Sowa und Zachman in 1992 publizierten Version [SZ92] in das Framework aufgenommen. Neben der Aufnahme dieser neuen Fragestellungen, die in einer Matrix bestehend aus fünf Zeilen und sechs Spalten mündet (siehe Abbildung 2.11),

[15] planningIT Version 2.1 wurde im Oktober 2006 veröffentlicht.

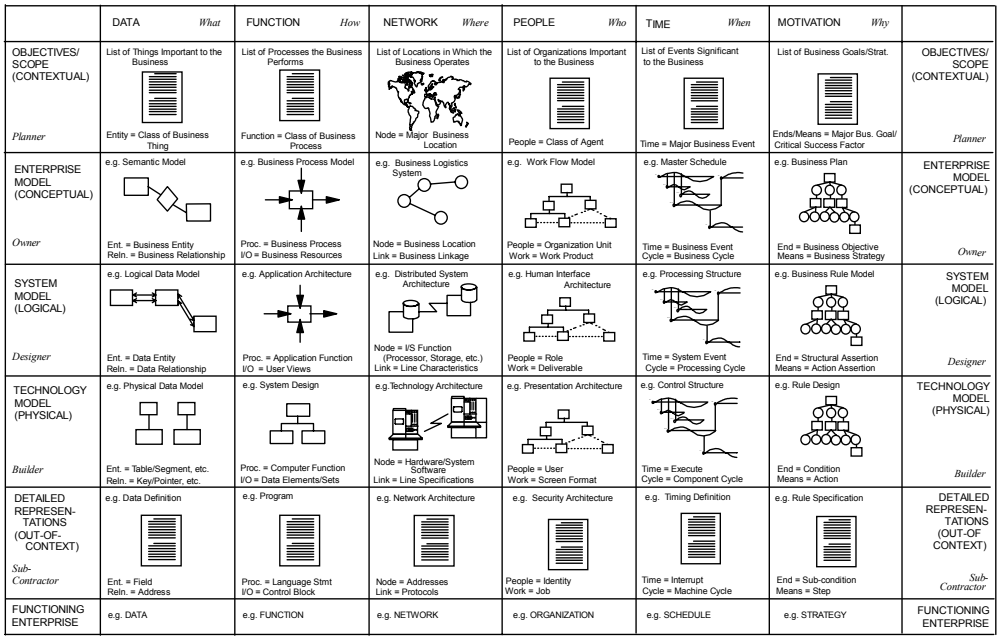


Abbildung 2.11.: Zachman-Framework: „A Framework for Enterprise Architecture“ [Za06]

ergänzen Sowa und Zachman zum einen Regeln und zum anderen ein Datenmodell für Teile des Frameworks, wobei die einzelnen Entitäten weder informell beschrieben noch formal definiert werden.

Die Regeln definieren Eigenschaften der einzelnen Zellen des Frameworks, beispielsweise dass die Spalten keiner Ordnung unterliegen oder jede Zelle einzigartig ist. Das Datenmodell, welches Sowa und Zachman in Form einer E/R-diagrammartigen Notation beschreiben, definiert für die einzelnen Zellen die relevanten Entitäten. Die Verbindungen zwischen Zellen in der horizontalen Ebene modelliert Zachman durch ein durchgehendes E/R-Modell, während die Verbindung zwischen den Zeilen durch Namensgleichheit von Instanzen (nicht Entitäten) informell beschrieben wird. Als Modellierungssprache auf Instanzebene schlagen Sowa und Zachman Konzeptgraphen vor, die für das Datenbankdesign [Za76] entwickelt und mittels einer Prädikatenlogik formalisiert wurden.

Zachman bezeichnet das von Sowa und ihm entwickelte Framework inzwischen als ein Framework für Unternehmensarchitekturen (siehe Abbildung 2.11), wobei das Informationsmodell der ursprünglichen Variante [SZ92] nicht angepasst wurde. Dies bedingt, dass das bestehende Informationsmodell aus der Arbeit von Sowa und Zachman [SZ92] nicht für eine Unternehmensarchitektur verwendet werden kann, da dieses zu feingranular ist. Sowa und Zachman erlauben es in ihrem Modell beispielsweise nicht, dass ein Geschäftsprozess von mehreren Anwendungssystemen unterstützt wird, da sich das ursprüngliche Modell auf einzelne Informationssysteme konzentriert[16].

[16] Ein Anwendungssystem ist Teil eines Informationssystems (siehe Abschnitt 1.3).

In den Publikationen [Za96] und [Za06] stellt Zachman sein Framework (siehe Abbildung 2.11) als generischen Ansatz zur Analyse und Beschreibung von Systemen dar. Die einzelnen Fragestellungen (Spalten) und Perspektiven (Zeilen) des Frameworks sollen als generischer Leitfaden dienen, um für die Interessenten des System die Fragestellungen zu beantworten, wobei ein System sowohl ein physischer Gegenstand, wie ein Flugzeug, als auch ein virtuelles System, wie ein Unternehmen, sein kann.

Festzuhalten ist, dass das Zachman-Framework im Kontext von Unternehmensarchitekturen ein Framework ohne mögliche Instantiierung ist und nur eingeschränkt Vorgaben hinsichtlich der Modellierungssprachen und zu verwendenden Methoden macht.

Eine direkte Beziehung zwischen den in Abschnitt 2.1 vorgestellten Prozessen und dem Framework von Zachman ist nicht vorhanden. Nichtsdestotrotz können die einzelnen Modelle, die in den unterschiedlichen Prozessen der Plattform Verwendung finden, mit dem Framework von Zachman in Beziehung gesetzt werden, indem diese mit den Zellen des Frameworks assoziiert werden. Da das Framework von Zachman in der Version aus [Za96] und [Za06] keine detaillierten Vorgaben macht, wie die einzelnen Modelle, die sich hinter den Zellen des Frameworks verbergen, auszusehen haben und welche Modellierungssprachen zu verwenden sind, ist diese Verbindung einfach zu realisieren. Der Mehrwert hierbei ist jedoch fraglich.

Wird die Version des Frameworks von Zachman aus [SZ92] herangezogen, so bleibt die Kritik, dass das Datenmodell zu feingranular ist und eine Erklärung der Entitäten fehlt. Da Sowa und Zachman [SZ92] des Weiteren nur die Spalten *Data*, *Process* (in Abbildung 2.11 *Function*) und *Network* mit einem Datenmodell versehen, ist ein Vergleich der Prozesse aus Abschnitt 2.1 und dem Framework von Zachman auf Basis eines Datenmodells nicht möglich.

2.3.4. Architektur integrierter Informationssysteme nach Scheer

Die Architektur integrierter Informationssysteme (ARIS) stellt Methoden und Modelle zur Beschreibung und Entwicklung integrierter, betriebswirtschaftlicher Informationssysteme bereit. Das ARIS-Haus, welches von Scheer als Bezugsrahmen zur Geschäftsprozessbeschreibung definiert wird [Sc01, S. 1], ist in Abbildung 2.12 dargestellt. Dies ist gleichfalls die Basis für das ARIS-Toolset der IDS Scheer AG. Neben dem ARIS-Haus wurden weitere Konzepte und Ansätze im Kontext von ARIS entwickelt, beispielsweise das ARIS - House of Business Engineering [Sc96], die im Folgenden nicht weiter betrachtet werden.

Das ARIS-Haus in Abbildung 2.12 trennt zwischen den Sichten Funktion, Organisation, Daten, Leistung und Steuerung. Es enthält ebenso ein Phasenmodell innerhalb der einzelnen Sichten, um den Weg von der strategischen Geschäftsprozessanalyse und der Sollkonzeption zur Umsetzung in die Informations- und Kommunikationstechnik darzustellen.

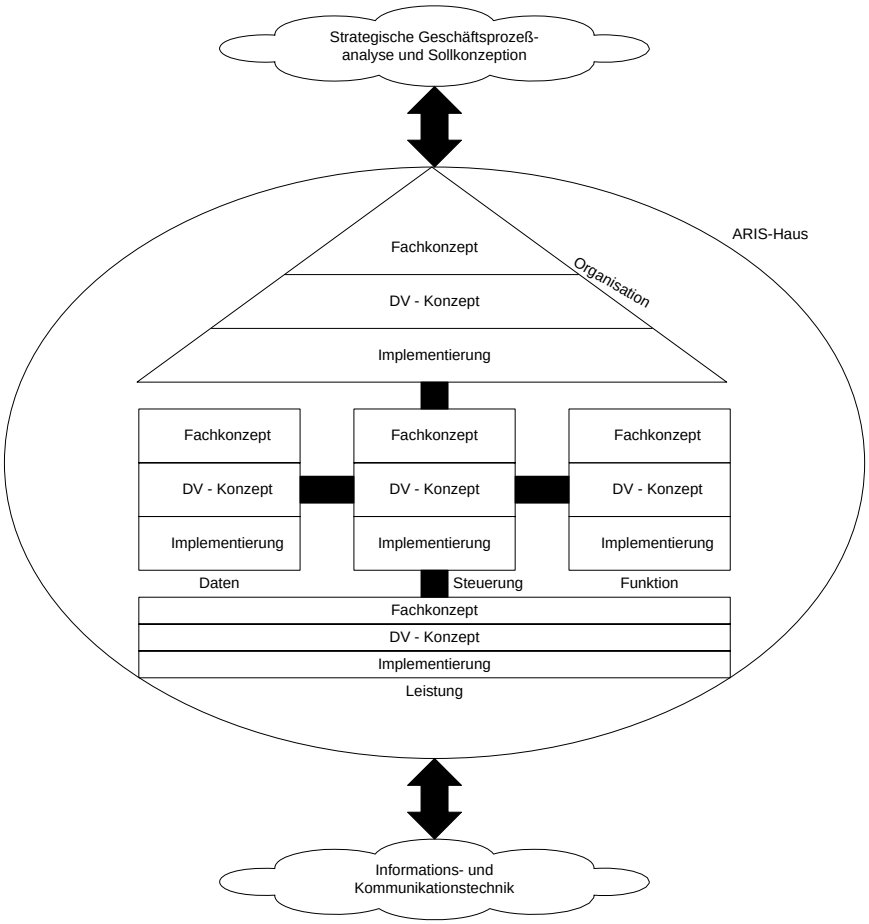


Abbildung 2.12.: ARIS-Haus mit Phasenkonzept nach Scheer [Sc02]

Die einzelnen Sichten werden von Scheer wie folgt definiert [Sc02]:

- Die *Funktionssicht* fasst die Vorgänge, welche Input-Leistungen zu Output-Leistungen transformieren, zusammen.
- Die *Organisationssicht* bildet mit der Klasse Organisationseinheiten die Aufbauorganisation ab.
- Die *Datensicht* umfasst die Umfelddaten der Vorgangsbearbeitung sowie die Nachrichten, welche Funktionen auslösen bzw. von Funktionen erzeugt werden.
- Die *Leistungssicht* enthält alle materiellen und immateriellen Input- und Output-Leistungen einschließlich der Geldflüsse.
- Die *Steuerungssicht*, die auch als Prozesssicht bezeichnet wird, behandelt die Beziehungen zwischen den Sichten sowie den Geschäftsprozess in seiner Gesamtheit. Sie bildet den Rahmen für die systematische Betrachtung aller bilateralen Beziehungen der Sichten sowie der vollständigen Prozessbeschreibung.

Die einzelnen Sichten werden von Scheer in [Sc01] um ein Informationsmodell ergänzt, welches für die einzelnen Sichten die Modellierungssprache definiert. Das Informationsmodell wird dabei mittels einer Modellierungssprache, die an UML-Klassendiagramme angelehnt ist, beschrieben und um natürlichsprachliche Beschreibungen ergänzt, wobei die einzelnen Elemente ohne Attribute modelliert sind.

Wird ARIS mit den vorgestellten Prozessen aus Abschnitt 2.1 in Beziehung gesetzt, so sind Gemeinsamkeiten vornehmlich in den Prozessen *Management der Unternehmensarchitektur* (siehe Abschnitt 2.1.7) und *IT-Architekturmanagement* (siehe Abschnitt 2.1.5) zu finden. Mittels ARIS, welches im ARIS-Toolset der IDS Scheer AG implementiert ist, können verschiedenste Elemente der Unternehmensarchitektur modelliert werden, jedoch zeigt die Methode bei problemadäquaten Sichten im Kontext des Managements der Unternehmensarchitektur Schwächen hinsichtlich abstrakteren Sichten und einer Abbildung der Evolution der Anwendungslandschaft (siehe Abschnitt 5.1 und [se05a]). Des Weiteren fokussiert das Management der IT-Architektur auf die Architektur einzelner Informationssysteme, dies jedoch nicht aus der Perspektive der ARIS-Sichten. Dies ist darauf zurückzuführen, dass ARIS in erster Linie zur Entwicklung von Informationssystemen konzipiert wurde und nur zweitrangig auf das Management zahlreicher Informationssysteme, die mittels unterschiedlichsten Architekturen realisiert sind, ausgerichtet ist.

Die Funktionen der weiteren Prozesse der Plattform aus Abschnitt 2.1 können teils mit ARIS modelliert werden, jedoch fehlen ARIS hierzu geeignete Sichten, die jedoch in dem ARIS-Toolset ergänzt (siehe Abschnitt 5.2 und [se05a]) wurden. Dies basieren nicht auf der Methode und den Modellen von ARIS. Diese These und die obige Kritik an ARIS wird untermauert durch die Tatsache, dass die IDS Scheer AG in einer neuen Version der ARIS Design Plattform weitere Funktionalitäten anbietet [Ma06b], die einige der angesprochenen Problemstellungen beseitigt.

2.4. Bewertung

Die in Abschnitt 2.1 vorgestellten Prozesse zum Management von Anwendungslandschaften sind ein Ansatz, um verschiedene Managementtätigkeiten mittels mehrerer integrierter Prozesse zu unterstützen, die ebenso in einer Plattform umgesetzt werden können. Die Funktionen der Prozesse integrieren sich dabei in einen IT-Projektlebenszyklus und beziehen nicht nur die Informationstechnologie sondern auch betriebswirtschaftliche Elemente mit ein. Die Funktionen der Prozesse müssen durch Umsetzung der IT-Governance in ein geeignetes Rollen- und Rechtemodell (siehe Abschnitt 2.2) operationalisiert werden, um die Umsetzung und Anwendung der Prozesse in einem Unternehmen zu ermöglichen.

Die in Abschnitt 2.3 vorgestellten Modelle und Frameworks für Unternehmensarchitekturen wurden mit den Prozessen und der Plattform aus Abschnitt 2.1 in Beziehung gesetzt und aufgezeigt, dass existierende Konzepte und Ansätze vereinigt werden und somit das Operationalisieren verschiedener Modelle und Frameworks ermöglicht wird. Insbesondere die großen Überschneidungen mit dem Werkzeug planningIT (siehe Kapitel 2.3.2) zeigen, dass die von den Prozessen und der Plattform definierten Funktionen, auch in einem existierenden Werkzeug umgesetzt werden.

Im Folgenden konzentriert sich diese Arbeit auf den Prozess zum Management der Unternehmensarchitektur aus Abschnitt 2.1. Hierbei liegt der Fokus im Detail auf Modellen und Methoden zur Beschreibung, Bewertung und Gestaltung der Anwendungslandschaft, die in den folgenden Kapiteln erarbeitet werden. Die Funktionen dieses Prozesses der Plattform im Sinne einer Anforderungsanalyse werden jedoch nicht beschrieben.

Der Prozess zum Management der Unternehmensarchitektur nutzt, wie in Abschnitt 2.1.7 beschrieben, graphische Darstellungen der Unternehmensarchitektur, um beispielsweise

- Transparenz der Unternehmensarchitektur und der Anwendungslandschaft zu erhalten,
- die Gestaltung der Anwendungslandschaft mit Bezug zu den betriebswirtschaftlichen Elementen der Unternehmensarchitektur geeignet zu planen,
- die strategische Ausrichtung der Informationstechnologie zu operationalisieren sowie zu kontrollieren und
- auftretende Abhängigkeiten von Projekten, die die Unternehmensarchitektur verändern, zu identifizieren.

Die verwendeten graphischen Darstellungen sind Modelle der Unternehmensarchitektur, um die obigen Tätigkeiten zu unterstützen.

Die Darstellungen, die einen Bezug zu mehreren Anwendungssystemen besitzen und diese geeignet visualisieren, wurden bisher in der Literatur nicht ausreichend analysiert, um als eine Modellierungssprache für Anwendungslandschaften bezeichnet zu werden. Im Folgenden werden diese Modelle, als Softwarekarten bezeichnet, detailliert betrachtet. Es werden Anforderungen an die Darstellungen erhoben (siehe Abschnitt 3), der Aufbau der Modelle objektorientiert beschrieben (siehe Abschnitt 4) und ihre Anwendungen aufgezeigt (siehe Abschnitt 5).

Die in diesem Abschnitt eingeführten Prozesse zum Management der Anwendungslandschaft zeigen, wie die im Folgenden beschriebenen Modelle beim Management von Unternehmensarchitekturen verwendet werden und wie sich der Prozess zum Management von Unternehmensarchitekturen in andere IT-Managementprozesse und ein IT-Projektlebenszyklus integriert, um ein ganzheitliches Management der Anwendungslandschaft zu ermöglichen.

KAPITEL 3

Softwarekarten als Modellierungssprache für Anwendungslandschaften

Inhaltsverzeichnis

Die Visualisierung von Anwendungslandschaften, als Objekte bestehend aus in Software implementierten Anwendungssystemen, legt eine Visualisierung mit Modellen und

Methoden der Informatik oder Wirtschaftsinformatik nahe. In der Informatik und Wirtschaftsinformatik wurden zahlreiche Modellierungssprachen entwickelt, um Systeme hinsichtlich Struktur und Verhalten zu beschreiben, die insbesondere in den unterschiedlichen Phasen der Softwareentwicklung eingesetzt werden. Namhafte Modellierungssprachen in diesem Kontext sind die *Unified Modeling Language* [OM05d] (UML) der *Object Management Group* (OMG), die ereignisgesteuerten Prozessketten [KNS92] von Scheer, das Entity-Relationship-Modell [Ch76] von Chen und die Petrinetze [JV87] von Petri.

Die angeführten Modellierungssprachen verfügen durchweg über graphische Notationen, um eine gegenüber der abstrakten Syntax[1] *einfache* und verständliche Repräsentation des Modells zu ermöglichen. Allen Modellierungssprachen gemein ist, dass sie für eine oder mehrere Problemstellungen konstruiert wurden. Die ereignisgesteuerten Prozessketten wurden beispielsweise entwickelt, um betriebliche Abläufe hinsichtlich des Kontrollflusses zu beschreiben, zu analysieren und zu simulieren, und das Entity-Relationship-Modell, um eine einheitliche Sicht auf Daten zu ermöglichen.

Neben diesem Verständnis für Modellierungssprachen und die mit ihnen erstellten Modelle hat sich im Software-Engineering der Begriff der Architektur, der seinen Ursprung in der Baukunst und im Bauwesen hat, etabliert, um ein System unter bestimmten Kriterien zu analysieren und ebenso bestimmte Anforderungen an das System zu stellen. Eine der frühesten Verwendung des Architekturbegriffs in der Baukunst wird Vitruvius (ca. 30 v. Chr.) zugeschrieben, der von einer guten Architektur die drei Eigenschaften *Firmitas* (Stabilität), *Utilitas* (Zweckdienlichkeit) und *Venustas* (Schönheit) verlangt [Sm03]. Eine allgemeine Definition für den Begriff der Architektur ist in der Encyclopædia Britannica zu finden: „The art and technique of designing and building, as distinguished from the skills associated with construction.“ [En06a].

Die enge Verwandtschaft der Worte *building* und *constructing* sowohl im englischen Sprachgebrauch als auch in einer freien deutschen Übersetzung lassen eine Trennung von Design und Implementierung, wie sie im Software-Engineering üblich ist, nur schwerlich zu. In dieser Arbeit wird der Begriff *Architektur* des IEEE Std 1471-2000 (kurz IEEE 1471), wie bereits in Kapitel 1 eingeführt, verwendet:

> The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. [IE00]

Die Definition des IEEE 1471 konzentriert sich auf *Systeme* und ist ausreichend abstrakt, um sowohl für einzelne Softwaresysteme als auch für Unternehmensarchitekturen dienlich zu sein. Andererseits ist die Definition entsprechend anwendbar für IT-Systeme und legt fest, dass zu einer Architektur die Komponenten, ihre Beziehungen untereinander und zur Umgebung zählen, die den fundamentalen Aufbau des System beeinflussen bzw. beeinflusst haben. Der Zusatz, dass ebenso die Prinzipien, die das Design und die Evolution des Systems bedingen, zu einer Architektur zu zählen sind, ergänzt die Definition der Architektur dahingehend, dass inhärente Elemente des Systems zur Architektur zählen.

[1] Zur Begriffsklärung von abstrakter und konkreter Syntax wird auf Abschnitt 3.1.2 verwiesen.

Für das weitere Verständnis der Arbeit ist von Bedeutung, dass jedes System eine Architektur besitzt und eine Architekturdokumentation eine *erfahrbare* und ebenso *kommunizierbare* Beschreibung der Architektur darstellt. Dies gilt sowohl für eine Softwarearchitektur als auch für eine Unternehmensarchitektur. Dieses Verständnis von Architektur entspricht der Sichtweise des IEEE 1471: „Every system has an architecture, [...]“ [IE00, S. 4].

Der folgende Abschnitt 3.1 führt zunächst in die Grundlagen der Architekturdokumentation ein und baut zugleich das Verständnis von Modellen und Methoden in dieser Arbeit auf. Der sich anschließende Abschnitt 3.2 gibt eine kurze Einführung in die (thematische) Kartographie, um den Begriffsapparat für Karten zu erweitern. Mit den zuvor gelegten Grundlagen der Architekturdokumentation und dem Begriffsapparat der Kartographie werden im Anschluss die Modelle zur Architekturdokumentation von Anwendungslandschaften in der Praxis (siehe Abschnitt 3.3) beschrieben.

In Abschnitt 3.4 werden die eingeführten Modelle von Anwendungslandschaften hinsichtlich ihres Aufbaus kategorisiert. Die entstehenden Softwarekartentypen werden mittels Gestaltungsprinzipien beschrieben und die Rolle von Titeln und Legenden für Softwarekarten hervorgehoben.

3.1. Grundlagen der Architekturdokumentation

Bei der Dokumentation von Architekturen werden oftmals unterschiedliche Modelle eingesetzt, um eine Architektur aus verschiedenen Perspektiven zu beschreiben und auf diese Weise verschiedene Interessen an dem betrachteten System zu adressieren. Im Software-Engineering existieren hierzu verschiedene Ansätze, um ein Softwaresystem adäquat zu dokumentieren. Ebenso existiert eine Vielzahl von Modellierungssprachen, die hierbei zum Einsatz kommen können. Beispielhaft seien die 4+1 Sichten von Kruchten [Kr95] oder die Sichten und Stile von Clements et al. [Cl02] genannt.

Um die verschiedenen Einflussfaktoren bei der Architekturdokumentation genauer zu betrachten, werden in Abschnitt 3.1.1 die Konzepte des IEEE 1471 [IE00] erläutert, die aufzeigen, welche Aspekte bei der Dokumentation von Architekturen zu berücksichtigen sind. Ein weiterer Teil dieser Einführung ist eine Kritik am IEEE 1471, die sich aus einer Modellierungsschwäche ergibt (siehe auch Lankes, Matthes und Wittenburg [LMW05]).

Aufbauend auf dieser Einführung in den IEEE 1471 wird die Rolle von Modellen und Methoden bei der Visualisierung von Anwendungslandschaften aufgezeigt, um die für diese Arbeit wesentlichen Begriffe Modell und Methode zu definieren (siehe Abschnitt 3.1.2).

3.1.1. Architekturdokumentation im Kontext des IEEE 1471

Die Notwendigkeit, für bestimmte Stakeholder relevante Merkmale des zu beschreibenden Systems (auf hinreichend hohem Abstraktionsniveau) abzubilden, führt zu einer Menge von Darstellungen. Diese Darstellungen fließen in eine Architekturdokumentation ein bzw. eine Architekturdokumentation besteht aus denselben.

Diese Vorgehensweise ist zentrales Leitbild des IEEE 1471, der Empfehlungen zur Beschreibung der Architektur von Software-intensiven Systemen gibt und als Ziel die Unterstützung des Dokumentierens, Explizierens und Kommunizierens von Architekturen hat [IE00].

Im Gegensatz zu den verschiedenen Typen von Architekturdarstellungen, wie beispielsweise Kruchten [Kr95] oder Clements et al. [Cl02], die graphische Notationen zur Beschreibung von Systemen bereitstellen, adressiert der IEEE 1471 Probleme, die der Erstellung von eindeutigen, klaren und verständlichen Architekturbeschreibungen entgegenstehen. Eines der adressierten Probleme ist das Auftreten unklarer Begriffsverwendungen.

Der Standard legt Definitionen zu Schlüsselkonzepten und Begriffen fest, soweit sich zu diesen eine Konsensbildung feststellen lässt. Im Folgenden werden die Terminologie und die Konzepte des IEEE 1471 vorgestellt. Wie sich der IEEE 1471 mit Modellierungssprachen wie UML verbindet, wird beispielsweise von Hilliard [Hi99] dargestellt.

Grundsätzlich befasst sich der IEEE 1471 mit Software-intensiven Systemen. Unter diese Bezeichnung fallen sämtliche Systeme, bei denen Software einen essentiellen Einfluss auf Design, Konstruktion, Einsatz (engl. *Deployment*) und Evolution des Systems in seiner Gesamtheit ausübt [IE00]. Damit ist der IEEE 1471 auch auf Anwendungslandschaften anwendbar.

3.1.1.1. Terminologie und Konzepte

Neben dem Begriff Architektur, der bereits entsprechend des IEEE 1471 eingeführt wurde, definiert der IEEE 1471 elf weitere Begriffe in einem konzeptuellen Modell[2], die Abbildung 3.1[3] dargestellt werden und wie folgt definiert sind:

- *Stakeholder* (deutsch Interessenvertreter) sind nach dem IEEE 1471 Einzelpersonen, Teams oder Organisationen, die bestimmte *Concerns* (deutsch Interessen) am betrachteten System besitzen. Dies gilt auch bei der Darstellung von Anwendungslandschaften.
- Im IEEE 1471 kann eine *View* (deutsch Sicht) aus mehreren *Models* (deutsch Modelle) bestehen. Jedes dieser *Models*, die im IEEE 1471 auch als *Architectural Models* bezeichnet werden, basiert auf im zugehörigen *Viewpoint* (deutsch Blickwinkel) definierten Methoden inklusive Notationen. Dabei verhält sich ein Viewpoint zu einem View, wie eine Klasse zu einem Objekt. Zweck eines Viewpoints ist es, bei der Adressierung der ihm zugewiesenen Concerns unterstützend zu wirken. So kann beispielsweise eine Management-View aus einem Organisationsmodell und einem Kostenmodell bestehen, wobei jedes Model eine eigene Notation verwendet.

[2] Die englischen Begriffe des IEEE 1471 werden in der Beschreibung durchgehend verwendet. Wenn ein passender Terminus in der deutschen Sprache existiert, so wird dieser in Klammern angegeben.

[3] Fehlende Multiplizitäten wurden in der Darstellung ergänzt, um eine UML-konforme Darstellung zu erhalten.

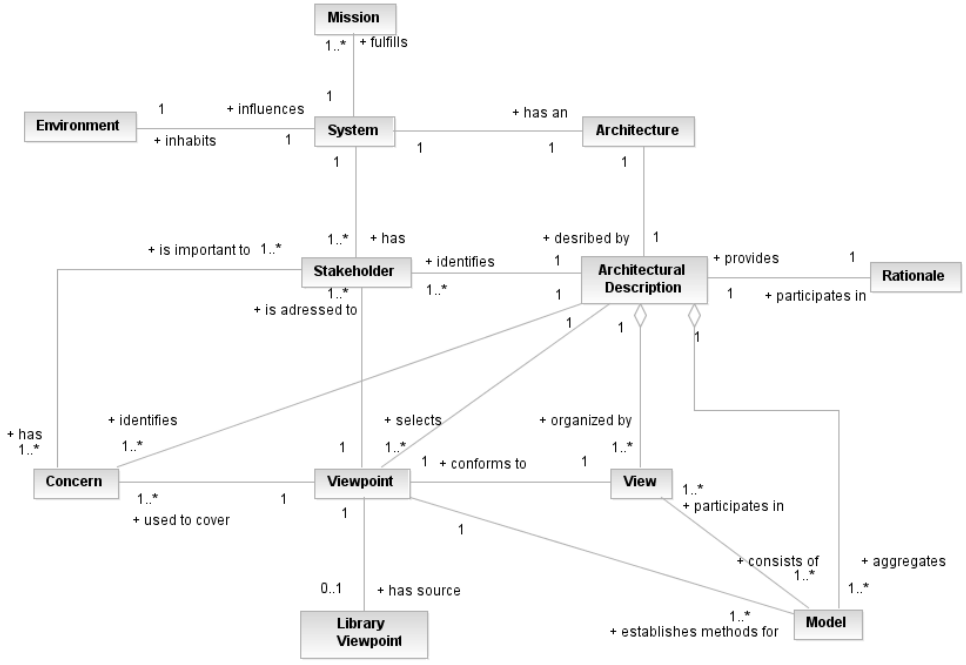


Abbildung 3.1.: Konzeptuelles Modell des IEEE 1471 [IE00]

- Als *Library Viewpoint* bezeichnet der IEEE 1471 Viewpoints, die nicht im Rahmen der Erstellung der betrachteten *Architectural Description* (deutsch Architekturbeschreibung) entwickelt wurden. Solche Viewpoints können aus der Literatur (z. B. Kruchten [Kr95]) stammen oder z. B. innerhalb einer Organisation als Standard vorgegeben sein. Im Rahmen einer bestimmten Architectural Description kann ein Library Viewpoint nach Anpassung an projektspezifische Gegebenheiten als Viewpoint zum Einsatz kommen.
- Mit dem Begriff *System* (deutsch System) bezeichnet der IEEE 1471 das zu beschreibende Software-intensive System, welches ein Teil eines Systems, ein einzelnes System im Ganzen oder ein aggregiertes System („system of systems“ [IE00]) sein kann. In der Softwarekartographie ist das zu beschreibende System die Anwendungslandschaft oder ein Teil dieser.
- Die *Architectural Description* (deutsch Architekturbeschreibung) setzt sich entsprechend des IEEE 1471 aus Views und Models zusammen. Die Architectural Description enthält somit alle für die Stakeholder relevanten Informationen zur Architektur.
- Der *Rationale* (deutsch Beweggrund) beschreibt die Gründe, die zur Auswahl der Architektur geführt haben, und den Zweck, den der Architekt mit seinen Entscheidungen verfolgt. Die Dokumentation von Sachverhalten bezüglich einer Architektur, die außerhalb der einzelnen Views stehen und für diese gemeinsam gelten, beschreiben auch Clements et al. [Cl02]. Dieses Vorgehen ist auch für Anwendungslandschaften sinnvoll, da es zu einem besseren Verständnis der Architektur beiträgt

und unnötiges Wiederaufleben von bereits geführten Diskussionen zu Architekturentscheidungen verhindert.

- Die *Environment* (deutsch Umgebung) des zu beschreibenden Systems wird vom IEEE 1471 als die Umgebung oder der Kontext, der einen Einfluss auf die Gestaltung des Systems ausübt, definiert. Dies umfasst auch andere Systeme, mit denen das betrachtete System interagiert. Damit legt die Environment auch die Grenzen des betrachteten Systems fest. Für Anwendungslandschaften besteht die *Environment* aus allen Faktoren, welche die Gestaltung der Anwendungslandschaft beeinflusst haben und beeinflussen. Zu diesen Einflussfaktoren zählen z. B. Organisationsstrukturen, Strategien oder technische Rahmenbedingungen.
- Als *Mission* (deutsch Aufgabe) betrachtet der IEEE 1471 den Verwendungszweck, den die Stakeholder dem System zugedacht haben.

3.1.1.2. Diskussion des IEEE 1471

Die Stärken des IEEE 1471 liegen vor allem in der Terminologie und den verwendeten Konzepten, die in Abschnitt 3.1.1.1 eingeführt wurden. Insbesondere das deutliche Hervorheben von *Stakeholder*, *Concern* und *View* stellt einen Wertbeitrag des IEEE 1471 dar. Architekturbeschreibungen, die im Sinne des IEEE 1471 u. a. aus Views bestehen und nicht auf die Interessen (*Concerns*) der an der Architektur interessierten Personen (Stakeholder) eingehen, sind wertlos.

Doch eben diese Terminologie kann auch irreführend sein, da beispielsweise das Konzept *Model* nicht mit der Verwendung im *Model-View-Controller* Entwurfsmuster [Bu96] oder einem Informationsmodell übereinstimmt. Das *Model* im IEEE 1471 meint ein *Architectural Model*, welches bereits eine vom *Viewpoint* vorgegebene Notation zur Modellierung verwendet. Im Kontext von Anwendungslandschaften wird der Begriff Modell zwar ebenso unterschiedlich benutzt, beruht aber auf einer einheitlichen Begriffsdefinition und wird, wenn bestimmte Modelle gemeint sind, mit einem Präfix oder Suffix versehen. Das Informationsmodell (siehe Abschnitt 4.2) ist ein Modell, welches die Objekttypen (Anwendungssysteme, Geschäftsprozesse, Kosten etc.) und deren Assoziationen definiert, aber noch keine Visualisierung festlegt. Eine Instanz eines Informationsmodell ist ein so genanntes *semantisches Modell* (siehe Abschnitt 4.1) und eine visuelle Darstellung eines semantischen Modells ist ein *symbolisches Modell* (siehe Abschnitt 4.1), welches dann dem Modellbegriff des IEEE 1471 entspricht.

Des Weiteren stehen die Begriffe Viewpoint und View in einer 1:1 Beziehung, die auf den ersten Blick dem im Standard als Beispiel angeführtem Verhältnis analog von Klasse zu Objekt [IE00] widerspricht. Diese Multiplizität der Assoziation wird mit einer Redundanzfreiheit begründet, da für eine bestimmte *Architectural Description* ein Viewpoint nur einen korrespondierenden View besitzt. Auf der anderen Seite ist, wie z. B. von Niemann [Ni05] oder Starke [St05a] beschrieben, die Architektur einer Anwendungslandschaft oder eines Softwaresystems nicht statisch, sondern verändert sich im Rahmen der Bestrebungen, der *Mission* gerecht zu werden. Zur Beschreibung der Veränderung von Architekturdokumentationen bzw. Versionierung von Views liefert der IEEE 1471 keine

Konzepte. Dies widerspricht sowohl der iterativen Entwicklung von Softwaresystemen, in denen sich Views in den einzelnen Schritten verändern, als auch dem Management von Anwendungslandschaften, bei dem zwischen mehreren Versionen von Ist-, Plan- und Soll-Landschaften unterschieden wird.

Ein weiterer Kritikpunkt ist die verwendete Notation im konzeptuellen Diagramm des IEEE 1471. In der Notiz zum konzeptuellen Modell wird angemerkt, dass die Notation für Aggregationen von UML übernommen wurde, was nicht zwingend eine Übernahme der gesamten UML-Notation für das Diagramm darstellt. Assoziationen werden hingegen mittels einer freien Notation dargestellt, wobei keine Unterscheidung zwischen Rollen- und Assoziationsnamen vorgenommen wird. Ebenso wird eine Assoziation ohne Angabe von Multiplizitäten als eine 1:1 Beziehung modelliert, welches der UML-Notation widerspricht, da UML zumindest die Annotation einer Obergrenze, bspw. * für 0..* (0 bis unendlich), verlangt [OM05d].

Um Missverständnisse mit der bekannten UML-Notation zu vermeiden, wurden in Abbildung 3.1 bereits alle fehlenden Multiplizitäten entsprechend des Hinweises im IEEE 1471 ergänzt und alle Namen an Assoziationen als Rollennamen interpretiert, obwohl die Bezeichnungen keine *Rollen* entsprechend UML darstellen. Eine Interpretation der Beschriftungen im IEEE 1471 als Assoziationsnamen ist jedoch ebenso nicht möglich, da in UML eine Assoziation nur genau einen Assoziationsnamen besitzt, im IEEE 1471 jedoch einige Assoziationen zwei Beschriftungen aufweisen.

Neben den obigen Kritikpunkten besitzt das konzeptuelle Modell des IEEE 1471 einen Missstand, der im folgenden Abschnitt 3.1.1.3 diskutiert wird. Zusätzlich wird ein Lösungsvorschlag aufgezeigt, der den Missstand in der Modellierung aufhebt.

3.1.1.3. Erweiterung des IEEE 1471 um das Konzept Modellierungssprache

Im Sinne des IEEE 1471 etabliert ein Viewpoint, der bestimmte Interessen (*Concerns*) an dem betrachteten System adressiert, eine Methode für ein Modell. Eine Methode meint in diesem Kontext, dass der Viewpoint beschreibt, wie das Modell zu erstellen ist.

In das konzeptuelle Modell hat dieses Verständnis durch die Assoziation zwischen *Viewpoint* und *Model* Eingang gefunden (siehe Abbildung 3.2). Ein Viewpoint adressiert die Interessen an dem System in einer abstrakten Art und Weise und wird erst durch seine Instantiierung zu einer Sicht auf das System, welches in der Terminologie des IEEE 1471 ein *View* ist. Der View beantwortet somit die Fragen eines *Stakeholder* (Interessenten), während der Viewpoint vorschreibt, wie der View auszusehen hat.

Der Tatsache, dass ein View aus mehreren Teilen zusammengesetzt sein kann, trägt der IEEE 1471 Rechnung, indem er vorsieht, dass ein View aus mehreren Modellen bestehen kann. Auf der anderen Seite kann ein Modell Teil verschiedener Views sein, da ein einzelnes Modell ggf. Informationen beinhaltet, die für verschiedene Views relevant sind. Zusätzlich führt der Standard aus, dass „each such architectural model is developed using the methods established by its associated architectural viewpoint“ [IE00, S. 5].

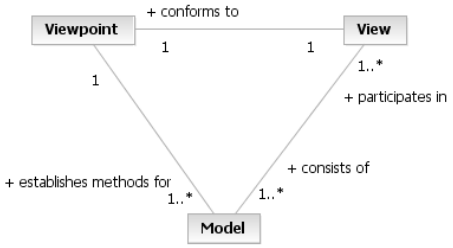


Abbildung 3.2.: Ausschnitt Viewpoint, View und Model aus dem konzeptuellen Modell des IEEE 1471

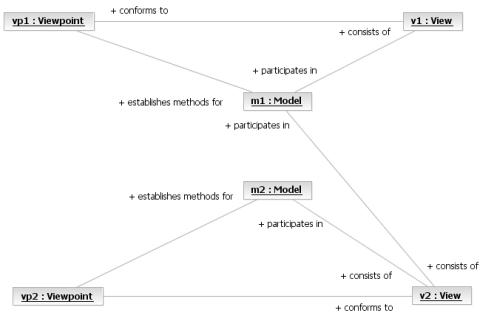


Abbildung 3.3.: Instantiierter Ausschnitt Viewpoint, View und Model aus dem konzeptuellen Modell des IEEE 1471

Dieses Dreieck der Elemente *Viewpoint*, *View*, *Model* und den Beziehungen zwischen den Elementen führen, wenn man die Elemente instantiiert, zu einem Missstand, welcher exemplarisch in Abbildung 3.3 dargestellt ist. In der Abbildung ist `vp1` ein Viewpoint, der die Methodik für das Modell `m1` vorschreibt und `vp2` ein zweiter Viewpoint, welcher die Methode für das Modell `m2` vorschreibt. Der View `v1` verwendet ausschließlich das Modell `m1`, dies ist konform zum IEEE 1471. Der View `v2` besteht aus den Modellen `m1` *und* `m2`, dies ist zwar konform zur vorgegebenen Modellierung aus Abbildung 3.2, widerspricht jedoch dem Zusatz des IEEE 1471 in Teilen, da ein View ein Modell verwendet, dessen Viewpoint nicht die Methode vorgibt. Durch die Multiplizität 1:1 zwischen Viewpoint und View ist es somit ggf. auch notwendig einen View zu instantiieren, um ein Modell in einem anderen View zu verwenden, obwohl der View nicht verwendet wird.

Bestätigt wird diese These von einem der Autoren des IEEE 1471, der in einer E-Mail das konzeptuelle Modell als „dark corner“ bezeichnet: „Architectural model is definitely one of the dark corners of IEEE 1471-2000. It is a fairly informal notion in that version.“ [Hi05b].

Dieser Missstand ist begründet in der Mischung von zwei Meta-Ebenen in einem konzeptuellen Modell. Der IEEE 1471 enthält sowohl den Viewpoint als abstrakte Beschreibung als auch den View als ontologische Instantiierung[4]. Jedoch existiert nur die Klasse Model ohne ein *passendes* Gegenstück für die Instantiierung.

Eine Möglichkeit, diesen Missstand zu beseitigen, ist die Einführung eines Konzeptes *Modelling Language* (Modellierungssprache), welches die von einem oder mehreren Modellen verwendete Sprache beschreibt. Durch die Einführung dieses Konzeptes (siehe Abbildung 3.4) existieren sowohl für den View als auch für das Modell die ontologischen Metaklassen, so dass ein Viewpoint eine oder mehrere Modellierungssprachen nutzen kann, um einen View zu erstellen. Ein Modell verwendet wiederum genau eine Modellierungssprache.

[4]Zu ontologischer vs. linguistischer Instantiierung siehe Hitz et al. [Hi05a, S. 306].

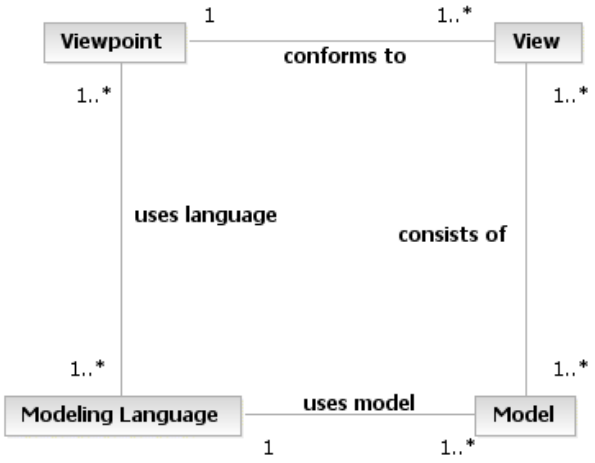


Abbildung 3.4.: Erweiterung des konzeptuellen Modells des IEEE 1471 um das Konzept Modellierungssprache

In Abbildung 3.4 eingeflossen ist auch die Kritik an der Kardinalität zwischen Viewpoint und View (siehe Abschnitt 3.1.1.2), so dass eine zu Abbildung 3.4 konforme Instantiierung mehrere Views zu einem Viewpoint zulässt.

3.1.2. Modelle und Methoden

Abschnitt 3.1.1 hat die Bedeutung von Modellen und Methoden in der Architekturdokumentation dargelegt. Da jedoch unter einem Modell und einer Methode in der Informatik Unterschiedliches subsumiert wird, werden im folgenden konkrete Definitionen für diese Begriffe eingeführt, die im weiteren Verlauf als Basis dienen.

3.1.2.1. Modelle in der Architekturdokumentation

Die exakte Bedeutung sowie Abgrenzung zwischen Modellbegriffen, und die Frage ob diese einen abbildungsorientierten oder konstruierenden Charakter besitzen, haben bereits zu Diskussionen geführt (siehe [Sc99, Sc00, Ka00]). Im Falle von Anwendungslandschaften – dem zu untersuchenden Original in dieser Arbeit – ist festzuhalten, dass Modelle und die zugehörigen Originale in unterschiedlichen Beziehungen zueinander stehen können: Wird eine Ist-Landschaft modelliert, so handelt es sich um eine Abbildung der Realität; bei einer Plan- oder Soll-Landschaft wird ein zukünftiger, fiktiver Zustand modelliert. Für beide Arten von Beziehungen zwischen dem Modell und dem Original sind

die Hauptmerkmale des allgemeinen Modellbegriffs von Stachowiak [St73, S.131ff] zutreffend:

Modell : Ein Modell (engl. *Model*) zeichnet sich durch die folgenden Hauptmerkmale aus:

Abbildungsmerkmal: Modelle sind stets Modelle von etwas, nämlich von Abbildungen, Repräsentationen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können.

Verkürzungsmerkmal: Modelle erfassen im allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellerschaffern und/oder Modellbenutzern relevant scheinen.

Pragmatisches Merkmal: Modelle sind ihren Originalen nicht per se eindeutig zugeordnet. Sie erfüllen ihre Ersetzungsfunktion

- für bestimmte – erkennende und/oder handelnde, modellbenutzende – Subjekte,
- innerhalb bestimmter Zeitintervalle und
- unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen.

Die drei Hauptmerkmale *Abbildungsmerkmal*, *Verkürzungsmerkmal* und *pragmatisches Merkmal* des allgemeinen Modellbegriffs nach Stachowiak definieren die grundlegenden Eigenschaften eines Modells. Insbesondere die Beziehung zwischen einem Modell und dem durch das Modell repräsentierten Originals ist eine wesentliche Aussage, da ein Modell ohne ein korrespondierendes natürliches oder künstliches Original kein Modell im Sinne von Stachowiak ist. In der Informatik findet sich gleichfalls das Verkürzungsmerkmal, da Modelle einfache Repräsentationen eines Originals darstellen sollen. Im Software-Engineering kann beispielsweise ein UML-Klassendiagramm ein Modell eines Quelltextes sein und hierbei bestimmte Eigenschaften weglassen, z. B. die Implementierung der Methodenrümpfe.

Werden die Bestandteile eines Modells weitergehend untersucht, so wird in der Informatik stets zwischen Syntax und Semantik unterschieden. Welche Semantik und Syntax einem Modell zugrunde liegen, definiert das (linguistische) Metamodell des Modells. Der Zusammenhang zwischen Semantik und Syntax in einem Metamodell enthält die folgenden Konzepte (siehe Abbildung 3.5[5]):

Modellierungssprache: Die Modellierungssprache setzt sich aus den Konzepten Semantik, abstrakte Syntax, konkrete Syntax und Serialisierungssyntax zusammen. Ein Modell verwendet zur Modellbildung eine Modellierungssprache, die mindestens aus einer abstrakten Syntax und einer Semantik besteht.

[5] Eigene Darstellung in Anlehnung an Kühn [Kü06].

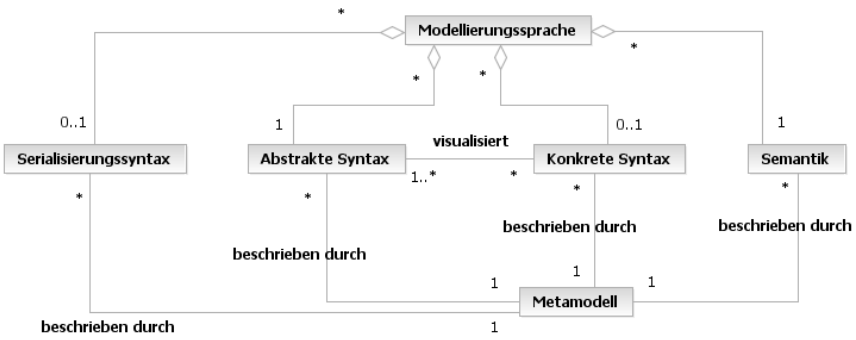


Abbildung 3.5.: Bestandteile einer Modellierungssprache und eines Metamodells

Metamodell: Das Metamodell beschreibt die Semantik, abstrakte Syntax, konkrete Syntax und Serialisierungssyntax, wobei ein einzelnes Metamodell mehrere Syntaxen und Semantiken beschreiben kann.

Semantik: Die Semantik beschreibt die Bedeutung der Modellierungssprache. Zur Beschreibung der Semantik können formale, semi-formale oder informelle Beschreibungen verwendet werden. In der theoretischen Informatik werden die formalen Beschreibungsformen operationale, denotationelle und axiomatische Semantik unterschieden [Sc95]. Hinzukommt die Übersetzersemantik, die eine existierende Beschreibung einer Semantik verwendet und hierbei die eigene Semantik auf die existierende übersetzt. Die semi-formale Semantik beschreibt eine Semantik mittels natürlichsprachlichen und formalen Konzepten, wobei die Beschreibung nicht durchgehend formal und nicht eindeutig auf eine der formalen Konzepte abbildbar ist. Die informelle Beschreibung einer Semantik erfolgt mittels natürlicher Sprache.

Abstrakte Syntax: Die abstrakte Syntax beschreibt die Sprachkonzepte und wie diese kombiniert werden dürfen, jedoch nicht die Repräsentation oder Bedeutung der Konzepte.

Konkrete Syntax: Die konkrete Syntax beschreibt die (graphische) Notation, um Sprachkonzepte intuitiv (d.h. für den Benutzer) darzustellen. Sie wird häufig eingesetzt, um Modelle zu konstruieren, da sie verständlicher ist als die abstrakte Syntax.

Serialisierungssyntax: Die Serialisierungssyntax beschreibt die Konzepte zur Serialisierung eines Modells, um dies beispielsweise in Werkzeugen persistent zu speichern oder zwischen Werkzeugen auszutauschen.

Übertragen auf UML ist die konkrete Syntax informell mittels Tabellen beschrieben, in denen definiert wird, welche UML-Diagramme welche graphischen Mittel verwenden dürfen. Es wird beispielsweise beschrieben, dass in einem Strukturdiagramm ein Rechteck ohne weitere Stereotypen eine Klasse repräsentiert und dass eine Linie eine Assoziation

darstellt (siehe [OM05d, S. 135f]). Detaillierte Teile der Notation, wie beispielsweise die Signatur einer Operation, werden in der Backus-Naur-Form (BNF) angegeben.

Die Definition der abstrakten Syntax ist in UML unvollständig und wird nur teilweise mittels OCL-Bedingungen[6] angegeben. Es wird beispielsweise definiert, dass eine Instanz einer Assoziation, ein so genannter *Link*, zwischen zwei Objekten eindeutig ist und die Zulässigkeit mehrfacher Links muss explizit durch eine Annotation der Assoziation angegeben werden.

Die Semantik von UML wird mittels natürlicher Sprache definiert, wobei sich die Beschreibung der Semantik auf die *UML Superstructure* [OM05d] und die *UML Infrastructure* [OM05c] verteilt. Zusätzliche Variationspunkte in der Semantik werden explizit eingeführt, um den Anspruch einer *universellen* Modellierungssprache gerecht zu werden. Die Serialisierungssyntax von UML wird in der *XML Metadata Interchange* (XMI) [OM05b] beschrieben.

Werden formale Modellierungssprachen, wie beispielsweise S/T-Netze[7] betrachtet, so ist auch dort die Trennung der einzelnen Konzepte vorhanden. Die konkrete Syntax von S/T-Netzen, eine Form von Petri-Netzen, besteht aus Kreisen, Rechtecken und Linien. Die abstrakte Syntax eines S/T-Netzes wird als 6-Tupel (siehe [JV87]) und die Semantik mittels einer denotationellen Semantik beschrieben.

Nachdem die Bestandteile einer Modellierungssprache vorgestellt wurden, wird ersichtlich, warum dieses Konzept Eingang (siehe Abschnitt 3.1.1.3) in den IEEE 1471 finden sollte. Eine Architekturdefinition, die sich wesentlich auf Modelle abstützt, muss die verwendete Modellierungssprache entweder selbst umfassen oder eine geeignete Modellierungssprache referenzieren. Die bereits zu Beginn des Abschnitt 3.1 angeführten Modellierungssprachen von Clements et al., Kruchten oder auch die Unified Modeling Language versuchen dies in unterschiedlichem Detaillierungsgrad und mit unterschiedlichen Mitteln.

Im weiteren Verlauf dieser Arbeit werden die einzelnen Bestandteile einer Modellierungssprache für die Softwarekartographie eingeführt. Die Unterscheidung der Elemente abstrakte Syntax, konkrete Syntax, Serialisierungssyntax und Semantik wird wiederholt aufgegriffen, um die einzelne Bausteine in Beziehung zu setzen und gegeneinander abzugrenzen.

3.1.2.2. Methoden in der Architekturdokumentation

Der Begriff *Methode* wird in der Informatik vielfältig genutzt und hat je nach Kontext eine unterschiedliche Bedeutung. In einer objektorientierten Programmiersprache ist eine Methode eine Funktion bzw. Operation einer Klasse, im Software-Engineering ist eine Methode die Beschreibung eines Verfahrens bzw. Vorgehens zur Entwicklung eines

[6] OCL - Object Constraint Language, siehe OMG [OM06e]
[7] S/T-Netz: Stellen/Transitions-Netz

Softwaresystems. In dieser Arbeit wird eine Methode als eine Vorgehens- und Verfahrensbeschreibung verstanden und wie folgt definiert:

> **Methode :** Eine Methode (engl. *Method*) ist eine Vorgehensbeschreibung, die sich aus Schritten und einer Abfolge dieser Schritte zusammensetzt, um ein bestimmtes Ergebnis zu erreichen. Eine Methode kann eine Modellierungssprache nutzen, wenn in den einzelnen Schritten Modelle verwendet werden.

Die obige Begriffsdefinition ist an Kronlöff et al. [KSH93] angelehnt, wobei Kronlöff et al. keine Modellierungssprache verlangen, sondern eine Sprache und ein zugrundeliegendes Modell. Da jedoch eine Modellierungssprache bereits in Abschnitt 3.1.2.1 als Aggregation von Syntax und Semantik definiert wurde, die in einem Modell abgebildet wird, verwendet obige Definition hierzu die Modellierungssprache.

Eine Methode zur Architekturdokumentation nutzt somit eine Modellierungssprache, um mittels einer Abfolge von definierten Schritten und unter zu Zuhilfenahme einer Anleitung ein Ergebnis zu erzielen.

Im Folgenden werden schrittweise die Teile für eine Modellierungssprache zur Architekturdokumentation von Anwendungslandschaften erarbeitet, welche die Sprache für die Modelle in der Softwarekartographie darstellt. Die Methoden, die im Management für Unternehmensarchitekturen verwendet werden und wie diese in Beziehung zu anderen Methoden des IT-Managements stehen, wurden bereits grundlegend in Abschnitt 2.1.7 eingeführt. Eine für die Softwarekartographie anwendungsbezogene Verfeinerung folgt in Abschnitt 5.

Bevor in Abschnitt 3.3 Modelle zur Architekturdokumentation von Anwendungslandschaften aus der Praxis vorgestellt werden, wird im folgenden Abschnitt 3.2 die Anwendung der Kartographie für die Softwarekartographie dargelegt. Der Abschnitt baut ein Verständnis für die Kartographie auf und führt grundlegende Begriffe ein, die bei der Analyse der Architekturdokumentation von Anwendungslandschaften verwendet werden.

3.2. Anwendung der Kartographie in der Softwarekartographie

Ziel dieses Abschnittes ist es, das Vokabular zur Beschreibung von Architekturdokumentationen in der Softwarekartographie einzuführen und hierbei das Verständnis für Karten in der Kartographie zu nutzen, um die auf die Softwarekartographie übertragbaren und anwendbaren Konzepte aufzuzeigen. Die Kartographie als die Disziplin, die sich mit der Konzeption, der Erstellung, der Verbreitung und dem Studium von Karten beschäftigt[8], stellt grundlegende Definitionen für Karten, deren Aufbau und ihre Verwendung bereit, die ebenso für die Softwarekartographie relevant sind.

[8] Übersetzung der Begriffsdefinition für Kartographie der Internationalen Kartographischen Vereinigung, entnommen aus Hake et al. [HGM02, S. 4].

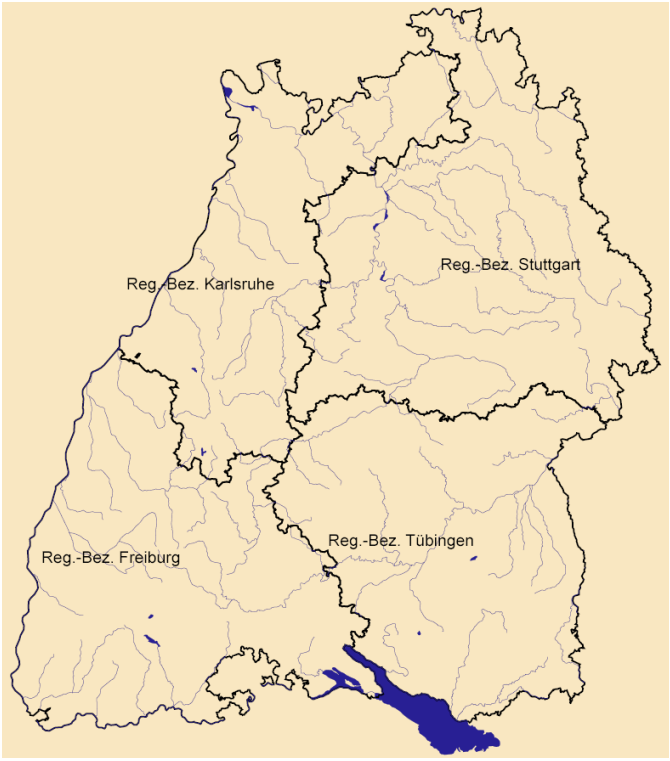


Abbildung 3.6.: Topographische Karte von Baden-Württemberg [La06]

Eine Karte ist in der Kartographie eine symbolisierte Abbildung einer geographischen Realität, die räumliche Bezüge in einem Modell darstellt [HGM02, S. 25]. Klassifikationen für Karten existieren in verschiedenster Form, wobei eine vorherrschende Klassifikation [HGM02, S. 27] nach dem Karteninhalt zwischen topographischen und thematischen Karten unterscheidet.

Topographische Karten stellen semantische und geometrische Objektinformationen dar; typische Beispiele für derartige Karten sind Land- oder Stadtkarten. Thematische Karten besitzen einen topographischen Kartengrund und ergänzen die topographische Karte um thematische Informationen. Bekannte Beispiele für derartige Informationen sind die Bevölkerungsdichte oder Temperatur. Netzpläne, die beispielsweise Nahverkehrsnetze in Großstädten darstellen, zählen auch zu den thematischen Karten, genauer zu den Topogrammen. Abbildung 3.6 zeigt eine topographische Karte des Landes Baden-Württemberg und Abbildung 3.7 ein thematische Karte, die als Kartengrund die topographische Karte aus Abbildung 3.6 verwendet und die Bevölkerungsverteilung in verschiedenen Altersstufen mittels Kreissektorendiagrammen darstellt.

Für die Softwarekartographie von besonderem Interesse sind die thematischen Karten, da diese mittels des kartographischen Zeichensystems verschiedene Merkmale auf Karten geeignet visualisieren.

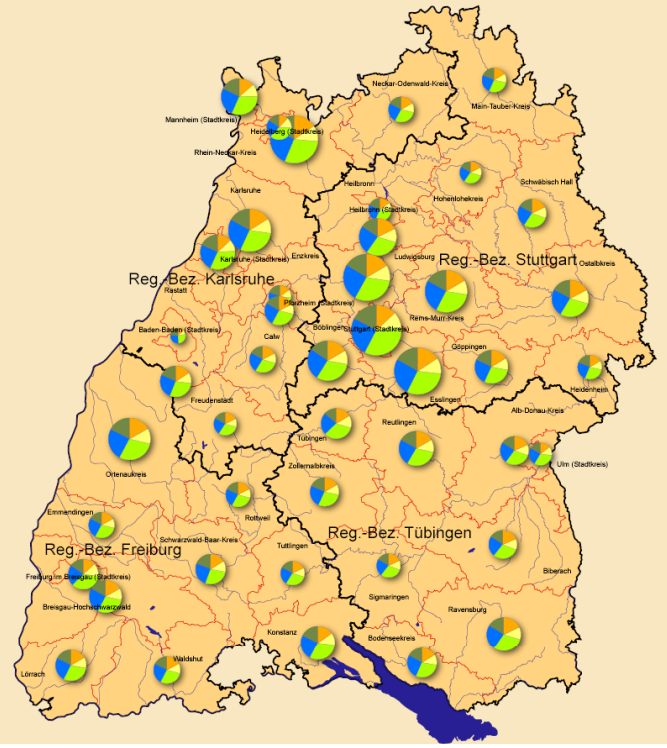


Abbildung 3.7.: Thematische Karte mit Bevölkerungsverteilung in Baden-Württemberg mittels Kreissektorendiagrammen [La06]

3.2.1. Kartographisches Zeichensystem

Mit der Gesamtheit aller und insbesondere der kartographischen Darstellungen von Karten befasst sich die Kartengraphik. Merkmale und Regeln dieser Darstellungen werden dabei in einem so genannten Zeichensystem zusammengefasst.

Hake et al. [HGM02] beschreiben die Struktur des Zeichensystems in einem dreistufigen Aufbau aus graphischen Elementen, zusammengesetzten Zeichen und graphischen Gefügen. Als Bausteine jeder Kartengraphik fungieren die *graphischen Grundelemente* (Punkte, Linien, Flächen), welche gemäß ihrer geometrischen Ausbreitung zu unterscheiden sind. Zu höheren Gebilden gruppierte graphische Elemente werden als *zusammengesetzte Zeichen* (Signatur, Diagramm, Halbton, Schrift) bezeichnet. Die graphischen Grundelemente und die zusammengesetzten Zeichen bilden die Menge der *Gestaltungsmittel* für die Kartographie:

> **Gestaltungsmittel :** Gestaltungsmittel (engl. *Map Symbols*) sind die graphischen Grundelemente (Punkt, Linie, Fläche) sowie die zusammengesetzten Zeichen (Signatur, Diagramm, Halbton, Schrift).

Punkte dienen entweder der Kennzeichnung der Lage eines Objekts, oder sie gelten als Teil eines Punktrasters oder ähnlicher Anordnung als Elemente flächenhafter Signatu-

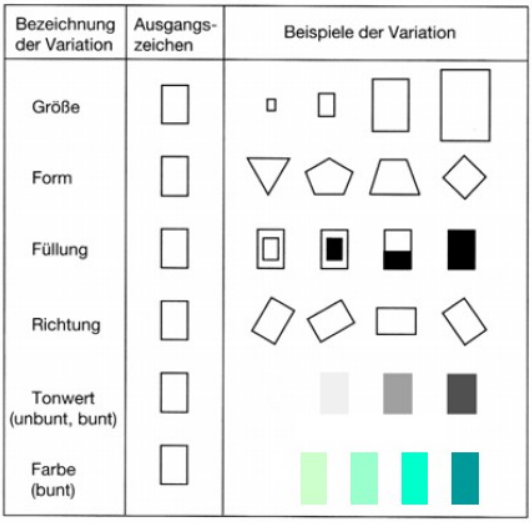


Abbildung 3.8.: Gestaltungsvariablen eines Zeichens nach Hake et al. [HGM02]

ren. Als zweites kartographisches Gestaltungsmittel gilt die Kategorie der Linien, die alle nicht unterbrochenen Striche zur Angabe einer Lage enthält. Die Fläche als drittes Grundelement, die in ihrer gesamten Ausdehnung in Farbton und Tonwert konstant ist, gestattet Aussagen bzgl. Lage und Qualität flächenhafter Diskreta, flächenbezogenen Quantitäten sowie Wertstufen eines Kontinuums.

Durch ihre Flexibilität bzgl. Ausdruck und Anwendung werden die Signaturen, die bildhafte, symbolhafte, geometrische Figuren sowie Buchstaben einschließen, zu einem der wichtigsten Gestaltungsmittel der Kartographie.

Die Softwarekartographie unterscheidet des Weiteren zwischen einem Gestaltungsmittel und einer Gestaltungsmittelinstanz:

Gestaltungsmittelinstanz : Ein Gestaltungsmittelinstanz ist die Instanz eines Gestaltungsmittels und verhält sich zu diesem analog wie ein Objekt zu seiner Klasse.

Bestimmte Sachverhalte lassen sich jedoch nicht ausschließlich durch die Elemente des kartographischen Zeichensystems visualisieren, sondern nur, indem Variationen dieser Elemente stattfinden. Diese Variationen werden als Gestaltungsvariablen bezeichnet. Sechs dieser visuellen Variablen zur Symbolisierung und Differenzierung von Zeichen finden sich in Abbildung 3.8.

Gestaltungsvariable : Eine Gestaltungsvariable (engl. *Visual Variable*) beeinflusst die Erscheinung einer Instanz eines Gestaltungsmittels, indem die Variable die Instanz hinsichtlich Größe, Form, Füllung, Tonwert, Richtung und Farbe verändert. Unterschiedliche Gestaltungsmittel besitzen unterschiedliche Gestaltungsvariablen.

Neben den Gestaltungsmitteln und den Gestaltungsvariablen wird in der Softwarekartographie ein weiteres Konzept für das Zeichensystem benötigt, welches in dieser Form in der Kartographie nicht explizit definiert ist. Abbildung 3.7 zeigt eine thematische Karte, die Kreissektorendiagramme relativ zu einer bestimmten Fläche positioniert. Die Position eines Kreissektorendiagramms gibt Aufschluss darüber zu welcher Fläche die visualisierten Informationen gehören. Diese Positionierung wird in der Softwarekartographie über eine *Gestaltungsregel* beschrieben, die definiert, dass ein bestimmtes Kreissektorendiagramm beispielsweise innerhalb der ihm zugehörigen Fläche visualisiert werden muss.

Gestaltungsregel : Eine Gestaltungsregel (engl. *Visualization Rule*) definiert Darstellungsbeschränkungen oder Darstellungswünsche auf einer oder mehreren Gestaltungsmittelinstanzen.

Innerhalb der Softwarekartographie existieren verschiedene Gestaltungsregeln, die sich aus den Beispielen für Softwarekarten in Abschnitt 3.3 ergeben und die sukzessive anhand der Beispiele eingeführt werden. Die Unterscheidung zwischen Beschränkungen und Wünschen resultiert aus der Notwendigkeit, zum einen Beschränkungen, wie Verschachtelung, aber auch Wünsche, wie Flächenminimierung, beschreiben zu können (siehe Abschnitt 4.3). Darstellungsbeschränkungen müssen eingehalten werden, um eine semantisch korrekte Darstellung zu erhalten, Darstellungswünsche sollen möglichst gut erfüllt werden, um die Lesbarkeit zu optimieren und *ästhetisch* ansprechende Karten zu erhalten.

3.2.2. Aufbau von thematischen Karten

Thematische Karten nutzen vorwiegend einen topographischen Kartengrund, um auf diesem Kartengrund zusätzliche Informationen zu visualisieren. Der Kartengrund dient hierbei als Referenzschicht. Weitere Informationen der thematischen Karte werden über den Elementen der Referenzschicht, zum Beispiel einer Fläche, positioniert, um die Beziehung zwischen dem Element der Referenzschicht und den Elementen der referenzierenden Schicht zu visualisieren. In Abbildung 3.7 wurden die Kreissektorendiagramme derart über dem topographischen Kartengrund positioniert, dass erkennbar wird, zu welcher Fläche die Informationen des Kreissektorendiagramms gehört.

Dieses Schichtenprinzip, Karten aus mehreren aufeinander aufbauenden Schichten zu gestalten, wird in der Kartographie häufig angewendet. Der Kartengrund besitzt in der Kartographie eine besondere Bedeutung und wird je nach Zielsetzung der thematischen Karte verändert und soll als *Hintergrundinformation* dienen [HGM02, S. 486f].

Da thematische Karten die gleichen Konzepte des kartographischen Zeichensystems, z. B. Farbe, Form oder Signatur, zur Darstellung unterschiedlicher Informationen nutzen, besitzen thematische Karten eine Legende, welche die Verwendung des Zeichensystems erläutert. Slocum et al. beschreiben eine Legende als das Kartenelement, welches *alle* thematischen Gestaltungsmittel einer Karte definiert [Sl05, S.205].

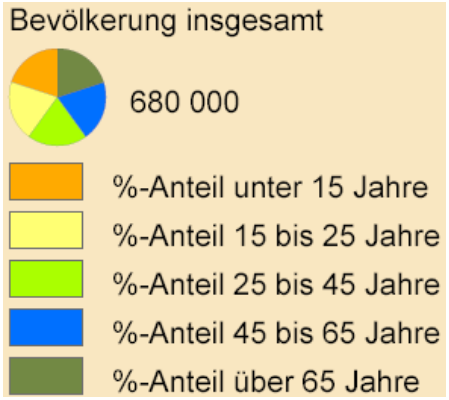


Abbildung 3.9.: Legende zur thematischen Karte aus Abbildung 3.7

Der Aufbau einer Legende für eine thematische Karte wird von Slocum et al. [Sl05] detailliert beschrieben. Beginnend bei der Position des Gestaltungsmittels zu dem erklärenden Text, über die Beschreibung von Gestaltungsvariablen bis hin zu speziellen Elementen von Legenden nach unterschiedlichen thematischen Kartentypen. Ein Beispiel einer Legende zeigt Abbildung 3.9, welche die Kreissektorendiagramme von Abbildung 3.7 erklärt.

3.3. Architekturdokumentation von Anwendungslandschaften in der Praxis

Nachdem in Abschnitt 3.1 die Grundlagen der Architekturdokumentation eingeführt wurden und Abschnitt 3.2 das im Folgenden verwendete Vokabular zur Dokumentation der Darstellung aufgebaut hat, werden im Weiteren verschiedene Modelle zur Architekturdokumentation von Anwendungslandschaften in der Praxis vorgestellt. Eine Kategorisierung der einzelnen Darstellungen folgt in Abschnitt 3.4.

Die Darstellungen zur Architekturdokumentation von Anwendungslandschaften werden in der Praxis und in der Forschung unterschiedlich bezeichnet. Beispiele für die verwendeten Begriffe sind *Bebauungsplan*, *Softwarelandkarte*, *Systemlandkarte*, *Anwendungslandkarte* und *IT-Landkarte*. Matthes und Wittenburg [MW04a] haben den Begriff der Softwarekarte, der unabhängig von einer mit der Darstellung verfolgten Zielsetzung ist, wie folgt eingeführt:

> **Softwarekarte :** Eine Softwarekarte (engl. *Software Map*) ist eine graphische Repräsentation der Anwendungslandschaft oder von Ausschnitten dieser. Eine Softwarekarte setzt sich zusammen aus einem Kartengrund und den auf dem Kartengrund aufbauenden Schichten, die verschiedene Merkmale visualisieren.

3.3.1. Softwarekarten in der Praxis

Die Abbildungen in den folgenden Abschnitten wurden von Industriepartnern des Forschungsprojektes Softwarekartographie zur Verfügung gestellt. Da die einzelnen Abbildungen vertrauliche Informationen enthalten, erscheinen diese jeweils in einer unleserlichen, stark verkleinerten Abbildung und einer anonymisierten Variante. Die Originale werden entweder in ausgedruckter Form, vorwiegend im Format DIN A0, in Präsentationen oder in Intranets verwendet.

Die anonymisierten Abbildungen beruhen auf den Daten eines fiktiven Kaufhauses *SoCaStore*[9] mit mehreren Standorten und Filialen. Das Kaufhaus SoCaStore entspricht der Datenbasis der Studie über Werkzeuge zum Management von Unternehmensarchitekturen [se05a].

Die Beschreibungen der Abbildungen in den folgenden Abschnitten erfolgt ausschließlich natürlichsprachlich. Dabei werden die in Abschnitt 3.2 eingeführten Konzepte Gestaltungsmittel, -variabel, -regel etc. verwendet. Eine semi-formale Beschreibung erfolgt zusammen mit einer Kategorisierung der Softwarekarten in Abschnitt 3.4.

Die in den folgenden Abschnitten vorgestellten Beispiele für Softwarekarten finden sich in mehreren Unternehmen. Da die Zielsetzung darin besteht, Modelle zum Management von Anwendungslandschaften zu entwickeln, wird eine Darstellung, die in der gleichen Art und Weise mehrfach verwendet wird, nur einmal eingeführt.

3.3.1.1. Softwarekarten eines Versicherungsunternehmens (Beispiel 1)

Ziel der Softwarekarte in Abbildung 3.10 (Originalform) bzw. Abbildung 3.11 (anonymisierte Form) ist die Visualisierung aller Anwendungssysteme einer Regionalgesellschaft eines Versicherungskonzerns, die Zuordnung der Systeme zu Funktionsbereichen und die Darstellung der Kommunikationsbeziehungen zwischen diesen Systemen. Das Unternehmen betreibt ca. 150 Anwendungssysteme in der betroffenen Regionalgesellschaft.

Bei der Softwarekarte handelt es sich um eine Darstellung des Ist-Zustands, ohne die Visualisierung von Plan- oder Soll-Zuständen, wobei der Zeitpunkt des Ist-Zustandes in der Karte angegeben wird. Die Darstellung hat den Zweck, Transparenz über die Anwendungslandschaft hinsichtlich der vorhandenen Anwendungssysteme, ihrer Zuordnung zu Bereichen und den Kommunikationsbeziehungen zwischen den Anwendungssystemen zu schaffen.

Die Anordnung der Gestaltungsmittelinstanzen auf der Karte erfolgt anhand einer für das Unternehmen individuellen logischen Gruppierung (*Subsidiaries*, *Customer Management*, *Warehouse* etc.) mittels Rechtecken. Die Gruppierung erfolgt in diesem Fall in zwei Ebenen, wobei die erste Ebene eine Aufteilung nach funktionalen Bereichen ist, dargestellt durch graue Rechtecke. Diese Gruppierung wird auf der zweiten Ebene durch Untergruppen verfeinert, dargestellt durch farbige Rechtecke.

[9] SoCaStore - Software Cartography Store

Abbildung 3.10.: Softwarekarte eines Versicherungsunternehmens, Beispiel 1, im Original Format DIN A0

Abbildung 3.11.: Softwarekarte eines Versicherungsunternehmens, Beispiel 1, anonymisierte Form

Im Original werden beispielsweise auf der ersten Ebene die Bereiche *Sparte*, *Kundenmanagement*, *Zugriffskanal* oder *Business Administration* unterschieden. Die zweite Ebene im Bereich *Sparte* unterteilt dann in die Versicherungssparten, beispielsweise *Leben*, *Sach*, *Unfall* etc.

Die farbliche Unterscheidung in dieser Softwarekarte ist im Original getrieben durch einen Farbcode, der den einzelnen Gruppierungen bestimmte Farben zuweist, die bereits in anderen Darstellungen des Unternehmens verwendet werden.

Anwendungssysteme werden den Gruppen ausschließlich in der zweiten Ebene zugeordnet, wobei eine Zuordnung einer Beziehung der Form *ist funktional zugeordnet* entspricht. Die Beziehung lässt es zu, dass ein Anwendungssystem in mehreren Untergruppen vertreten ist (in Abbildung 3.11 bspw. das Anwendungssystem *MoTr-D*). Ein Anwendungssystem wird durch ein Rechteck mit einer abgesetzten Kopfzeile dargestellt, wobei in dieser Kopfzeile eine eindeutige Systemnummer angegeben ist. Die Beschriftung des Rechtecks enthält den (abgekürzten) Namen des Anwendungssystems.

Die Gestaltungsregel, die bei der Positionierung der Rechtecke für die Bereiche und für die Anwendungssysteme ihre Verwendung findet, wird als *Nesting* (deutsch Verschachtelung) bezeichnet und verlangt bei ihrer Anwendung eine Positionierung der Gestaltungsmittelinstanz vollständig innerhalb einer anderen Gestaltungsmittelinstanz.

Kommunizieren zwei Systeme über eine Schnittstelle miteinander, so wird dies mit zwei Pfeilen als Gestaltungsmittelinstanzen visualisiert. Der Server erhält einen eingehenden Pfeil, der mit dem Namen des Clients beschriftet ist; der Client entsprechend einen ausgehenden Pfeil, beschriftet mit dem Namen des Servers. Eine Unterscheidung zwischen Daten- oder Kontrollfluss wird in dieser Softwarekarte nicht durchgeführt.

Die für die Pfeile angewendete Gestaltungsregel, die verlangt, dass ein Pfeil sich an ein Rechteck *anhängt*, wird als *Attachment* (deutsch Anhängen) bezeichnet und fordert, dass für zwei Gestaltungsmittelinstanzen ein gemeinsamer Punkt auf der Softwarekarte existiert.

Die Elemente auf dieser Softwarekarte erfahren keine spezifische Anordnung entlang einer bestimmten Achse. Die Positionierung der Gruppen auf der Softwarekarte, der Untergruppen innerhalb der Gruppen und der Anwendungssysteme in den Untergruppen wird platzoptimierend vorgenommen. Des Weiteren besitzt die Größe der Rechtecke für Anwendungssysteme keine eigene Semantik und wird derart gewählt, dass alle eingehenden bzw. ausgehenden Pfeile an das Rechteck angehängt werden können.

Die Softwarekarte in Abbildung 3.10 wurde mittels des Werkzeugs Microsoft Visio vollständig manuell erstellt. In der Originaldarstellung nicht vorhanden ist die Legende in Abbildung 3.11.

3.3.1.2. Softwarekarte eines Versicherungsunternehmens (Beispiel 2)

Der Aufbau und die Zielsetzung der Softwarekarte in Abbildung 3.12 (Originalform) bzw. Abbildung 3.13 (anonymisierte Form) ist eng verwandt mit dem Beispiel aus Ab-

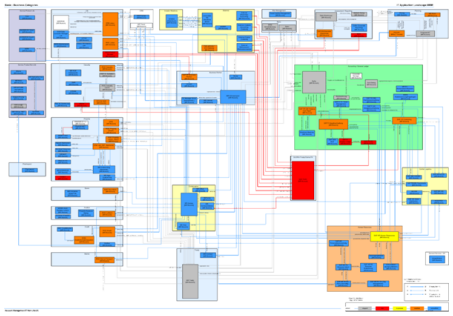


Abbildung 3.12.: Softwarekarte eines Versicherungsunternehmens, Beispiel 2, im Original Format DIN A0

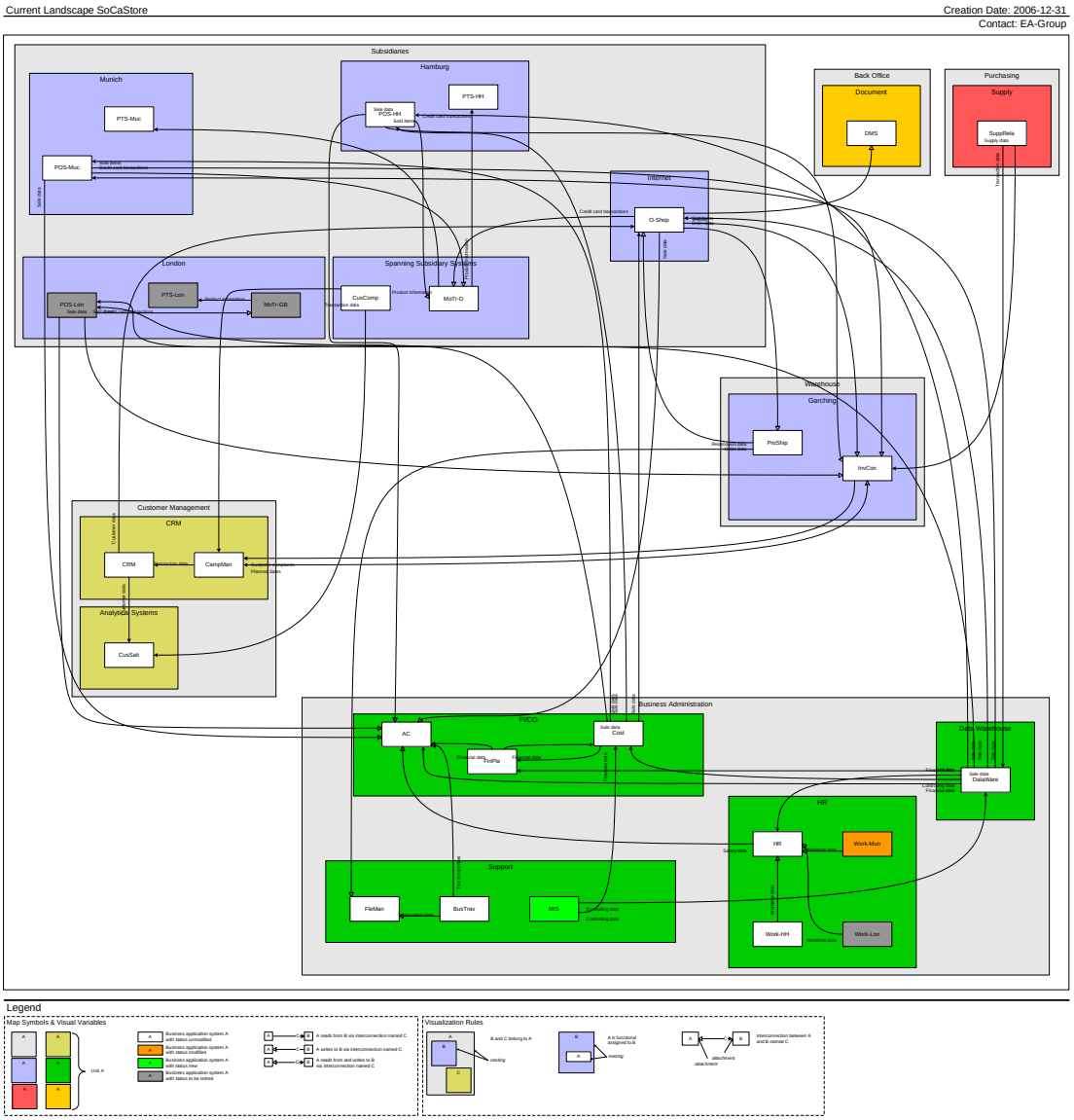


Abbildung 3.13.: Softwarekarte eines Versicherungsunternehmens, Beispiel 2, anonymisierte Form

schnitt 3.3.1.1. Auch hier bilden verschachtelte Rechtecke die Softwarekarte und die Zielsetzung besteht ebenfalls in der Schaffung von Transparenz über die Anwendungslandschaft. Im Unterschied zu dem Beispiel aus Abschnitt 3.3.1.1 werden zusätzlich unterschiedliche Status für Anwendungssysteme angegeben und die Kommunikationsbeziehungen detaillierter dargestellt. Die Softwarekarte visualisiert im Original ca. 160 Anwendungssysteme, die am Hauptstandort des Unternehmens betrieben und weltweit genutzt werden.

Die Instanzen des Gestaltungsmittels Rechteck werden in dieser Karte anhand von unternehmensspezifischen Bereichen gruppiert. Zusätzlich werden Instanzen des Gestaltungsmittels Rechteck genutzt, um sowohl Bereiche als auch Anwendungssysteme darzustellen, wobei die Beschriftung für Bereiche vertikal oben und für Anwendungssysteme vertikal zentriert ausgerichtet wird.

Die Farbcodierung der Rechtecke ist im Original für die Bereiche unternehmensspezifisch, und verwandte Bereiche erhalten die gleiche Hintergrundfarbe. Die Farbcodierung der Rechtecke für Anwendungssysteme gibt den Status des Anwendungssystems an. Beispielsweise werden Anwendungssysteme, die zum auf der Karte angegebenen Zeitpunkt abgeschaltet wurden, mit einem grauen Hintergrund, neue Systeme mit einem roten Hintergrund und Anwendungssysteme, die seit dem letzten Betrachtungszeitpunkt verändert wurden, mit einem orangefarbenen Hintergrund dargestellt. Durch das quartalsweise Aktualisieren dieser Softwarekarte ergeben sich die Betrachtungszeitpunkte.

Die Pfeile, die als Gestaltungsmittelinstanzen Kommunikationsbeziehungen darstellen, verbinden in der Softwarekarte die betroffenen Anwendungssysteme und werden mit einem Namen für die Kommunikationsbeziehung als Beschriftung versehen. Im Gegensatz zu der Softwarekarte in Abschnitt 3.3.1.1 verbinden Pfeile die Anwendungssysteme direkt und stellen ausschließlich Datenflüsse dar, wobei zwischen lesenden und schreibenden Zugriffen (siehe Legende in Abbildung 3.13) unterschieden wird. Auch hier wird die Gestaltungsregel *Attachment* verwendet, um die Pfeile an die jeweiligen Rechtecke anzubinden, wobei die jeweilige Gestaltungsmittelinstanz für einen Pfeil an jeweils zwei Gestaltungsmittelinstanzen vom Typ Rechteck angehängt werden.

Die Positionierung der Rechtecke für Bereiche und für Anwendungssysteme erfolgt mittels mehrerer Gestaltungsregeln, wobei die Reihenfolge in der Aufzählung die Priorisierung der Gestaltungsregeln angibt:

1. Positioniere Rechtecke für Anwendungssysteme in den Rechtecken der Bereiche mit denen eine Beziehung *ist funktional zugeordnet* besteht (Gestaltungsregel *Nesting*).
2. Positioniere verwandte Bereiche möglichst nahe beieinander.
3. Positioniere die Rechtecke für die Bereiche und die Rechtecke für Anwendungssysteme innerhalb der Bereiche derart, dass möglichst wenige Überschneidungen der Pfeile für Kommunikationsbeziehungen entstehen.

Bei den obigen Gestaltungsregeln ist eine wesentliche Unterscheidung zwischen zwei Arten von Regeln zu erkennen. Die erste Regel verlangt eine Positionierung in der Form,

dass sie eine bestimmte Position relativ zu einer anderen Gestaltungsmittelinstanz verlangt, die nicht verletzt werden darf. Diese Art von Regel wird im Folgenden als *Muss-Gestaltungsregel* bezeichnet:

> **Muss-Gestaltungsregel :** Eine Muss-Gestaltungsregel (engl. *Constraint Visualization Rule*) ist eine Gestaltungsregel, deren Instanzen eine graphische Beziehung zwischen Gestaltungsmittelinstanzen beschreiben, die erfüllt sein muss, um eine korrekte Darstellung zu erhalten.

Die Regeln 2 und 3 verfolgen eine Positionierung, die *möglichst* genau eingehalten werden soll. Diese Art von Gestaltungsregeln werden als *Ziel-Gestaltungsregel* bezeichnet:

> **Ziel-Gestaltungsregel :** Eine Ziel-Gestaltungsregel (engl. *Target Visualization Rule*) ist eine Gestaltungsregel, deren Instanzen eine graphische Beziehung zwischen Gestaltungsmittelinstanzen beschreiben, die im Sinne einer Zielfunktion möglichst *gut* erfüllt werden soll, um eine gewünschte Darstellung zu erhalten.

Die Zielfunktion der 2. Regel kann beispielsweise als Optimum einen Abstand von 10 Pixeln auf einer Softwarekarte verlangen oder fordern, dass die Rechtecke direkt benachbart sind, also kein anderes Rechteck zwischen ihnen liegen darf. Die einzelnen Ziel-Gestaltungsregeln werden eigenständig in Abschnitt 4.3 eingeführt.

Als Hilfsmittel zur Erstellung der Softwarekarte in Abbildung 3.12 dient das Werkzeug *VisualizeIT* [Vö03], welches von der Firma sd&m entwickelt und durch das nutzende Unternehmen angepasst wurde. Das Werkzeug verwendet zur Datenhaltung und für das Reporting Microsoft Access und generiert aus den Daten die Softwarekarte in Microsoft Visio. Beim ersten Generieren der Softwarekarte wird keine Optimierung hinsichtlich der Position der Rechtecke durchgeführt, die verwandte Bereiche nahe beieinander positioniert oder die Anzahl von Überschneidungen der Pfeile minimiert. Das manuelle Nachpositionieren nach dem ersten Generieren wird jedoch bei folgenden Generierungsläufen berücksichtigt. Lediglich neue Bereiche werden am unteren Ende der Karte und neue Anwendungssysteme am unteren Ende eines Bereiches visualisiert. Die Karte ist somit semi-automatisch generiert.

Die Legende der Originaldarstellung enthält Angaben zur Hintergrundfarbe der Anwendungssysteme und zu den Arten von Kommunikationsbeziehungen. Alle weiteren Informationen in der Legende von Abbildung 3.13 wurden hinzugefügt.

3.3.1.3. Softwarekarten eines Logistikdienstleisters

Die Softwarekarte in Abbildung 3.14 (anonymisierte Form in Abbildung 3.15) soll zum einen Transparenz der Anwendungslandschaft schaffen und zum anderen IT-Projekte mit den betroffenen Systemen und deren Entwicklungsstand bzw. Projektfortschritt visualisieren. Der Nutzer dieser Anwendungslandschaft ist ein Unternehmensbereich eines Logistikkonzerns, der ca. 150 Anwendungssysteme verwendet.

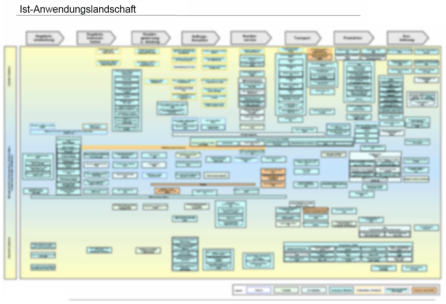


Abbildung 3.14.: Softwarekarte eines Logistikdienstleisters, im Original Format DIN A0

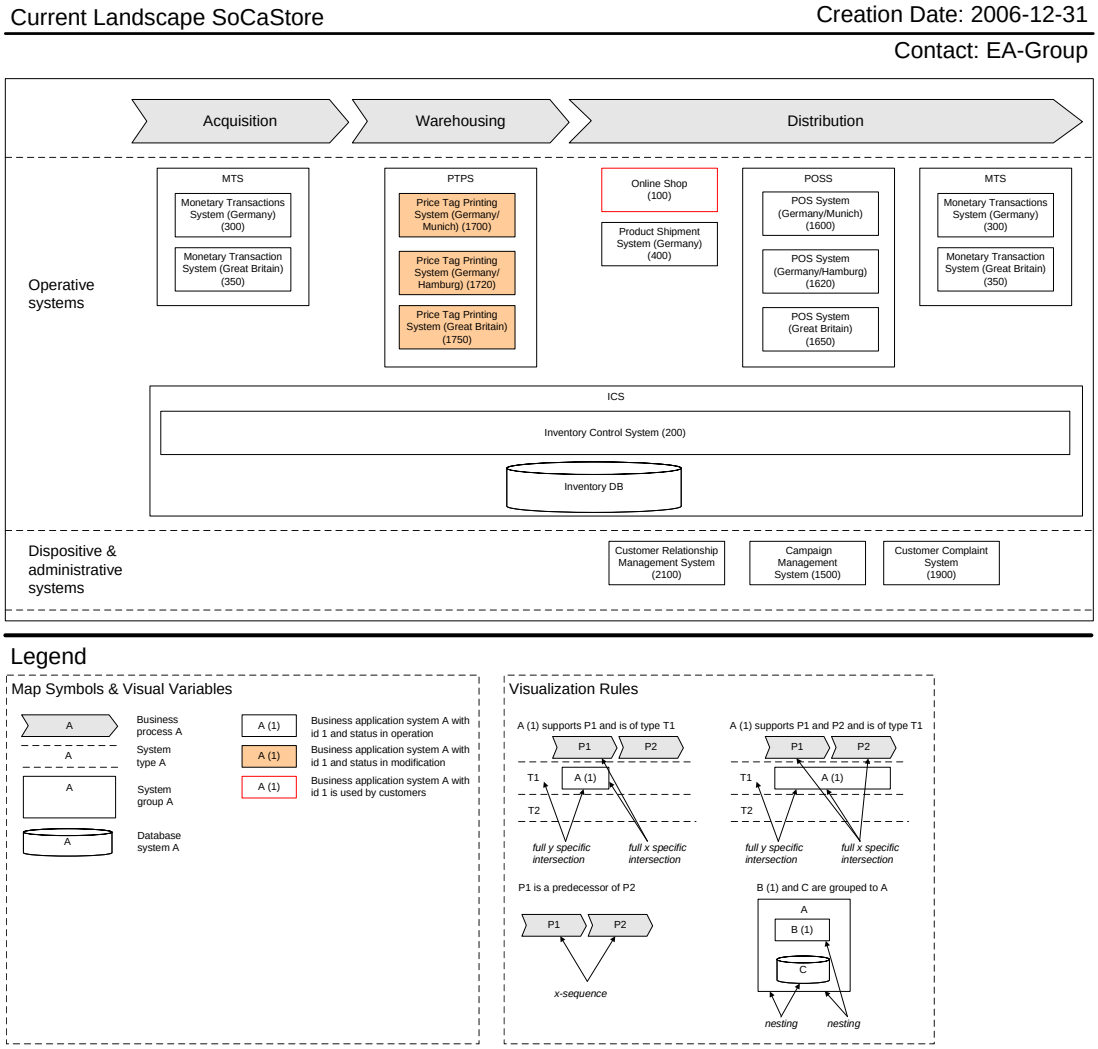


Abbildung 3.15.: Softwarekarte eines Logistikdienstleisters, anonymisierte Form

Im Gegensatz zu den Softwarekarten in den Abschnitten 3.3.1.1 und 3.3.1.2 wird bei dieser Softwarekarte eine Anordnung entlang der x- und y-Achsen vorgenommen. Die x-Achse enthält die Prozessschritte der Wertschöpfungskette des Unternehmens und die y-Achse unterscheidet die Systeme nach Systemtypen wie beispielsweise *dispositiv* oder *operativ*. Die Prozessschritte auf der x-Achse sind durch Instanzen des Gestaltungsmittels *Chevron*[10] dargestellt, die Unterscheidung auf der y-Achse erfolgt durch eine Beschriftung der Bereiche.

Hieraus ergeben sich zwei neue Muss-Gestaltungsregeln, die es ermöglichen, Elemente in einer bestimmten Reihenfolge zu positionieren. Diese Muss-Gestaltungsregeln werden als *X-Sequence* bzw. *Y-Sequence* (deutsch x-Sequenz bzw. y-Sequenz) bezeichnet. Sie ermöglichen es, mehrere Instanzen von Gestaltungsmitteln über eine Gestaltungsregelinstanz in Beziehung zu setzen. Diese Gestaltungsregeln erlauben es, dass bei einer Instantiierung die in Beziehung gesetzten Gestaltungsmittelinstanzen in einer geordneten Reihung platziert werden, so dass eine Reihenfolge der Elemente bezüglich der x- oder y-Achse entsteht. Die Muss-Regel *X-Sequence* bzw. *Y-Sequence* verlangt zusätzlich, dass die in Beziehung gesetzten Gestaltungsmittelinstanzen die gleiche Position bezüglich der y- bzw. x-Achse haben.

Innerhalb des durch die Prozessschritte und Systemtypen gebildeten Koordinatensystems werden verschiedene Arten von IT-Systemen visualisiert, die von dem Nutzer wie folgt verwendet werden:

Anwendungssystem: Ein Anwendungssystem wird durch das Gestaltungsmittel Rechteck visualisiert.

Teil-Anwendungssystem: Ein Teil-Anwendungssystem, welches als Teil keine funktionsfähige Schnittstelle gegenüber einem Anwender zur Verfügung stellt, wird mittels des Gestaltungsmittels Rechteck visualisiert.

Datenbanksystem: Ein Datenbanksystem ist eine Sammlung von Daten, die zueinander in Beziehung stehen, systematisch oder methodisch angeordnet und einzeln mit Hilfe elektronischer Mittel zugänglich sind. Ein Datenbanksystem wird mittels des Gestaltungsmittels *Zylinder* visualisiert.

Systemgruppe: Eine Systemgruppe ist eine logische Zusammenfassung physisch voneinander abhängiger Komponenten, die mittels Funktion und Daten die Geschäftsprozesse oder Teile davon unterstützen. Zu einem System gehört eine integrierte Datenhaltung. Eine Systemgruppe wird mit dem Gestaltungsmittel Rechteck visualisiert.

Die Beschriftung der obigen vier Elemente erfolgt mit einem Kurznamen und einer Kurzbeschreibung[11]. Zusätzlich gilt, wenn der Name der Systemgruppe gleich dem Namen des Anwendungssystems und des Namens des Datenbanksystems ist, so wird nur ein Rechteck für die Systemgruppe verwendet. Des Weiteren muss für ein Teil-Anwendungssystem ein übergeordnetes Anwendungssystem visualisiert werden.

[10] Das Chevron (-s, -en) entspricht der geometrischen Form eines Fischsymbols.
[11] In der anonymisierten Darstellung in Abbildung 3.15 wird nur ein Name verwendet.

Auffällig bei den obigen Definitionen ist sowohl die Semantik als auch die Wahl der Symbolik. Die Unterscheidung, ob ein System ein Anwendungssystem, ein Teil-Anwendungssystem etc. ist, muss individuell getroffen werden und kann nicht eindeutig aus der Begriffsdefinition hergeleitet werden. Nichtsdestotrotz handelt es sich für den Nutzer um eine wesentliche Unterscheidung, welche in die Darstellung eingeflossen ist.

Hinsichtlich der Symbolik ist die häufige Verwendung von Rechtecken ohne zusätzliche Annotationen anzumerken, da aus der Darstellung nur durch zusätzliche Konventionen abgeleitet werden kann, ob es sich beispielsweise bei dem System *ICS* um eine Systemgruppe oder ein Anwendungssystem handelt. Die geringe Werkzeugunterstützung (die Originaldarstellung wurde mit Microsoft PowerPoint erstellt) erklärt, warum die Anzahl von Symbolen möglichst gering gehalten wurde. Die Erstellung und Pflege der Softwarekarte ist bereits mit großen Aufwand verbunden, so dass der Fokus auf Datenkonsistenz und eine einfache Symbolik gelegt wurde.

Die Symbole für die Systeme werden auf der Softwarekarte derart positioniert, dass die Geschäftsprozessunterstützung und der Typ von Anwendungssystemen erkennbar sind. Hierzu werden die Symbole an den Prozessen auf der x-Achse und an den Systemtypen auf der y-Achse geeignet ausgerichtet. Diese Art der Positionierung wird durch die Muss-Gestaltungsregeln *Full X-Specific Intersection* bzw. *Full Y-Specific Intersection* (deutsch vollständige x- bzw. y-spezifische Überschneidung) geregelt, wobei erstere beispielsweise verlangt, dass die x-Koordinaten des Rechtecks für das Anwendungssystem *Product Shipment System (Germany)* im Bereich der x-Koordinaten des Chevrons für den Geschäftsprozess *Distribution* liegen.

Eine Besonderheit stellt die Ausdehnung eines Rechtecks für ein Anwendungssystem über mehrere Prozessschritte dar, wobei die Ausdehnung bedeutet, dass ein einzelnes Anwendungssystem mehrere Prozessschritte unterstützt. Diese graphische Variation ist jedoch nur möglich, wenn die von dem Anwendungssystem unterstützten Prozessschritte direkt nebeneinander angeordnet sind. Ist dies nicht der Fall, so müssen die Rechtecke dupliziert werden. In der Softwarekarte in Abbildung 3.15 wird die horizontale Ausdehnung eines Rechtecks exemplarisch für das Anwendungssystem *Inventory Control System*, welches mehrere Geschäftsprozesse unterstützt, dargestellt.

Die Darstellungen für eine Geschäftsprozessunterstützung eines Anwendungssystems für zwei graphisch benachbarte Geschäftsprozesse durch ein Rechteck, das horizontal gedehnt ist, oder durch zwei Rechtecke, die nicht horizontal gedehnt sind, sind semantisch äquivalent. Die gedehnte Darstellung eines Rechtecks bedingt eine Erweiterung der Muss-Gestaltungsregel *Full X-Specific Intersection*, die verlangt, dass eine Instanz dieser Regel eine einzelne Gestaltungsmittelinstanz mit mehreren anderen Gestaltungsmittelinstanzen in Beziehung setzen darf. Die Umsetzung der Regel erzwingt in diesem Fall, dass sich ein Rechteck für ein Anwendungssystem über mehrere Chevren für Geschäftsprozesse erstreckt.

Des Weiteren wird die Muss-Gestaltungsregel *Nesting* verwendet, um eine Gruppierung von Anwendungssystemen und/oder Datenbanksystemen zu Systemgruppen zu erreichen.

Zur Darstellung verschiedener Status der Systeme, wie beispielsweise *System in Betrieb* oder *System wird geändert*, und weiterer Eigenschaften der Systeme, wie *System wird von Kunden benutzt*, werden in der Originaldarstellung verschiedene Farben und Schraffuren für Hintergründe und Farben für Rahmen der Elemente gewählt. In dem anonymisierten Beispiel in Abbildung 3.15 bedeutet ein roter Rahmen beispielsweise, dass das betreffende System direkt von Kunden bedient wird (siehe Legende).

3.3.1.4. Softwarekarten eines Konsumgüterherstellers

Ziel der Softwarekarte in Abbildung 3.16 (anonymisiert in Abbildung 3.17) ist wiederum die Schaffung von Transparenz über die Anwendungslandschaft. Die Softwarekarte wurde von einem Konsumgüterhersteller erstellt, der die Abbildung mit dem englischen Begriff *Quilt* (deutsch Flickenteppich) umschreibt.

Analog zu der Softwarekarte in Abschnitt 3.3.1.3 werden die Bezugselemente für die Anwendungssysteme am äußeren Rand der Softwarekarten positioniert und bilden eine x- und y-Achse. Auf der x-Achse wird auf zwei Ebenen gruppiert, wobei die obere Ebene durch die Organisationseinheiten und die untere Ebene durch zugeordnete Funktionsbereiche gebildet wird. Auf der y-Achse finden sich zwei Ebenen, welche von den Vertriebsbereichen und den Produktgruppen gebildet werden, wobei die Vertriebsbereiche auf der linken und die Produktgruppen auf der rechten Seite angeordnet werden. In der anonymisierten Form wurden die Organisationseinheiten (*Headquarter*, *Subsidiaries* und *Warehouse*) und die Funktionsbereiche (*Back Office*, *Purchasing* etc.) auf der x-Achse und die Vertriebsbereiche (*Munich*, *Hamburg*, *London* und *Online*) auf der y-Achse positioniert.

Die Visualisierung der Organisationseinheiten, Funktionsbereiche und Vertriebsbereiche erfolgt in der Originaldarstellung mittels eines Textfeldes ohne ein zusätzliches Symbol. In der anonymisierten Form wird die Beschriftung der einzelnen Spalten und Zeilen mittels des Gestaltungsmittels Rechteck, welches ein Textfeld beinhaltet, mit unterschiedlichen Hintergrundfarben dargestellt, um die Unterscheidung von Organisationseinheiten, Funktionsbereichen und Vertriebsbereichen hervorzuheben.

Der Aufbau der Softwarekarte ist des Weiteren vergleichbar mit einer Matrix, wobei dies nur einen Spezialfall der Softwarekarte mit x- und y-Achsen darstellt und beide Achsen diskrete Werte visualisieren[12].

Die Anwendungssysteme werden analog zu der Softwarekarte in Abschnitt 3.3.1.3 positioniert und durch ein Rechteck visualisiert. Die Hintergrundfarbe wird genutzt, um Anwendungssysteme zu Gruppen zusammenzufassen. Im Original wurden beispielsweise SAP-Systeme mit der gleichen Hintergrundfarbe dargestellt, in der anonymisierten Form wurden Standardsoftwaresysteme mittels eines gleichfarbigen Hintergrunds visualisiert.

Eine weitere Gemeinsamkeit mit der Softwarekarte in Abschnitt 3.3.1.3 ist das Ausdehnen von Rechtecken für Anwendungssysteme, welches in diesem Fall vertikal erfolgt,

[12] Die genaue Unterscheidung der Softwarekartentypen folgt in Abschnitt 3.4.1

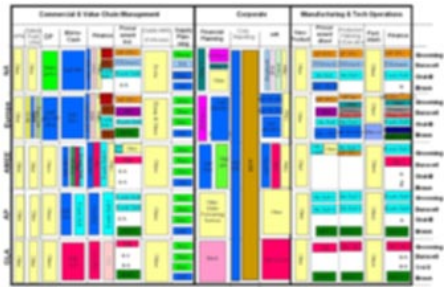


Abbildung 3.16.: Softwarekarte eines Logistikdienstleisters, im Original Format DIN A0

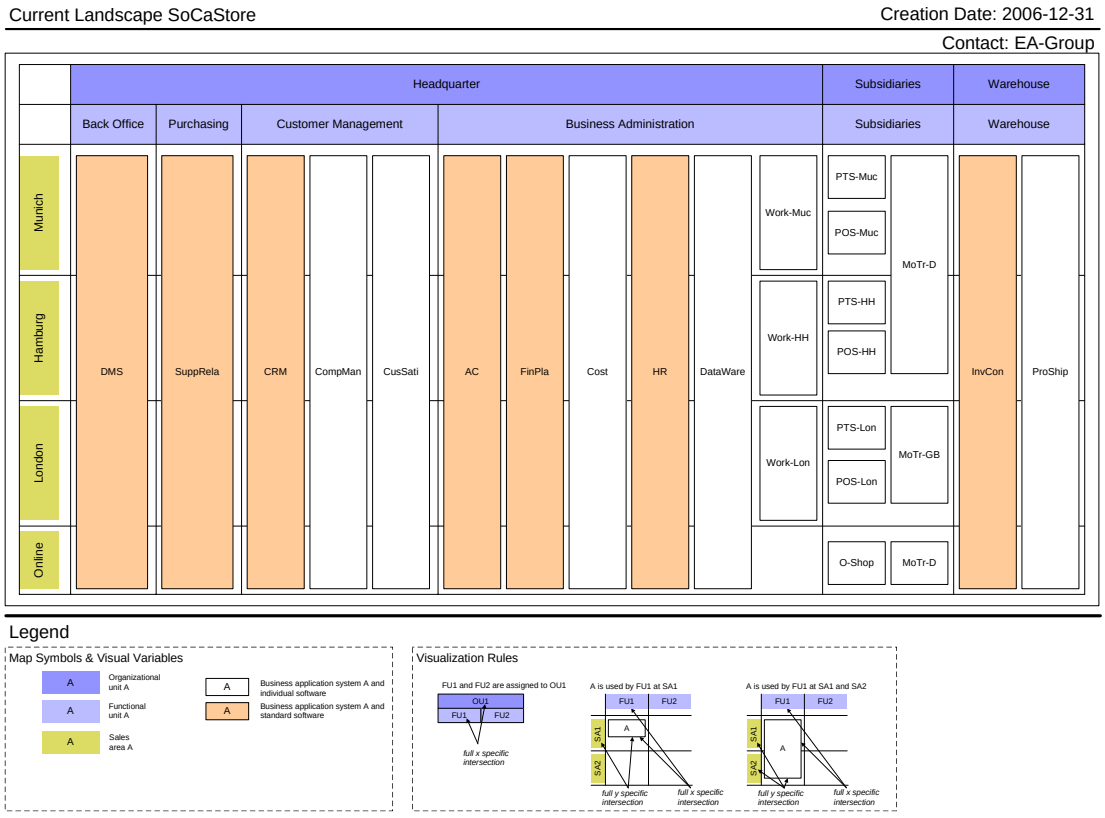


Abbildung 3.17.: Softwarekarte eines Logistikdienstleisters, anonymisierte Form

wenn mehrere Produktgruppen innerhalb des gleichen Funktionsbereiches dasselbe Anwendungssystem nutzen. Die verwendeten Gestaltungsregeln sind somit ebenso *Full X-Specific Intersection* bzw. *Full Y-Specific Intersection*.

Das Original dieser Softwarekarte wurde mittels Microsoft PowerPoint erstellt.

3.3.1.5. Softwarekarten eines Automobilherstellers

Im Gegensatz zu den bisherigen Beispielen aus den Abschnitten 3.3.1.1 bis 3.3.1.4 zeigen die Abbildungen 3.18, 3.20 und 3.19 (anonymisiert in den Abbildungen 3.21, 3.23 und 3.22) nicht nur den Status der Anwendungslandschaft zu einem bestimmten Zeitpunkt, sondern drei verschiedene Versionen bzw. Zustände. Der Automobilkonzern, der diese Darstellungen nutzt, unterscheidet zwischen so genannten Ist-, Plan- und Soll-Landschaften:

- Der Ist-Zustand spiegelt den Status quo der Anwendungslandschaft wider.
- Der Plan-Zustand ist ein Zustand der Anwendungslandschaft in der Zukunft, der sich durch in der Zukunft abgeschlossene und laufende Projekte zu einem bestimmten Zeitpunkt ergibt.
- Der Soll-Zustand ist eine langfristige Vision, die einer Zielvorstellung entspricht.

Diese Unterscheidung wurde bereits in Abschnitt 2.1.7 eingeführt und ermöglicht es beispielsweise die Zielerreichung zu steuern und nachzuvollziehen.

Der Aufbau der einzelnen Darstellungen ist vergleichbar mit dem der Softwarekarte in Abschnitt 3.3.1.3. Es werden ebenso Geschäftsprozesse entlang der x-Achse positioniert, wobei die Originale der Softwarekarten in diesem Abschnitt mehrere Prozessebenen visualisieren und die einzelnen Geschäftsprozesse in einer Sub- bzw. Super-Beziehung miteinander stehen. Die Gestaltungsregeln die hierzu verwendet werden, sind zum einen die Muss-Gestaltungsregel *X-Sequence*, um die Prozesse entlang der x-Achse zu positionieren, und zum anderen die Muss-Gestaltungsregel *Full X-Specific Intersection*, um die Sub-Prozesse unterhalb der Super-Prozesse darzustellen.

Die y-Achse wird durch Organisationseinheiten gebildet, so dass beispielsweise verschiedene Vertriebskanäle oder auch Werke des Automobilherstellers, die sich auch in der Organisationsstruktur wiederfinden, auf diese Art und Weise abgebildet werden. Die einzelnen Softwarekarten in diesem Abschnitt zeigen hierbei nicht die gesamte Anwendungslandschaft des Automobilherstellers, der ca. 2.500 Anwendungssysteme nutzt und betreibt, sondern jeweils nur bestimmte Ausschnitte für ausgewählte Geschäftsprozesse (x-Achse) und Organisationseinheiten (y-Achse).

Die Visualisierung einer Geschäftsprozessunterstützung für eine bestimmte Organisationseinheit erfolgt, indem ein Rechteck als Gestaltungsmittelinstanz mit dem Namen des Anwendungssystems an dem Geschäftsprozess und der Organisationseinheit ausgerichtet wird. Analog zu den Softwarekarten aus den Abschnitten 3.3.1.3 und 3.3.1.4 wird ebenso mit einer Ausdehnung der Rechtecke für Anwendungssysteme gearbeitet. Eine horizontale Ausdehnung eines Rechtecks bedeutet, dass mehrere Geschäftsprozesse von diesem

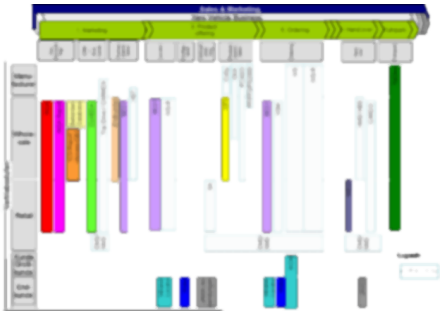


Abbildung 3.18.: Softwarekarte eines Automobilherstellers, Ist-Zustand, im Original Format DIN A0

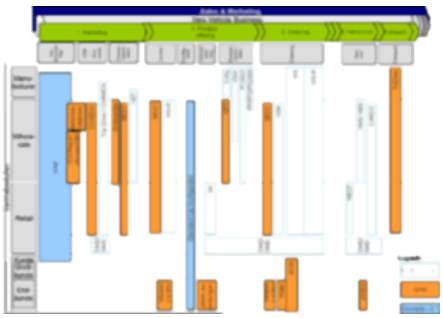


Abbildung 3.19.: Softwarekarte eines Automobilherstellers, Plan-Zustand, im Original Format DIN A0

Abbildung 3.20.: Softwarekarte eines Automobilherstellers, Soll-Zustand, im Original Format DIN A0

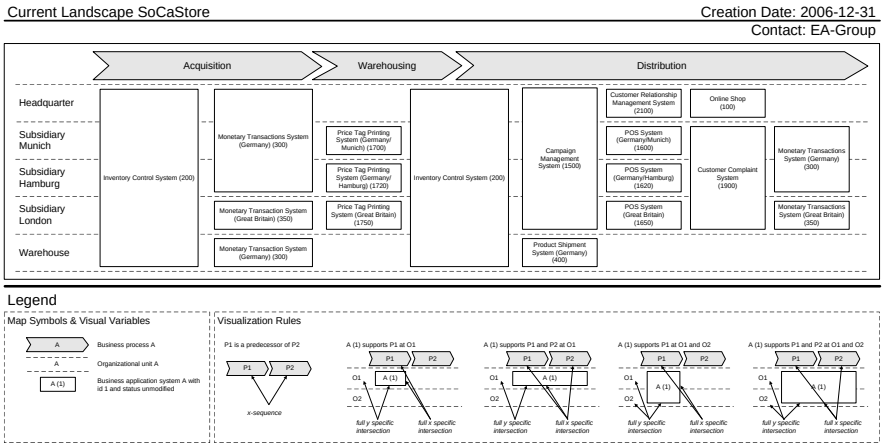


Abbildung 3.21.: Softwarekarte eines Automobilherstellers, Ist-Zustand, anonymisierte Form

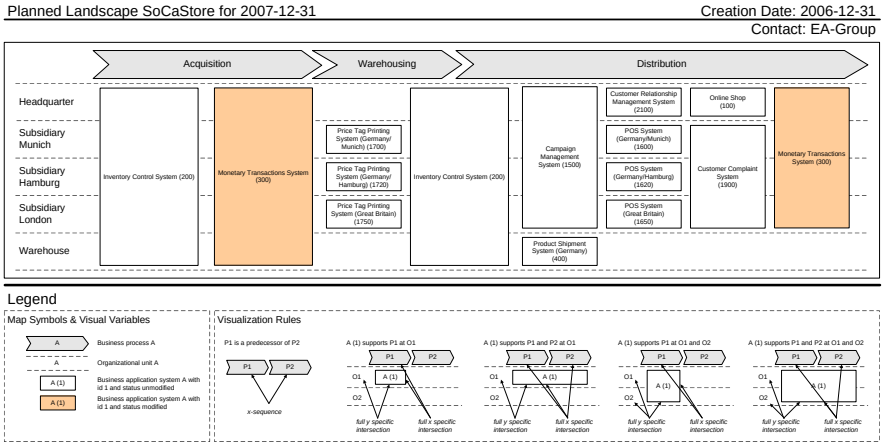


Abbildung 3.22.: Softwarekarte eines Automobilherstellers, Plan-Zustand, anonymisierte Form

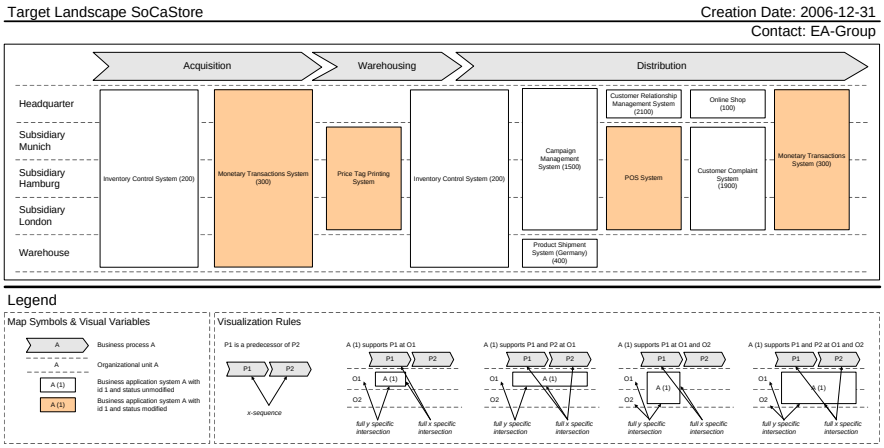


Abbildung 3.23.: Softwarekarte eines Automobilherstellers, Soll-Zustand, anonymisierte Form

System unterstützt werden, und eine vertikale Ausdehnung, dass mehrere Organisationseinheiten das gleiche Anwendungssystem nutzen. Die hierzu verwendeten Gestaltungsregeln sind die bereits eingeführten Regeln *Full X-Specific Intersection* und/oder *Full Y-Specific Intersection*.

Da die Softwarekarten im Original für ein bestimmtest Projekt angefertigt wurden, wird die Hintergrundfarbe genutzt, um Projektinformationen zu visualisieren. Eine weiße Hintergrundfarbe bedeutet, dass das visualisierte Anwendungssystem im Rahmen des Projektes nicht verändert wird. Die anderen Hintergrundfarben der Rechtecke in den Softwarekarten für die Ist- und Soll-Landschaften kennzeichnen, dass das entsprechende Anwendungssystem im Projekt verändert wird.

In der Plan-Landschaft wird über die Hintergrundfarben der Rechtecke zusätzlich die Leistungsstufe visualisiert, die bis zum Zeitpunkt der Plan-Landschaft umgesetzt werden soll oder welche Änderungen für spätere Leistungsstufen vorgesehen sind. In den anonymisierten Softwarekarten werden die Hintergrundfarben in den Soll- und Plan-Landschaften genutzt, um Veränderungen bezüglich der Ist-Landschaft darzustellen.

Die Erstellung der Softwarekarten erfolgt bei dem Automobilhersteller je nach Organisationseinheit mittels unterschiedlicher Werkzeuge. Die Beispiele in den Abbildungen 3.18, 3.20 und 3.19 wurden mittels Microsoft PowerPoint erstellt, andere Organisationseinheiten nutzen ein selbst erstelltes Werkzeug auf Basis von Microsoft Access und Microsoft Visio, welches eine vergleichbare Funktionalität zu dem in Abschnitt 3.3.1.2 vorgestellten Werkzeug *VisualizeIT* besitzt.

3.3.1.6. Weitere Softwarekarten

Die in den Abschnitten 3.3.1.1 bis 3.3.1.5 vorgestellten Softwarekarten sind die prominentesten Darstellungsformen für Anwendungslandschaften, die bei den untersuchten Industrieunternehmen zum Einsatz kommen. Die einzelnen Darstellungen finden sich nicht nur in den betroffenden Unternehmen bzw. Branchen, sondern werden ebenso in anderen Unternehmen und Branchen eingesetzt. Die Softwarekarte in Abbildung 3.13 wird beispielsweise ebenso von einem IT-Dienstleister für Banken verwendet und die Darstellungen in Abbildung 3.21 ebenso von einem Elektronikkonzern [Ro05a].

Zwei weitere, bisher nicht vorgestellte Formen von Softwarekarten sind Karten zur Analyse des Lebenszyklusses von Anwendungssystemen und zur Analyse von Beziehungen zwischen Elementen der Unternehmensarchitektur.

Die Softwarekarte in Abbildung 3.24 stellt den Lebenszyklus eines Anwendungssystems bzw. einer Anwendungssystemversion mit verschiedenen Status dar. Hierzu wird analog zu bereits eingeführten Darstellungen (bspw. in Abschnitten 3.3.1.4) ein Koordinatensystem aufgebaut, wobei die x-Achse durch Zeitintervalle und die y-Achse durch Anwendungssysteme und Anwendungssystemversionen gebildet werden. Die hierbei zum Einsatz kommenden Gestaltungsregeln sind wiederum die *X-Sequence* und *Full X-Specific Intersection* für die Zeitintervalle und die Regeln *Y-Sequence* und *Full Y-Specific Intersection* für die Anwendungssysteme. Um die äquidistante Darstellung der Zeitreihe zu

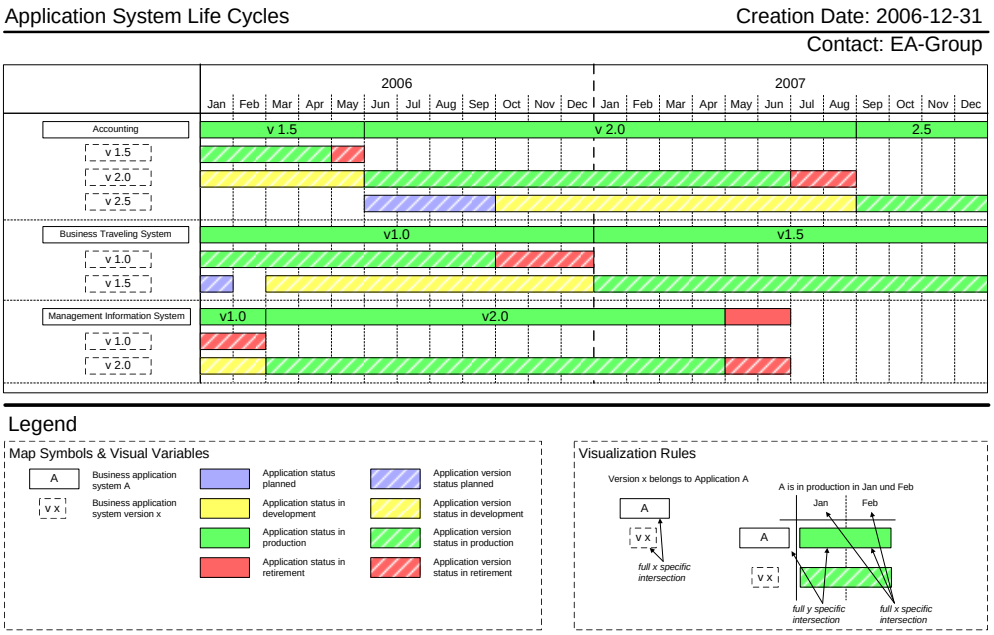


Abbildung 3.24.: Softwarekarte zur Analyse des Lebenszyklus von Anwendungssystemen

erhalten, werden die Gestaltungsmittelinstanzen zur Darstellung der Monate auf der x-Achse geeignet in Beziehung gesetzt, um eine einheitliche Höhe und Breite zu erhalten. Die verwendete Gestaltungsregel wird als *Identity of Projection* (deutsch Identität unter Projektion) bezeichnet und kann beliebig viele Gestaltungsmittelinstanzen bezüglich einer Gestaltungsvariable miteinander in Beziehung setzen. Diese Muss-Gestaltungsregel verlangt, dass die gewählten Gestaltungsvariablen mehrerer Gestaltungsmittelinstanzen den gleichen Wert besitzen. In der Softwarekarte in Abbildung 3.24 werden beispielsweise die Gestaltungsvariablen für die Breite der Gestaltungsmittelinstanzen, welche die Monate visualisieren, derart in Beziehung gesetzt.

Innerhalb des Koordinatensystems werden die Status der Anwendungssysteme bzw. Anwendungssystemversionen durch Instanzen des Gestaltungsmittels Rechteck mit einem farblichen Hintergrund dargestellt (siehe Legende), so dass ein Aufbau entsteht, der mit einem Gantt-Diagramm vergleichbar ist. Die Darstellung ermöglicht somit die Analyse des Lebenszyklusses von Anwendungssystemen. Neue Projekte können beispielsweise während der Planungsphase erkennen, welche Anwendungssystemversion zu einem bestimmten Zeitpunkt in Betrieb und welche neuen Versionen in Vorbereitung sind.

Eine hinsichtlich des Aufbaus ähnliche Darstellung wird von einen Finanzdienstleister genutzt, um Projektabhängigkeiten zwischen fachlichen IT-Projekten und IT-Plattformprojekten zu identifizieren [Br05a]. Ein fachliches IT-Projekt nutzt bei der (Neu-) Entwicklung eines Anwendungssystems eine oder mehrere IT-Plattformen, wobei diese eine sinnvolle Kombination mehrerer Komponenten (bspw. Betriebssystems, Hardwaresystems und Applikationsserver) darstellt. Ein fachliches IT-Projekt muss somit auf die

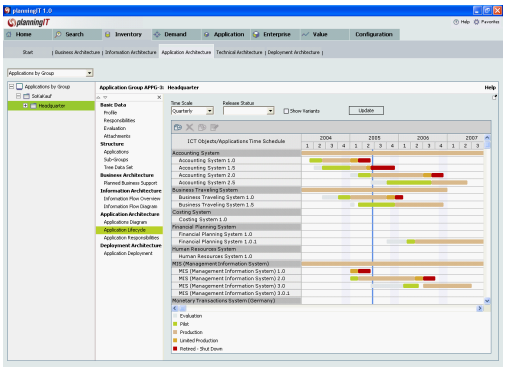


Abbildung 3.25.: *planningIT*: Softwarekarte zur Analyse des Lebenszyklusses von Anwendungssystemen

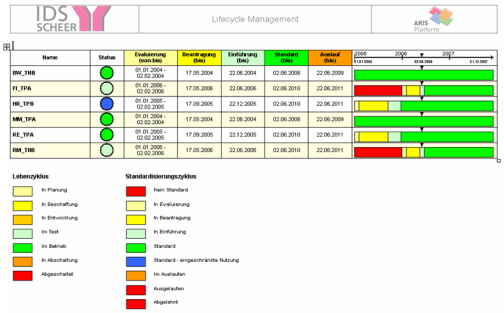


Abbildung 3.26.: *ARIS IT Architect*: Softwarekarte zur Analyse des Lebenszyklusses von Anwendungssystemen

Freigabe bzw. Fertigstellung einer IT-Plattform warten, um das neue Anwendungssystem, welches die IT-Plattform verwendet, in den Betrieb zu übergeben.

Diese Variante einer Softwarekarte, wie sie in Abbildung 3.24 dargestellt ist, wird seit Version 1.0 von *planningIT* (siehe Abbildung 3.25) und seit Version 7.0 in Berichten des *ARIS IT Architect* (siehe Abbildung 3.26) unterstützt. Weitere Einzelheiten zur Werkzeugunterstützung für Softwarekarten werden in Abschnitt 5.2 diskutiert.

Die zweite noch nicht eingeführte Darstellungsform zur Dokumentation von Anwendungslandschaften wird vorwiegend verwendet, um Abhängigkeitsanalysen durchzuführen. Derartige Darstellungen sind entweder wie Graphen aufgebaut oder verwenden Schwimmbahnen, um Gestaltungsmittelinstanzen zu positionieren. Das Beispiel in Abbildung 3.27 ist als Graph aufgebaut und visualisiert alle Kommunikationsbeziehungen des Anwendungssystems *O-Shop*. Die Anwendungssysteme werden wiederum durch Instanzen des Gestaltungsmittels Rechteck visualisiert und die Verbindungen zwischen den Systemen durch Instanzen des Gestaltungsmittels Pfeil. Die Anzahl der angewendeten Gestaltungsregeln ist bei einer derartigen Softwarekarte im Vergleich zu den anderen Softwarekarten sehr gering, es wird lediglich die Regel *Attachment* verwendet, um die Pfeile mit den Rechtecken zu verbinden.

Für weitere Darstellungen in bestimmten Unternehmen wird auf die Diplomarbeiten von Brendebach [Br05a] für die HVB Systems, Lauschke [La05b] für Krones und Sekatzek [Se05b] für die BMW Group verwiesen. In Publikationen außerhalb des universitären Bereiches finden sich ein Teil der gezeigten Softwarekarten bei Dern [De06], Keller [Ke06][13] und Niemann [Ni05].

[13] Der Beitrag zur Darstellung von Anwendungslandschaften in Keller [Ke06] stammt von Lankes, Matthes und Wittenburg [LMW06].

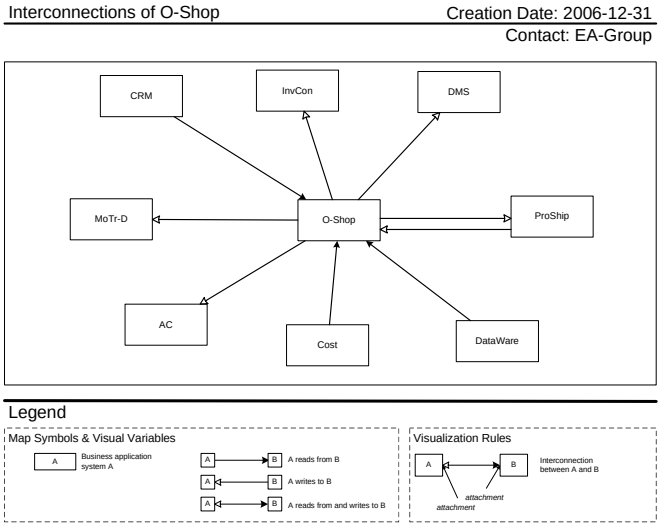


Abbildung 3.27.: Softwarekarte zur Analyse von Beziehungen zwischen Elementen der Unternehmensarchitektur

3.3.2. Merkmale von Anwendungslandschaften

Die in Abschnitt 3.3.1 vorgestellten Softwarekarten zeigen nicht alle relevanten Merkmale von Anwendungslandschaften für die Softwarekartographie. In 2003 und 2004 wurden neben einer ersten Analyse von Softwarekarten ebenso verschiedene Interessenvertreter von Anwendungslandschaften befragt, um einen Überblick über die relevanten Merkmale zu erhalten [MW04a], die auf Softwarekarten visualisiert werden sollen. Im Folgenden werden diese Merkmale entlang der folgenden Kategorisierung vorgestellt[14]:

- Planerische/Strategische Merkmale
- Fachliche Merkmale
- Technische Merkmale
- Wirtschaftliche Merkmale
- Operative Merkmale

Die *planerischen/strategischen Merkmale* beinhalten Merkmale, die eine Veränderung der Anwendungslandschaft über die Zeit ergeben. Projekte und Programme[15], die Auswirkungen auf die Anwendungslandschaft haben, sollen beispielsweise auf einer Softwarekarte visualisiert werden, um Abhängigkeiten zwischen den Projekten zu erkennen.

[14] Eine detaillierte Auflistung der einzelnen Merkmale findet sich in Anhang A.
[15] Programm sei an dieser Stelle als Zusammenfassung mehrerer Projekte zu verstehen.

Drei Einsatzszenarios, die durch eine Softwarekarte mit planerischen/strategischen Merkmalen unterstützt werden, sind:

- Ein Projekt hat das Ziel, ein Anwendungssystem zu verändern oder abzulösen. Um die *Umgebung* des Projektes in der Anwendungslandschaft zu erfassen, muss ersichtlich sein, welche anderen Projekte und Anwendungssysteme von diesem Projekt betroffen sind. Eng verbunden mit dieser Anforderung sind die technischen Merkmale *Kommunikationsbeziehungen* und *Konnektoren für Kommunikationsbeziehungen* (siehe Anhang A), da insbesondere die Beziehungen zwischen Anwendungssystemen für die Planung relevant sind. Projektleiter und Planer haben durch eine Softwarekarte, die Projekte und Kommunikationsbeziehungen visualisiert, die Möglichkeit, Abstimmungs- und/oder Kommunikationsbedarf zu erkennen und entsprechend zu reagieren.
- Zu Beginn einer Planungsperiode muss das vorhandene/genehmigte IT-Budget den beantragten Projekten zugeordnet werden. Nach einer Priorisierung und Auflösung von Abhängigkeiten zwischen den Projekten entsteht eine neue Planung, welche die zurückgestellten bzw. zurückgezogenen Projekte nicht mehr enthält. Die Planer und Strategen erhalten durch eine derartige Softwarekarte die Möglichkeit, die ursprüngliche Planung der Anwendungslandschaft mit der neuen Planung zu vergleichen und über mehrere Planungsperioden die Entwicklung der Anwendungslandschaft zu überwachen sowie die Strategiekonformität zu gewährleisten.
- Ein Unternehmen hat das strategische Ziel, die Zahl der Individualsoftwaresysteme zu reduzieren und diese durch Standard-Softwaresysteme zu ersetzen. Ein derartiger Vorgang kann sich über mehrere Planungsperioden erstrecken, so dass die zeitliche Veränderung über die einzelnen Planungsperioden in einer Softwarekarte angezeigt werden soll. Projektleiter, Strategen und Manager erkennen durch eine derartige Softwarekarte, welche Anwendungssysteme zu welchem Zeitpunkt abgelöst werden sollen und in welchen Perioden Veränderungen geplant sind. Eine Variation dieses Szenarios ist die Einführung eines *Enterprise Portals*, bei der verschiedene Anwendungssysteme zeitversetzt an das Portal angeschlossen bzw. durch das Portal abgelöst werden.

Die Softwarekarten in Abschnitt 3.3.1.5 sind Beispiele für Softwarekarten mit planerischen/strategischen Merkmalen und visualisieren die zukünftige Entwicklung (Plan-Landschaft) bzw. Zielsetzung (Soll-Landschaft) bezüglich der Anwendungslandschaft.

Fachliche Merkmale lassen sich unterteilen in organisatorische, produkt- und prozessorientierte Merkmale, die eng miteinander verzahnt sind, da Organisationseinheiten für die Durchführung bestimmter Prozessschritte verantwortlich sind und umgekehrt Prozessschritte von Personen, die Organisationseinheiten zugeordnet sind, durchgeführt werden. Produkte entstehen entlang der Wertschöpfung durch Geschäftsprozesse in bzw. von unterschiedlichen Organisationseinheiten.

Die Softwarekarten in Abschnitt 3.3.1 enthalten durchgehend fachliche Merkmale. Die Softwarekarten in den Abschnitten 3.3.1.1 und 3.3.1.2 haben beispielsweise den Fokus auf den Organisationseinheiten bzw. Funktionsbereichen und gruppieren die einzelnen

Anwendungssysteme entsprechend dieses Merkmals. Die Softwarekarten in den Abschnitten 3.3.1.3 und 3.3.1.5 ordnen wiederum die einzelnen Anwendungssysteme entlang von Geschäftsprozessen.

Technische Merkmale erstrecken sich von der Implementierungssprache eines Anwendungssystems über die Kommunikationsbeziehungen bis hin zu Eigenschaften wie Softwarearchitektur oder genutzter Middleware. Diesen technischen Merkmalen wird zum Teil in existierenden Modellierungssprachen Rechnung getragen. UML kann beispielsweise mit den Verteilungs- und Komponentendiagrammen die Architektur eines Anwendungssystems modellieren, Kommunikationsbeziehungen und genutzte Middleware können ebenso abgebildet werden.

Die Softwarekartographie adressiert diese Anforderungen auf der Ebene der Anwendungslandschaft, so dass die Kommunikationsbeziehungen innerhalb eines Anwendungssystems nicht von Relevanz sind, sondern ausschließlich zwischen Anwendungssystemen. Durch die Verschiebung des Fokus auf die Anwendungslandschaft lassen sich insbesondere transitive Abhängigkeiten erkennen und Projekte planen (siehe auch planerische Merkmale).

Durch den Bezug der Softwarekartographie zur Anwendungslandschaft können Merkmale auch anders genutzt werden. Die Programmiersprachen der Anwendungssysteme in Kombination mit der Eigenschaft Standard-/Individualsoftware geben Aufschluss über das benötigte Know-how innerhalb eines Unternehmens. Bedingt durch den Wandel von modularen Programmiersprachen zu objektorientierten können in Unternehmen viele Mitarbeiter mit Pascal oder Modula Know-how arbeiten. Ist jedoch der prozentuale Anteil an Anwendungssystemen mit objektorientierten Programmiersprachen stark gestiegen, so können ggf. Kapazitätsüberschüsse bzw. -engpässe entstehen, die so erkannt werden.

Die *wirtschaftlichen Merkmale* umfassen die verschiedenen Kostenarten, die bei der Entwicklung, dem Betrieb, der Wartung, dem Einkauf, etc. von Anwendungssystemen entstanden sind, entstehen bzw. entstehen werden. Die verschiedenen Kostenarten sollen miteinander kombiniert und mit der Anwendungslandschaft in Verbindung gebracht werden, so dass Softwarekarten als Hilfsmittel beim IT-Controlling dienen können.

Die Betrachtung von Betriebskosten in Kombination mit Wartungskosten per anno, kann Anwendungssysteme aufzeigen, die im Vergleich zu anderen Anwendungssystemen einen höheren Wartungsaufwand haben, jedoch im Betrieb vergleichsweise günstig sind.

Die Möglichkeit, Kosten-Kennzahlen bei der Darstellung von Anwendungslandschaften zu berücksichtigen, wurde bisher von den Projektpartnern nur bedingt wahrgenommen. Insbesondere die Problematik, dass nicht alle Unternehmen eine Zuordnung zwischen Kosten und Anwendungssystemen besitzen, erschwert den Aufbau von derartigen Kennzahlensystemen. Da des Weiteren die Visualisierung mit den existierenden Hilfsmitteln sehr aufwändig ist, wurde bisher auf die Visualisierung derartiger Kennzahlen verzichtet. Möglichkeiten zur Visualisierung derartiger Kennzahlen werden in Abschnitt 4.3 aufgezeigt.

Operative Merkmale beziehen sich auf den unmittelbaren Betrieb von Anwendungssystemen und die verbundenen Ereignisse. Das Merkmal *Betriebsort* eines Anwendungs-

systems gibt beispielsweise den Standort des bzw. der physikalischen Server an. Dieser Betriebsort kann von dem Nutzungsort/den Nutzungsorten eines Anwendungssystems abweichen, wobei die Information *Betriebs- vs. Nutzungsort* von Projektpartnern als wertvoll erachtet wird.

Ein anderes operatives Merkmal ist der Ablauf von verschiedenen *Batch-Programmen*, die sich in gegenseitiger Abhängigkeit befinden und zeitlich versetzt laufen müssen. Das Visualisieren derartiger Abhängigkeiten und Abläufe in Anwendungslandschaften unter Nutzung von existierenden Diagrammtypen (z. B. Gantt), wird bereits von Projektpartnern durchgeführt.

3.4. Aufbau von Softwarekarten und Softwarekartentypen

Die in Abschnitt 3.3 vorgestellten Softwarekarten zum Management von Anwendungslandschaften weisen hinsichtlich ihres Aufbaus zahlreiche Gemeinsamkeiten auf. Dieser Aufbau, der sich insbesondere in den Gestaltungsprinzipien und Gestaltungsregeln widerspiegelt, resultiert in den Softwarekartentypen in Abschnitt 3.4.1. Der Bedarf zur Visualisierung verschiedener Merkmale auf gleichartigen Softwarekarten verlangt einen schichtenartigen Aufbau von Softwarekarten, der in Abschnitt 3.4.2 vorgestellt wird. Da die unterschiedlichen Merkmale des Weiteren verschiedene Darstellungen mittels Gestaltungsregeln, Gestaltungsmitteln und Gestaltungsvariablen erfordern, wird die Rolle und der Aufbau von Titeln und Legenden für Softwarekarten in Abschnitt 3.4.3 vorgestellt.

3.4.1. Softwarekartentypen: Clusterkarte, kartesische Karte und Graphlayoutkarte

Bei dem Versuch, Techniken aus der Kartographie (siehe Abschnitt 3.2) zur Erstellung von Softwarekarten als graphische Repräsentation einer Anwendungslandschaft zu nutzen, erweist sich eine grundsätzliche Begrenzung von Karten in der Kartographie als hinderlich. Die Techniken der Kartographie eignen sich nur zur Abbildung von Gegebenheiten, die verortet in einem zwei- oder drei-dimensionalen Raum auftreten. Bei der Darstellung von geographischen Informationen stellt dies keine Beschränkung dar, der Kartengrund ist immer ein topographischer, was unter anderem auch auf die enorme Bedeutung derartiger Informationen zurückgeht: „Our desire for spatial imagery of things in our environment is as normal as breathing.“ [Ro95, S. 9]

Anwendungssysteme als Elemente einer Anwendungslandschaft lassen sich anhand verschiedenster Merkmale (siehe Abschnitt 3.3.2) beschreiben (unterstützte Geschäftsprozesse, Zeitraum des aktiven Betriebs, betreibende Organisationseinheit etc.). Darunter findet sich aber kein Satz von zwei oder drei Merkmalen, die als Dimensionen derartig prominent hervortreten, wie die räumlichen Dimensionen in der Kartographie. Damit steht für Softwarekarten kein eindeutiger Kartengrund fest.

Durch Auswahl bestimmter Merkmale als Dimensionen, die dann zur Bildung des Kartengrunds (zur Verortung) zum Einsatz kommen, erhält der *Softwarekartograph* zusätzlichen

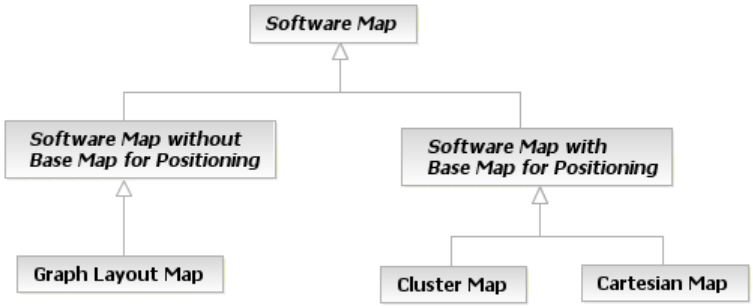


Abbildung 3.28.: Vererbungshierarchie von Softwarekartentypen

Freiraum bei der Gestaltung der Karte. Die Art und Weise, wie diese Dimensionen zur Verortung verwendet werden, führt zu den Softwarekartentypen in Abbildung 3.28, die im Folgenden beschrieben werden.

3.4.1.1. Softwarekarten mit Kartengrund zur Verortung

Ein wichtiges Attribut eines Elements auf einer Karte in der Kartographie stellt die Position des Elements auf dem Kartengrund dar, die sich aus der geographischen Position des Objekts ableitet. Dies legt die Verortung von Elementen auf der Karte fest: Erscheint ein bestimmtes Element an einer anderen Stelle der Karte, ändert sich die von ihr transportierte Botschaft.

Durch den gleichartigen Aufbau und eine gleichbleibende Positionierung von Elementen auf einer Softwarekarte entsteht zusätzlich ein Wiedererkennungswert, der es dem Betrachter erleichtert, sich schnell in der Karte zu orientieren.

3.4.1.1.1. Clusterkarte

Bei dem Softwarekartentyp Clusterkarte (engl. *Cluster Map*) bilden logische Einheiten, die in der Organisation, deren Anwendungslandschaft untersucht wird, existieren, den Kartengrund. Als logische Einheiten einsetzbar sind z. B. Funktionsbereiche, Organisationseinheiten oder auch geographische Einheiten wie Standorte, Städte oder Regionen. Eine Möglichkeit, die verschiedenen logischen Einheiten zu unterscheiden, besteht neben einer geeigneten Beschriftung in der Verwendung eines Farbcodes. Die in den Abschnitten 3.3.1.1 und 3.3.1.2 vorgestellten Softwarekarten sind vom Typ Clusterkarte.

Das Problem der Verortung von Elementen der Karte löst dieser Kartentyp, indem jedes Element in der logischen Einheit, zu der es in Beziehung steht, dargestellt wird. Damit stehen beim Erstellen einer derartigen Karte bereits grobe Regeln fest, wo ein bestimmtes Anwendungssystem (mit den Kartenelementen, die bestimmte Attribute dieses Systems visualisieren) darzustellen ist. Es besteht die Möglichkeit, dass ein System, das zu mehreren der in der Darstellung verwendeten logischen Einheiten gehört, auf der Karte mehrmals erscheint.

Dieser Kartentyp spezifiziert nicht, wie die logischen Einheiten auf der Karte platziert werden und wie sich die verschiedenen Elemente innerhalb der Darstellung einer logischen Einheit anordnen. Bezüglich dieser Fragestellungen werden zwei Vorgehensweisen unterschieden:

- platzoptimierende Verortung
- konventionsgestützte Verortung

Die platzoptimierende Verortung versucht eine Karte mit minimaler Größe zu Erzeugen. Bei der konventionsgestützten Verortung werden beispielsweise Einheiten mit Kundenkontakt rechts und Einheiten mit Lieferantenkontakt links auf der Karte angeordnet.

Der Kartengrund besteht bei diesem Kartentyp vornehmlich aus Instanzen des Gestaltungsmittels Rechteck, wobei auch andere flächenartige Gestaltungsmittel denkbar sind. Um sicherzustellen, dass sich die Cluster auf dem Kartengrund nicht überschneiden, wird die Muss-Gestaltungsregel *Separation* verwendet. Diese Muss-Gestaltungsregel verlangt bei ihrer Anwendung, dass zwei flächenartige Gestaltungsmittelinstanzen separiert dargestellt werden und sich hierbei in maximal einem Punkt überschneiden.

Innerhalb der Cluster werden die Gestaltungsmittelinstanzen für die Anwendungssysteme positioniert. Wie bereits in Abschnitt 3.3.1.1 eingeführt, wird hierzu die Muss-Gestaltungsregel *Nesting* genutzt, welche die Gestaltungsmittelinstanz für den Cluster mit den Gestaltungsmittelinstanzen für die Anwendungssysteme in Beziehung setzt. Um zu gewährleisten, dass sich die Gestaltungsmittelinstanzen für die Anwendungssysteme innerhalb der Cluster nicht überschneiden, wird wiederum die Gestaltungsregel *Separation* verwendet.

Für eine selektive Informationspräsentation bieten sich bei Clusterkarten verschiedene Möglichkeiten an:

- Visualisierung von Kommunikationsbeziehung mit verschiedenem Detaillierungsgrad (siehe Abschnitt 3.3.1.1, Abschnitt 3.3.1.2 und Abschnitt 4.3)
- Visualisierung von Merkmalen durch die Nutzung der Gestaltungsmittelvariablen (siehe Abschnitt 4.3)
- Visualisierung von Merkmalen auf mehreren Schichten (siehe Abschnitt 3.4.2)

Eine Anwendung für diese selektive Informationspräsentation zeigen beispielsweise Abbildung 3.11 und Abbildung 3.13, die mittels unterschiedlicher Darstellungsarten Kommunikationsbeziehungen zwischen Anwendungssystemen darstellen. Abbildung 3.11 visualisiert die Schnittstellen mit kurzen Pfeilen, die mit dem Quell- bzw. Zielsystem beschriftet sind; Abbildung 3.13 verbindet die Anwendungssysteme und verwendet unterschiedliche Pfeilenden für Kommunikationsarten. Mittels Schichten kann diese Information je nach Bedarf ein- oder ausgeblendet werden.

3.4.1.1.2. Kartesische Karte

Eine *kartesische Karte* (engl. *Cartesian Map*) zeichnet sich dadurch aus, dass der Kartengrund entlang einer x- und/oder y-Achse gebildet wird. Prinzipiell ist die Ausrichtung entlang einer Achse ausreichend.

Der Aufbau des Kartengrunds erfolgt mittels der Anwendung der Muss-Gestaltungsregeln *X*- und/oder *Y-Sequence*. Um zusätzlich festzulegen, ob die Achsen am oberen oder unteren bzw. linken oder rechten Rand visualisiert werden, wird eine Instanz des Gestaltungsmittels Rechteck mit transparenten Hintergrund an der entsprechenden Ecke der Softwarekarte positioniert. Die Elemente der Achsen werden mit diesem transparenten Rechteck über die Gestaltungsregeln *X*- bzw. *Y-Sequence* in Beziehung gesetzt. Sollen beispielsweise Chevren die x-Achse bilden, so wird das erste Elemente der Gestaltungsmittelinstanzen für die Prozesskette mit dem Rechteck, welches in der oberen linken Ecke positioniert wird, über eine Instanz der Gestaltungsregel *X-Sequence* in Beziehung gesetzt.

Da eine kartesische Karte lediglich den Aufbau des Kartengrunds im Sinne eines Gestaltungsprinzips definiert, ist dieser Kartentyp – ebenso wie die Clusterkarte – unabhängig von seiner Anwendung oder den verfolgten Interessen (*Concerns*). Die beiden prominentesten kartesischen Karten mit einem Anwendungsbezug sind die *Prozessunterstützungskarte*, deren x-Achse mit Prozessschritten gebildet wird, und die *Zeitintervallkarte*, deren x-Achse aus Zeitintervallen gebildet wird.

Eine Prozessunterstützungskarte fokussiert auf die Prozessorientierung, die insbesondere im Rahmen von organisationseinheitsübergreifenden Prozessen eine größere Bedeutung innerhalb von Unternehmen [Sc01, S. 7-11] erhält. Obwohl zwischen den einzelnen Prozessschritten Schnittstellen existieren, besteht ein Ziel der Prozessorientierung (insb. auf den höheren Ebenen) darin, den kontinuierlichen Verlauf der Wertschöpfung im Prozess zu betonen.

Damit bietet ein Prozess ähnliche Möglichkeiten im Rahmen der Verortung wie eine räumliche Dimension. Eine Voraussetzung für eine derartige Verwendung eines Prozesses ist, dass dieser linear abläuft. Deshalb kommen Prozessdarstellungen auf höheren Ebenen (üblicherweise Ebenen 0 bis maximal 3) zum Einsatz, deren Darstellung, wie beispielsweise in Abbildung 3.21 gezeigt, als Wertschöpfungskette erfolgt. Dieser Prozess legt die x-Achse der Softwarekarte fest.

Da in einer Organisation mehrere Prozesse existieren, die sich für die Verwendung in einer Softwarekarte eignen, muss ein bestimmter Prozess, dies kann der gesamte Primär-Prozess sein, zum Einsatz für die Verortung der Kartenelemente gewählt werden. Diese Auswahl kann sich am erwarteten Einsatzzweck der Karte orientieren. So ist es z. B. ein Einsatzzweck dieses Kartentyps, den Zusammenhang zwischen Prozessschritten und verwendeten Anwendungssystemen zu untersuchen. Durch die gewachsene Anwendungslandschaft eines Unternehmens werden in gleichen Prozessschritten oftmals unterschiedliche Anwendungen mit gleicher Funktion eingesetzt, die aber selbst wiederum verschiedenste Prozessschritte bedienen können. Die Softwarekarte unterstützt in diesem Fall die Identifikation von Optimierungspotentialen und Redundanzen (siehe Abschnitt 5.1).

Zur Verortung auf der y-Achse können bei einer Prozessunterstützungskarte verschiedene Merkmale zum Einsatz kommen, wie:

- Organisationseinheiten oder Standorte, in/an denen ein Anwendungssystem genutzt wird
- Zeit, während der ein Anwendungssystem besteht
- Anwendungssystemtyp, bspw. dispositiv, operativ, administrativ
- Produkte, deren Produktion, Entwicklung oder Vertrieb ein Anwendungssystem unterstützt
- ...

Die Prozessunterstützungskarte weist größere Ähnlichkeit zu den Darstellungen der Kartographie auf, als die Clusterkarte mit ihrer Verortung nach logischen Einheiten. Die logischen Einheiten stellen eher ein nominales Merkmal dar und müssen deshalb erst unter Festlegung einer Vorgehensweise auf dem Kartengrund positioniert werden. Dagegen basiert die Verortung bei Karten vom Typ Prozessunterstützung auf der Auswahl von ein oder zwei Merkmalen als Dimensionen zur Verortung. Dabei lässt sich mindestens eine Dimension, nämlich der Geschäftsprozess, als ordinal betrachten, was die Instanzen besonders geeignet zur Verortung macht. Ein Eignungsgrad von Entfernungen, wie sie in der Kartographie verwendet werden, die ein metrisches Merkmal darstellen, wird zwar nicht erreicht, jedoch ist dieses Merkmal für eine Positionierung auf Softwarekarten anwendbar.

Bei einer Zeitintervallkarte tritt als mögliche Dimension zur Verortung die Zeit in den Mittelpunkt, die auf der x-Achse als intervallskaliertes Merkmal visualisiert wird. Auf der y-Achse bieten sich bei diesem Kartentyp wiederum verschiedene Möglichkeiten an, die mit einem Zeitbezug dargestellt werden sollen. Hierbei ist auch die Betrachtung von mehreren Ebenen möglich, wobei die übergeordneten Elemente die Informationen der untergeordneten Elemente aggregieren. Die folgende Aufzählung zeigt Beispiele für eine y-Achse mit zwei Elementen:

Anwendungssysteme und Anwendungssystemversionen: Die Versionen eines Anwendungssystems stehen in einer Vorgänger-/Nachfolgerbeziehung. Bei einer Anwendungssystemversion werden die Status mit den entsprechenden Zeitintervallen als untergeordnetes Element dargestellt. Für ein Anwendungssystem (übergeordnetes Element) wird der aggregierte Zustand der zugehörigen Anwendungssystemversionen visualisiert (siehe Abbildung 3.24).

Anwendungssysteme und Geschäftsprozessunterstützung: Das Zeitintervall der Prozessunterstützung für einen Geschäftsprozess eines Anwendungssystems wird als untergeordnetes Element dargestellt. Das Anwendungssystem aggregiert als übergeordnetes Element die gesamte Zeitdauer der Geschäftsprozessunterstützungen.

Neben den obigen Beispiels sind weitere Beispiel für eine Zeitinverallkarte denkbar, bei der bspw. Zeitintervalle der Geschäftsprozessunterstützung als übergeordnetes Element und der Lebenszyklus einzelner Anwendungssysteme als untergeordnetes Element sowie

der Lebenszyklus der Anwendungssystemversionen als wiederum untergeordnete Elemente dargestellt werden.

Durch den Aufbau mit Zeitintervallen auf der x-Achse besteht eine Nähe zu vorgangsbezogenen Gantt-Diagrammen [Ba98, S. 32-42].

3.4.1.2. Softwarekarten ohne Kartengrund zur Verortung

Neben den in Abschnitt 3.4.1.1 vorgestellten Softwarekartentypen existieren auch Karten, bei denen die Verortung von Elementen auf der Karte keine festgelegte Bedeutung besitzt. Die Entscheidungen bezüglich der Positionierung der Elemente auf dem Kartengrund bleiben hier vollständig dem Ersteller der Karte überlassen. Damit rücken derartige Karten eher in die Nähe von graphischen Darstellungen, die nicht auf Verortung basieren, wie UML-Klassendiagramme [OM05d], E/R-Diagramme [Ch76], EPKs [KNS92], Petrinetze [JV87] oder auch die ADL ACME [GMW97].

Dieser Softwarekartentyp, der als *Graphlayoutkarte* (engl. *Graph Layout Map*) bezeichnet wird, kommt zum Einsatz, um speziell auf eine Problemstellung hin optimierte Visualisierungen einer Anwendungslandschaft oder von Ausschnitten daraus zu generieren. Ein Einsatzbeispiel ist die in Abschnitt 3.3.1.6 vorgestellte Analyse von Kommunikationsbeziehungen eines bestimmten Anwendungssystems.

Aus der Domäne der Darstellung von Graphen sind verschiedene Positionierungsregeln bekannt, die zu übersichtlichen Darstellungen führen sollen [Pu01]:

- Überschneidungen zwischen Verbindungslinien minimieren
- Rechtecke (z. B. Anwendungssysteme) gleichmäßig über das Diagramm verteilen
- Verbindungslinien von ähnlicher Länge zwischen Rechtecken verwenden
- Gerichtete Verbindungen möglichst in die gleiche Richtung zeigen lassen

Solche Regeln können die Entwicklung von Layout-Algorithmen [Gu04, WEK04] anleiten, die Softwarekarten automatisch aus einem Repository mit Daten zu einer Anwendungslandschaft generieren. Ein derartiger Algorithmus erzeugt z. B. die in Abbildung 3.27 gezeigte Karte, indem der Algorithmus ein vom Benutzer auszuwählendes System in der Mitte der Karte positioniert und die Systeme, die Kommunikationsbeziehungen zum ausgewählten System aufweisen, kreisförmig um dieses herum anordnet.

Stärker als die Befolgung von Regeln, wie den oben genannten, die allgemein für die Darstellung von Graphen entwickelt wurden, wirkt sich die Berücksichtigung von inhaltlichen Aspekten, die nicht direkt in den Gestaltungselementen ausgedrückt werden können, auf die Übersichtlichkeit aus. So wird über die Gruppierung bestimmter Elemente dargestellt, dass diese im Rahmen des betrachteten Problems eng zusammen hängen [Pu01]. In dieser Hinsicht ist es nützlich, Benutzern die Möglichkeit zu geben, die Positionierung der Elemente, wie z. B. bei UML-Werkzeugen üblich, manuell anzupassen.

Durch die relativ großen Freiheiten bei der Verortung, welche dieser Softwarekartentyp bietet, lassen sich Darstellungen erzeugen, die eine spezielle Problematik mit guter

Übersichtlichkeit visualisieren. Die Möglichkeit der freien Positionierung der Elemente steht hier neben der selektiven Darstellung von Information (über Schichten, siehe Abschnitt 3.4.2) als weiteres Mittel zur Komplexitätsreduktion bereit.

Mangels eines Kartengrunds, der auf Dimensionen zur Verortung basiert, fehlt bei diesen Karten die Möglichkeit, bestimmte Informationen mittels Positionierung von Elementen auszudrücken. Beziehungen zwischen Anwendungssystemen und Geschäftsprozessen müssen beispielsweise durch Assoziationslinien ausgedrückt werden.

3.4.2. Schichten in Softwarekarten

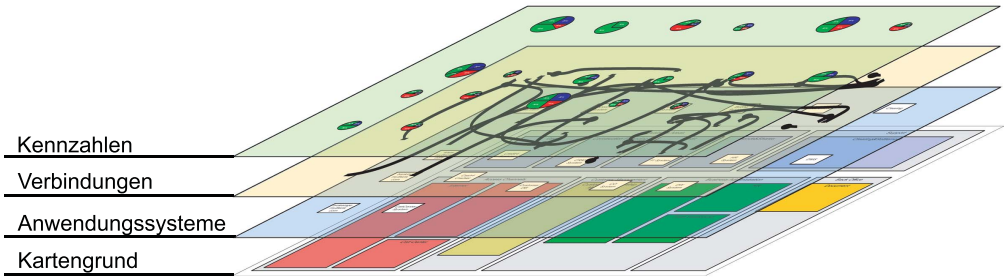


Abbildung 3.29.: Schichtenprinzip von Softwarekarten

Softwarekarten als graphische Repräsentationen von Anwendungslandschaften sollen Anwendungssysteme sowie relevante Merkmale und Beziehungen zwischen Anwendungssystemen visualisieren. Merkmale und Beziehungen können semantisch gruppiert und jeder Gruppe eine Schicht zugewiesen werden. Abbildung 3.29 zeigt beispielhaft den Schichtenaufbau einer Softwarekarte, die aus einem Kartengrund und mehren Schichten besteht. Auf den Kartengrund, der je nach Kartentyp (siehe Abschnitt 3.4.1.1) variiert, werden die unterschiedlichen Schichten aufgetragen, wobei zu jeder Schicht eine Referenzschicht existiert, auf die sich die Elemente dieser Schicht beziehen. Handelt es sich um eine Softwarekarte ohne Kartengrund zur Verortung (siehe Abschnitt 3.4.1.2), so muss die unterste Schicht, die beispielsweise aus den Anwendungssystemen besteht, als Referenzschicht genutzt werden. Durch das Ein-/Ausblenden von Schichten und das Zoom-In/Out können die angezeigten Informationen gefiltert und die Informationsdichte variiert werden, um für einen bestimmten Anwendungsfall die gewünschte Softwarekarte zu erhalten.

Der schichtenartige Aufbau führt zu zwei weiteren Muss-Gestaltungsregeln, die es ermöglichen, mehrere Gestaltungsmittelinstanzen auf einer Schicht zusammenzufassen. Hierbei kontrolliert die Anwendung der Gestaltungsregel *ZLayer*, dass mehrere Gestaltungsmittelinstanzen den gleichen Wert bezüglich der Variable *zIndex* besitzen, und die Gestaltungsregel *Visibility Layer*, dass mehrere Gestaltungsmittelinstanzen den gleichen Wert bezüglich der Variable *visibility* haben (siehe Abschnitt 4.3).

Durch die Verwendung eines Kartengrunds und mehrerer auf diesem Kartengrund aufbauenden Schichten wird es ermöglicht, verschiedene Merkmale mit einem hohen Wiedererkennungswert der Darstellung zu visualisieren und die Informationsdichte entsprechend

zu variieren. Sind beispielsweise die Kommunikationsbeziehungen für einen bestimmten Stakeholder uninteressant, so können diese ausgeblendet, die relevante Kennzahl *Erfüllung des SLA je Anwendungssystem* (siehe Anhang A) jedoch eingeblendet werden.

3.4.3. Titel und Legenden für Softwarekarten

Titel und Legende enthalten wichtige Informationen einer Softwarekarte für den Kartenbetrachter. In der Kartographie beschreiben Hake et al. [HGM02, S.142 ff] den grundlegenden Aufbau einer Karte, der sich aus den folgenden Elementen zusammensetzt:

Kartenfeld: Das Kartenfeld (engl. *Map Surface*) ist die Fläche, die den Karteninhalt als Hauptkarte enthält. Bei einer Softwarekarte ist dies analog.

Kartenrahmen: Der Kartenrahmen (engl. *Map Frame*) ist eine streifenförmige schmale Fläche zwischen der inneren Kartenschnittlinie, die das Kartenfeld abgrenzt, und einer äußeren Begrenzungslinie, an welcher der Kartenrand beginnt. In der Kartographie können im Kartenrahmen beispielsweise die Koordinaten des Kartennetzes angegeben werden. In einer Softwarekarte ist der Kartenrahmen eine durchgehende Linie.

Kartenrand: Der Kartenrand (engl. *Map Margin*) ist die Kartenfläche außerhalb des Kartenrahmens. In der Kartographie enthält der Kartenrand typischerweise die Legende und weitere Angaben zur Karte. Bei einer Softwarekarte besteht der Kartenrand aus der Legende und einem Titelfeld.

Der Aufbau einer Softwarekarte wird analog zu einer Karte in der Kartographie gewählt, um zu gewährleisten, dass die Softwarekarte richtig interpretiert werden kann. Abbildung 3.30 zeigt den grundlegenden Aufbau einer Softwarekarte, bestehend aus dem Kartenfeld, dem Kartenrahmen und dem Kartenrand, der das Titelfeld und die Legende enthält.

Das Titelfeld umfasst bei einer Softwarekarte einen Kartentitel, das Erstellungsdatum und einen Kontakt (Person, Gruppe etc.). Der Titel sollte *sprechend* gewählt werden, um beispielsweise zu unterscheiden, ob es sich um eine Ist- oder Soll-Landschaft handelt. Das Erstellungsdatum gibt Aufschluss über den Informationsstand, welcher der Visualisierung zugrunde liegt, und der Kontakt erlaubt es dem Kartennutzer, Fragen an den richtigen Adressaten zu richten.

Die Legende ist bei einer Softwarekarte zweigeteilt und umfasst zum einen die Gestaltungsmittel inkl. ihrer Gestaltungsvariablen und zum anderen die Gestaltungsregeln. Bei den Gestaltungsregeln sind insbesondere die Muss-Gestaltungsregeln anzugeben (siehe Abschnitt 4.3). Die anonymisierten Softwarekarten in Abschnitt 3.3 entsprechen bereits den geforderten Aufbau.

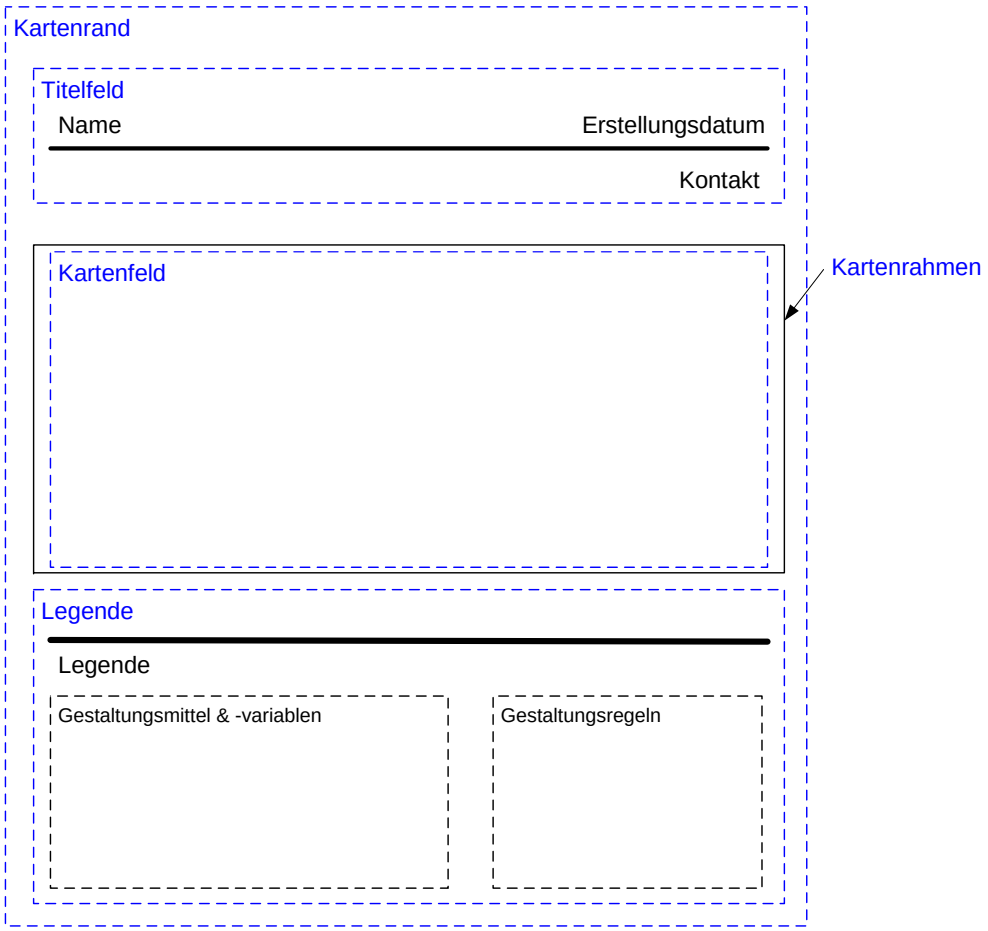


Abbildung 3.30.: Aufbau einer Softwarekarte mit Kartenfeld, Kartenrahmen und Kartenrand

3.5. Bewertung

In Abschnitt 3.1 wurden die grundlegenden Begriffe für Modellierungssprachen eingeführt, um eine Modellierungssprache mit Softwarekarten als zentrales Element adäquat zu entwickeln, zu beschreiben und zu definieren. Die Rolle von Modellen und Methoden für Modellierungssprachen – dies ebenso im Kontext des IEEE 1471 – haben gezeigt, welche Anforderungen an eine Modellierungssprache existieren.

Der Abschnitt zum Thema Kartographie (Abschnitt 3.2) hat den Aufbau eines kartographischen Zeichensystems vorgestellt und die Begriffe Gestaltungsmittel, -variable und -regel eingeführt, die in Abschnitt 3.3 zur Beschreibung der Softwarekarten genutzt wurden.

Nachfolgend hat Abschnitt 3.3 aufgezeigt, wie verschiedene Unternehmen und Branchen ihre Anwendungslandschaft dokumentieren und managen. Die gezeigten Softwarekarten

bieten einen Querschnitt über die verwendeten Modelle zum Management von Anwendungslandschaften und wurden durch die relevanten Merkmale für Softwarekarten abgerundet.

Die sich anschließende Kategorisierung der Softwarekarten in Abschnitt 3.4 mittels der Softwarekartentypen hat die verschiedenen Softwarekarten auf gemeinsame Gestaltungsprinzipien zurückgeführt, so dass wenige Kartentypen entstanden sind, welche die Modelle für Anwendungslandschaften unterscheiden. Die Wiederverwendung von Prinzipien aus der Kartographie hat dargelegt, welche Ansprüche an den Aufbau einer Softwarekarte existieren und insbesondere die Rollen von Schichten, Titeln und Legenden für Softwarekarten motiviert.

Die Grundlage für eine konkrete Syntax für Softwarekarten als Modelle in der Softwarekartographie wurde durch die Softwarekartentypen in Abschnitt 3.4 und die Analyse der Softwarekarten mit Gestaltungsmittel, -variablen und -regeln in Abschnitt 3.3 gebildet, so dass diese Teile im folgenden Kapitel 4 zu einer geschlossenen Syntax zusammengesetzt werden können.

Für eine in sich abgeschlossene Modellierungssprache wird neben dem Visualisierungsmodell in Kapitel 4, welches die abstrakte Syntax beschreibt und auch eine Serialisierungssyntax ergibt, ebenso die Rolle des Informationsmodells als Metamodell für die Semantik diskutiert. Eine enge Kopplung der Semantik an die Visualisierung wird zeigen, wie Softwarekarten als Managementwerkzeug fungieren und mehr als *gemalte Bilder* einer Anwendungslandschaft darstellen.

KAPITEL 4

Theoretische Grundlagen von Softwarekarten

Inhaltsverzeichnis

Die Beispiele für Softwarekarten in Kapitel 3 haben gezeigt, dass verschiedene gleichartige Visualisierungen für Anwendungslandschaften existieren, die entlang der Softwarekartentypen kategorisiert werden können. Dieses Kapitel verlässt die konzeptuelle Ebene der

Softwarekartographie und zeigt detailliert die Semantik und Syntax von Softwarekarten. Ziel ist es, eine in Werkzeugen implementierbare Syntax für Softwarekarten zu erhalten, die das Modellieren von Anwendungslandschaften ermöglicht.

Einer geschlossenen Modellierungssprache von Anwendungslandschaften wirkt insbesondere die Anwendung von Werkzeugen entgegen, die zur Erstellung von Softwarekarten verwendet werden. Es können grundsätzlich zwei Arten von Werkzeugen unterschieden werden[1]: Nicht-Repository-basierte Werkzeuge, wie beispielsweise Microsoft Visio oder Microsoft PowerPoint und Repository-basierte Werkzeuge, wie beispielsweise *ARIS Toolset* (IDS Scheer AG) oder *Corporate Modeler* (Casewise, Inc.).

Die Nicht-Repository-basierten Werkzeuge bieten ohne Erweiterungen keine Unterstützung bei der Erstellung von Softwarekarten. Existierende Schablonen für Microsoft Visio (beispielsweise von ArchiMate[2] [JLL06]), bieten für Softwarekarten rudimentäre Unterstützung an, jedoch treten bei der Anwendung von Nicht-Repository-basierten Werkzeugen folgende Probleme auf:

- Die Daten existieren ausschließlich in einer einzigen Visualisierung und können nicht in tabellarischen Auswertungen oder in anderen Visualisierungen wiederverwendet werden.
- Das Fehlen eines Informationsmodells[3], welches die Semantik der verwendeten Elemente, der Attribute und der Beziehungen festlegt, ermöglicht das *Zeichnen* beliebiger Modelle, ohne dass die Bedeutung der Elemente klar definiert ist.
- Die Werkzeuge unterstützen standardmäßig keine Konzepte zur relativen Positionierung von Elementen, wie sie beispielsweise eine Muss-Gestaltungsregel *Nesting* fordert, welche Einfluss auf die Positionierung nimmt. Die Positionierung der Elemente erfolgt manuell, ohne dass das Werkzeug die Gestaltungsregeln in das Modell aufnimmt oder deren Einhaltung gewährleisten kann. Werden Schablonen definiert, wie beispielsweise die Microsoft Visio Schablone von ArchiMate [JLL06], so bleibt dieser Missstand ohne zusätzliche Programmierung bestehen.
- Schablonen, die nicht Teil der Modellierungssprache sind, können genutzt werden, was zu einem Aufweichen der von der Modellierungssprache vorgegebenen Syntax und Semantik führt und *Zeichnungen* ohne definierte Semantik ermöglicht. Dieser Missstand kann von einer Schablone durch entsprechende Makros behoben werden.

Bei den Repository-basierten Werkzeugen werden einige der oben beschriebenen Probleme behoben. Derartige Werkzeuge besitzen entweder bereits ein Informationsmodell, welches ggf. adaptiert werden kann, oder ermöglichen das Definieren eines eigenen Informationsmodells. Hinsichtlich der Visualisierung bleiben jedoch oftmals die oben angesprochenen Probleme bestehen, wie die *Enterprise Architecture Management Tool Survey 2005* [se05a] gezeigt hat. Vielen Werkzeugen fehlt insbesondere die Möglichkeit, die

[1] Für eine Diskussion von Werkzeugen zum *Management von Unternehmensarchitekturen*, die zur Erstellung von Softwarekarten verwendet werden, wird auf Abschnitt 5.2 verwiesen.
[2] Zu ArchiMate siehe auch Abschnitt 4.5.2.
[3] Zu einer genaueren Diskussion von Informationsmodellen siehe Abschnitt 4.2.

Positionierung von Elementen zum Transport von Semantik zu nutzen, wie es die Softwarekartentypen in Abschnitt 3.4 verlangen.

Um die Probleme zu verdeutlichen, wird im Folgenden die Unterscheidung von semantischen und symbolischen Modellen in der Softwarekartographie eingeführt (Abschnitt 4.1). Aufbauend auf dieser Unterscheidung der Modelle werden die jeweiligen Metamodelle, das Informationsmodell (siehe Abschnitt 4.2) für das semantische Modell und das Visualisierungsmodell (siehe Abschnitt 4.3) für das symbolische Modell, eingeführt. Die Anwendung von Modelltransformationen in Abschnitt 4.4 zeigt, wie bei der Erzeugung von Softwarekarten aus semantischen Modellen symbolische Modelle entstehen und wie im Gegenzug Veränderungen in der Softwarekarte (symbolisches Modell) zu Änderungen an den Daten im semantischen Modell führen.

Abschnitt 4.5 diskutiert in diesem Kapitel abschließend die Modellierungssprachen UML sowie ArchiMate und grenzt diese gegenüber den Softwarekarten der Softwarekartographie ab, um zu zeigen, dass die existierenden Modellierungssprachen nicht den Anforderungen zur Beschreibung, Bewertung und Gestaltung von Anwendungslandschaften entsprechen.

4.1. Zusammenhang von semantischen und symbolischen Modellen

Abbildung 4.1 zeigt eine Softwarekarte vom Typ Clusterkarte, die den Betrieb (engl. *Hosting*) von Anwendungssystemen an verschiedenen Standorten unter der Anwendung der

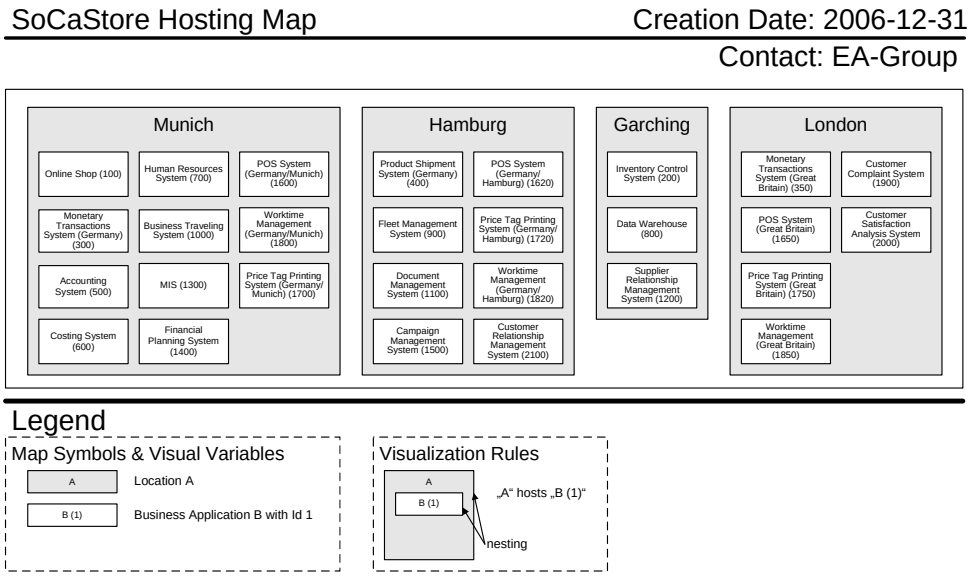


Abbildung 4.1.: Clusterkarte mit Standorten und betriebenen Anwendungssystemen

Muss-Gestaltungsregel *Nesting* auf die Gestaltungsmittelinstanzen darstellt. Das Anwendungssystem `Online Shop (100)` wird beispielsweise am Standort `Headquarter` betrieben. Um ein tiefergehendes Verständnis der Softwarekarte in Abbildung 4.1 zu erhalten, werden im Folgenden vier Modelle (semantisches Modell, Informationsmodell, symbolisches Modell und Visualisierungsmodell), die der Abbildung zugrundeliegen, extrahiert bzw. *reengineered*.

Das Extrahieren des semantischen Modells, welches die Informationen aus der Softwarekarte enthält, und des Informationsmodells, welches das Metamodell für das semantische Modell ist, resultiert in den UML-Diagrammen in den Abbildungen 4.2 und 4.3, wenn eine objektorientierte Notation gewählt wird.

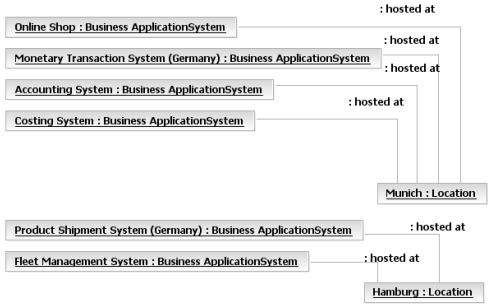


Abbildung 4.2.: Semantisches Modell der Clusterkarte in Abbildung 4.1

Abbildung 4.3.: Informationsmodell der Clusterkarte in Abbildung 4.1

Das Objektdiagramm in Abbildung 4.2 visualisiert einen Ausschnitt[4] des semantischen Modells, wobei jedes Objekt in dem Diagramm ein Informationsobjekt repräsentiert, beispielsweise den Standort `Munich` oder das Anwendungssystem `Human Resources System (700)`. In der Abbildung nicht enthalten sind die Eigenschaften der Informationsobjekte, beispielsweise der Wert `100` des Attributs `id` des Objekts `OnlineShop` der Klasse `ApplicationSystem`.

Der Begriff *semantisches Modell* wird wie folgt definiert:

> **Semantisches Modell :** Ein semantisches Modell (engl. *Semantic Model*) stellt die Semantik einer Visualisierung (beispielsweise einer Softwarekarte) dar. Das Modell ist eine Instanz eines Informationsmodells.

Das Informationsmodell in Abbildung 4.3 ist das Metamodell des semantischen Modells und besteht aus

- den Klassen `Location` und `ApplicationSystem`,
- dem Attribut `name` für beide Klassen,

[4] Das vollständige semantische Modell ist für eine lesbare Abbildung zu groß, da in diesem Fall alle Standorte, Anwendungssysteme und ihre Beziehungen visualisiert werden müssten (insg. 34 Objekte).

- dem Attribut `id` für die Klasse `ApplicationSystem` und
- der Beziehung `hosted at` zwischen `Location` und `ApplicationSystem`.

Da der Begriff *Informationsmodell* je nach Kontext unterschiedlich benutzt wird, werden an dieser Stelle zwei Begriffsdefinitionen angeführt, um anschließend die für diese Arbeit angewandte Definition wiederzugeben. Teubner [Te99] definiert ein Informationsmodell wie folgt:

> Informationsmodelle bilden nicht nur einzelne Informationssysteme ab, sondern das Unternehmen bzw. die Unternehmensorganisation und stellen damit einen Bezugsrahmen für die Beschreibung und organisatorische und softwaretechnische Integration der Informations- bzw. Anwendungssysteme dar. [...]

Eine andere Definition findet sich bei Becker und Schütte [BS96]:

> Ein Informationsmodell ist das immaterielle Abbild des betrieblichen Objektsystems aus Sicht der in diesem verarbeiteten Informationen für Zwecke des Informationssystem- und Organisationsgestalters.

Die Zahl der verschiedenen Definitionen lässt sich weiter erhöhen, wenn der Bereich der Informationsmodellierung (siehe z. B. Krcmar [Kr05]) betrachtet wird. Um für diese Arbeit ein einheitliches Verständnis zu erhalten, wird eine abstraktere Definition gewählt, und der Begriff des Informationsmodells wie folgt verstanden:

Informationsmodell : Ein Informationsmodell (engl. *Information Model*) ist ein Modell, welches die Semantik von Informationsobjekten, die möglichen Beziehungen zwischen den Informationsobjekten und die Attribute von Informationsobjekten definiert. Ein Informationsobjekt ist ein Abbild eines existierenden Objekts in einem betrieblichen Objektsystem.

Zur vollständigen Beschreibung eines Informationsmodells muss, wenn eine Modellierung mit UML gewählt wird, neben dem UML-Klassendiagramm die Semantik der Elemente (Klassen, Attribute, Assoziationen etc.) definiert werden. Die Beschreibung kann beispielsweise in Form eines Glossars als natürlichsprachlicher Text erfolgen[5]. Eine mögliche Definition der Klasse `ApplicationSystem` ist die folgende[6]:

> Ein *Anwendungssystem* ist ein System, das in Software implementiert ist, und welches Teil eines Informationssystems [WK94] ist und mindestens einen Geschäftsprozess unterstützt. Analog zu Krcmar [Kr05, S. 25] wird der Mensch als Teil des Informationssystems, einem soziotechnischen System, nicht zum Anwendungssystem hinzugezählt. Ebenso ist die Infrastruktur nicht Teil des Anwendungssystems, sondern wird vom Anwendungssystem genutzt und ist somit ein Serviceerbringer aus Sicht des Anwendungssystems.

[5] Erlauben die Elemente die Nutzung einer formalen Sprache oder ist diese ggf. notwendig, so ist dies eine adäquate Alternative.

[6] Zur Definition des Begriffs *betriebliches Anwendungssystem* siehe auch Abschnitt 1.3.

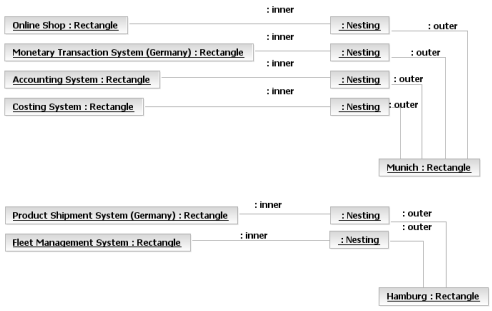


Abbildung 4.4.: Symbolisches Modell der Clusterkarte in Abbildung 4.1

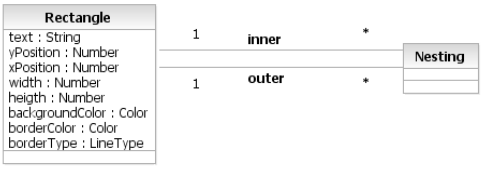


Abbildung 4.5.: Visualisierungsmodell der Clusterkarte in Abbildung 4.1

Neben den semantischen Informationen, die in dem Informationsmodell enthalten sind, könnten weitere Informationen aus der Softwarekarte in Abbildung 4.1 extrahiert werden, wenn beispielsweise die graue Hintergrundfarbe der Rechtecke für Standorte eine Bedeutung besäße. Im vorliegenden Beispiel sind die Informationen anhand der Legende als vollständig anzusehen, so dass das Informationsmodell vollständig ist.

Neben dem semantischen Modell und dem Informationsmodell werden aus der Softwarekarte in Abbildung 4.1 ein symbolisches Modell und ein Visualisierungsmodell abgeleitet; nachfolgend ebenso mittels UML dargestellt. Das symbolische Modell (siehe Abbildung 4.4 für einen Ausschnitt) zeigt auf Instanzebene, welche Elemente die Softwarekarte bilden und welche Einschränkungen hinsichtlich der Positionierungen existieren. Das Visualisierungsmodell, als Metamodell für das symbolische Modell, spezifiziert, welche Gestaltungsmittel und Gestaltungsregeln für eine Softwarekarte verwendet werden. Die Abbildung 4.5 zeigt ein vereinfachtes Visualisierungsmodell, welches ausschließlich die Elemente der Softwarekarte in Abbildung 4.1 enthält und nicht weiter generalisiert oder abstrahiert ist.

Die Begriffe *symbolisches Modell* und *Visualisierungsmodell*, die in diesem Kontext in der Informatik nicht verwendet werden, sind wie folgt definiert:

Symbolisches Modell : Ein symbolisches Modell (engl. *Symbolic Model*) stellt die Symbolik einer Visualisierung (beispielsweise einer Softwarekarte) dar. Das Modell ist eine Instanz eines Visualisierungsmodells.

Visualisierungsmodell : Ein Visualisierungsmodell (engl. *Visualization Model*) ist ein Modell, welches die in einer Visualisierung verwendbaren Visualisierungselemente definiert. Die Elemente eines Visualisierungsmodell sind die Gestaltungsmittel inkl. ihrer Gestaltungsvariablen und die Gestaltungsregeln sowie Elemente zur Konstruktion einer Darstellung (Rahmen, Beschriftung, Schicht, Legende etc.).

Das einzige in Abbildung 4.1 verwendete Gestaltungsmittel ist das Rechteck, welches als Gestaltungsvariablen beispielsweise die Breite, die Höhe und ein Textfeld besitzt und im objektorientierten Visualisierungsmodell (Abbildung 4.5) durch die Klasse `Rectangle` repräsentiert wird. Zusätzlich wird in der Darstellung die Muss-Gestaltungsregel *Nesting* angewendet, um die Semantik, dass ein bestimmter Standort ein Anwendungssystem betreibt, zu transportieren, indem das Rechteck für das Anwendungssystem innerhalb des Rechtecks für den Standort positioniert wird.

Diese beiden Elemente, das Gestaltungsmittel `Rectangle` und die Muss-Gestaltungsregel `Nesting`, bilden das Visualisierungsmodell. In Abbildung 4.5 nicht enthalten ist die Muss-Gestaltungsregel *Separation*, die verhindert, dass sich zwei Rechtecke für Anwendungssysteme überlappen. Diese Regel wird in der Abbildung nicht dargestellt, um die Lesbarkeit des UML-Diagramms für das symbolische Modell (siehe Abbildung 4.4) zu erhalten.

Die Objekte des symbolischen Modells in Abbildung 4.4 sind Instanzen der Klassen des Visualisierungsmodells in Abbildung 4.5 und korrespondieren mit den verwendeten Gestaltungsmitteln und -regeln der Clusterkarte. Nicht dargestellt in Abbildung 4.4 sind die Attribute der Gestaltungsmittel, beispielsweise der Wert `Munich` des Attributs `text` des Objekts `Munich:Rectangle`.

Das Extrahieren des semantischen Modells, des Informationsmodells, des symbolischen Modells und des Visualisierungsmodells ergibt eine objektorientierte Sicht auf die abstrakte Syntax und die Semantik[7] der Clusterkarte aus Abbildung 4.1. Die Semantik der Softwarekarte wird durch das Informationsmodell beschrieben und ist Gegenstand von Abschnitt 4.2. Die Anwendung der konkreten Syntax ergibt die Softwarekarte selbst und die abstrakte Syntax wird durch das Visualisierungsmodell beschrieben, welches in Abschnitt 4.3 detailliert wird.

Um die Bedeutung der Unterscheidung von semantischen und symbolischen Modell inklusive ihrer Metamodelle zu verdeutlichen, ist in Abbildung 4.6 ein Beispiel für eine *unzureichende* Softwarekarte gegeben, die Mehrdeutigkeiten hinsichtlich der Semantik zulässt. Diese Clusterkarte in Abbildung 4.6 zeigt eine Variation der Clusterkarte aus Abbildung 4.1, wobei ein zusätzliches Rechteck mit einem roten und gestrichelten Rahmen sowie der Beschriftung *Individual Software* eingefügt wurde. Die Legende erklärt in diesem Fall nicht, welche Semantik das zusätzliche Rechteck besitzt. Folgende Interpretationen sind denkbar:

- Alle Anwendungssysteme, die innerhalb des Rechtecks mit der Beschriftung *Individual Software* positioniert sind, sind Individualsoftwaresysteme; alle anderen Anwendungssysteme sind keine Individualsoftwaresysteme.
- Alle Anwendungssysteme, die innerhalb des Rechtecks mit der Beschriftung *Individual Software* und innerhalb des Rechtecks für den Standort `London` positioniert sind, sind Individualsoftwaresysteme; alle anderen Anwendungssysteme sind keine Individualsoftwaresysteme.
- ...

[7] Zu Syntax und Semantik von Modellierungssprachen siehe Abschnitt 3.1.2.

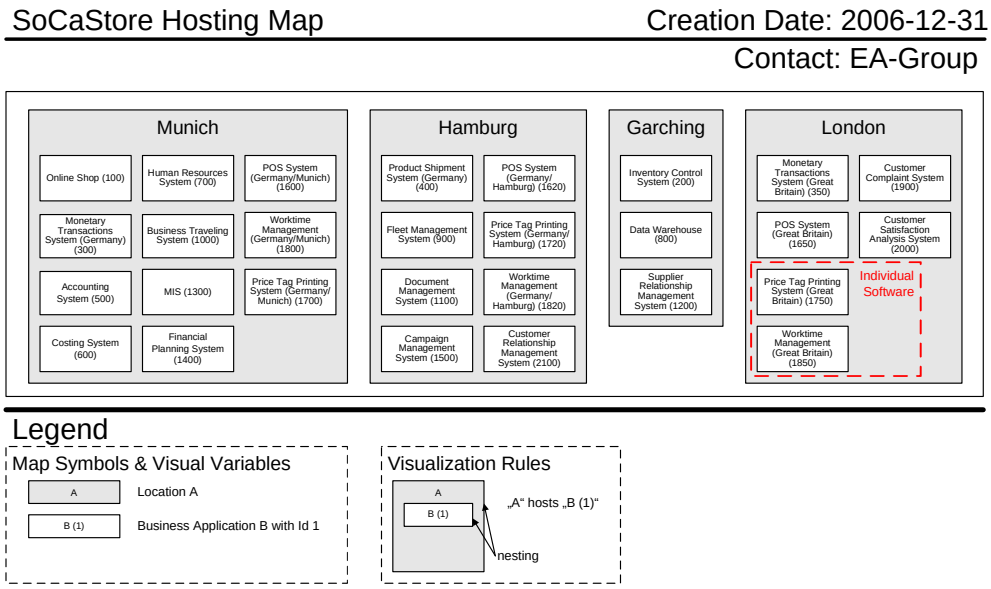


Abbildung 4.6.: Clusterkarte aus Abbildung 4.1 mit einem zusätzlichen Gestaltungsmittel

Ein derartiges Modell der Anwendungslandschaft kann entstehen, wenn das Modellierungswerkzeug das Hinzufügen bzw. Zeichnen von zusätzlichen graphischen Elementen zulässt, welche nicht Bestandteil der konkreten Syntax sind. Problematisch ist, dass zum einen deren Semantik nicht eindeutig ist und zum anderen die Information auf diese Art und Weise nicht in einem Repository gespeichert werden kann. Dies bedingt des Weiteren, dass diese Information nur in der Visualisierung existiert und nicht in anderen Auswertungen (textueller oder graphischer Form) enthalten sein kann.

Die obige Argumentation zeigt, dass eine *enge* Kopplung zwischen dem Informationsmodell und Visualisierungsmodell benötigt wird, um eine Konsistenz zwischen den Informationen (semantisches Modell) und den Visualisierungen (symbolisches Modell) zu gewährleisten.

Im Folgenden werden Anforderungen an ein Informationsmodell für Unternehmensarchitekturen vorgestellt (Abschnitt 4.2) und im Anschluss das Visualisierungsmodell für Softwarekarten aufgebaut (Abschnitt 4.3). Der in Abschnitt 4.4 vorgestellte Ansatz zeigt anschließend, wie durch die Anwendung von Modelltransformationen Konsistenz zwischen dem semantischen und symbolischen Modell erreicht werden kann.

4.2. Informationsmodelle für die Unternehmensarchitektur

Wie in Kapitel 2 dargelegt, bedient sich das Management der Anwendungslandschaft u. a. des Managements der Unternehmensarchitektur. Die Anwendungslandschaft, bestehend aus einer Vielzahl von Anwendungssystemen, kann nicht als autarkes System, sondern

nur eingebettet in den Beziehungskontext der Unternehmensarchitektur analysiert und gesteuert werden. Dies impliziert, dass ein Informationsmodell zum Management der Anwendungslandschaft Elemente der Unternehmensarchitektur, wie beispielsweise Organisationseinheiten, Geschäftsprozesse, Strategien, mit einbeziehen muss.

Das Informationsmodell[8] kann als Datenmodell, ergänzt um eine geeignete Beschreibung der Elemente (Klassen, Assoziationen, Attribute etc.), gesehen werden und legt fest, welche Zusammenhänge der Architektur dokumentiert werden sollen. Die Instanzen der Elemente des Informationsmodells sind die Informationsobjekte, die einzelne Elemente der Architektur beschreiben.

Das Informationsobjekt `Customer Relationship Management System` wäre beispielsweise eine Instanz der Klasse `ApplicationSystem`, wobei die Werte der Attribute, wie Identifikationsnummer, Name, Beschreibung etc., Eigenschaften des Informationsobjekts sind. Beziehungen zu anderen Objekten, zum Beispiel dem Prozess `Kundenakquisition`, komplettieren die Information. Es entsteht also ein Geflecht von Objekten, das durch geeignete Visualisierungen (beispielsweise tabellarisch oder graphisch) erfahrbar wird. Softwarekarten sind eine graphische Repräsentation der Informationsobjekte.

Abschnitt 4.2.1 diskutiert die Schichten und Querschnittsfunktionen eines Strukturmodells für Informationsmodelle und stellt Teile eines Informationsmodells vor. Im Anschluss werden existierende Informationsmodelle für Unternehmensarchitekturen in Abschnitt 4.2.2 kritisch reflektiert, um einen Verbesserungsvorschlag unter der Anwendung von Mustern in Abschnitt 4.2.3 vorzustellen.

4.2.1. Elemente von Informationsmodellen – Schichten und Querschnittsfunktionen

Ein Informationsmodell für das Management von Unternehmensarchitekturen muss die relevanten Informationsobjekte, ihre Attribute und ihre Beziehungen untereinander in einem geschlossenen Modell abbilden. In Werkzeugen zur Modellierung von Unternehmensarchitekturen sind Informationsmodelle mit 30 bis zu über 300 Klassen [se05a] zu finden.

Als Modellierungssprache für ein Informationsmodell kann die Meta Object Facility der OMG [OM06d] unter Zuhilfenahme der Klassendiagramme der UML [OM05d] verwendet werden. Die *ausschließliche* Anwendung der UML 2.0 Superstructure ist ungeeignet, da die Superstructure zum einen verhaltensorientierte Konzepte beinhaltet, die für ein Informationsmodell nicht relevant sind, und zum anderen keine Implementierung der Superstructure existiert. Für MOF 1.4 [OM02] existieren bereits Implementierungen[9] und für MOF 2.0 [OM06d] sind zahlreiche in Arbeit[10]. Die in dieser Arbeit verwendete Notation für Informationsmodelle entspricht der von UML-Klassendiagrammen, dies jedoch

[8] Für eine Begriffsdefinition siehe Abschnitt 4.1.

[9] Siehe beispielsweise `http://mdr.netbeans.org/` (abgerufen 2007-03-19).

[10] Siehe beispielsweise `http://www2.informatik.hu-berlin.de/sam/meta-tools/aMOF2.0forJava/index.html` (abgerufen 2007-03-19).

mit der Einschränkung, dass ausschließlich Konzepte von (*Complete*) MOF 2.0 verwendet werden. Dies führt dazu, dass beispielsweise keine Assoziationsklassen modelliert werden.

Im Folgenden wird anhand von Beispielen gezeigt, dass verschiedene Varianten von Informationsmodellen existieren. Es gibt somit nicht das *eine* Informationsmodell, welches ohne eine Adaption zum Einsatz kommen kann. Dies ist analog zu Standardsoftwaresystemen, welche bestimmte Adaptionsmöglichkeiten in ihren Modellen vorsehen.

Wird aus der Softwarekarte in Abbildung 3.21 das Informationsmodellfragment[11] extrahiert, so ergeben sich verschiedenen Modellierungsvarianten, die aus einer ternären Beziehung entstehen:

Variante 1: Modellierung der ternären Beziehung zwischen den Klassen `ApplicationSystem`, `BusinessProcess` und `OrganizationalUnit` durch die Klasse `SupportRelationship` (siehe Abbildung 4.7). Dies führt zu einer Modellierung, in der weder Versionen von Anwendungssystemen noch der Betrieb eines Anwendungssystem an einem Standort berücksichtigt werden.

Variante 2: Modellierung der ternären Beziehung zwischen den Klassen `ApplicationSystemVersion`, `BusinessProcess` und `OrganizationalUnit` durch die Klasse `SupportRelationship` (siehe Abbildung 4.8). Dies ergibt gegenüber Variante 1 eine detailliertere Modellierung, die es erlaubt zu unterscheiden, welche Anwendungssystemversion bei welcher Organisationseinheit einen Geschäftsprozess unterstützt. Insbesondere unter Berücksichtigung der Weiterentwicklung von Anwendungssystemen, die in neuen Anwendungssystemversionen resultieren, wird durch diese Variante im Informationsmodell eine erhöhte Genauigkeit erzielt. Eine Berücksichtigung des Betriebs an einem Standort erfolgt nicht.

Variante 3: Modellierung der ternären Beziehung zwischen den Klassen `DeployedApplicationSystemVersion`, `BusinessProcess` und `OrganizationalUnit` durch die Klasse `SupportRelationship` (siehe Abbildung 4.9). Im Unterschied zu Variante 2 kann in dieser Variante festgestellt werden, welche Geschäftsprozessunterstützung bei welcher Organisationseinheit betroffen ist, wenn eine betriebene (*deployed*) Anwendungssystemversion ausfällt. Ein vollständiges Informationsmodell, in welchem dieses Fragment existiert, würde zusätzlich eine Klasse `Location` für einen Standort und eine Assoziation dieser Klasse zu `DeployedApplicationSystemVersion` besitzen.

Die obige Aufzählung der drei Varianten kann durch weitere Modellierungsmöglichkeiten, beispielsweise unter Auslassung der Anwendungssystemversionen aber mit betriebenen Standort, erweitert werden. Welche Modellierungsvariante im Informationsmodell gewählt wird, muss auf die Interessen der Stakeholder zurückgeführt werden (siehe Abschnitt 3.1.1.1). Ist es beispielsweise nicht relevant, zu erkennen, welche Geschäftsprozessunterstützung bei welcher Organisationseinheit durch einen Ausfall eines Anwendungssystems betroffen ist, so wäre Variante 3 nicht *notwendig*. Sie würde Informationen verlangen, die aus Sicht der Interessen der Stakeholder nicht benötigt werden.

[11] Ein Informationsmodellfragment ist ein Teil eines Informationsmodells.

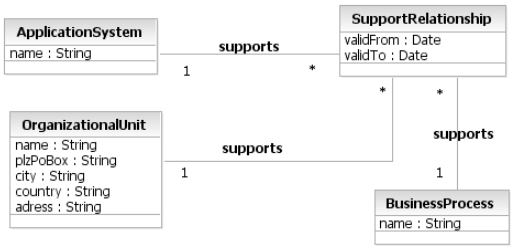


Abbildung 4.7.: Informationsmodellfragment für Prozessunterstützung – Variante 1

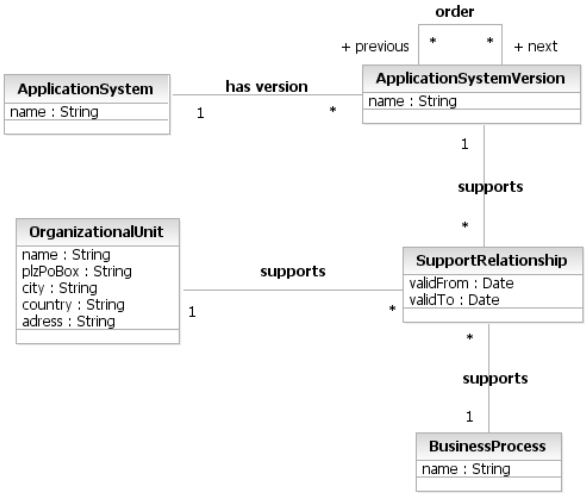


Abbildung 4.8.: Informationsmodellfragment für Prozessunterstützung – Variante 2

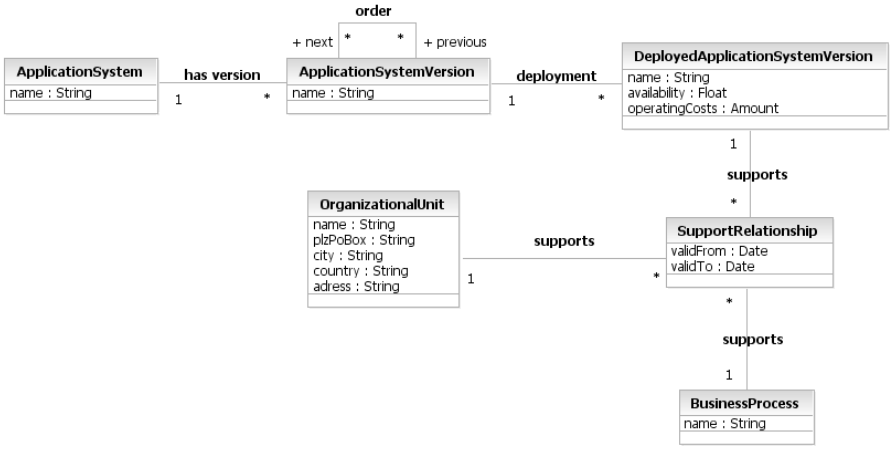


Abbildung 4.9.: Informationsmodellfragment für Prozessunterstützung – Variante 3

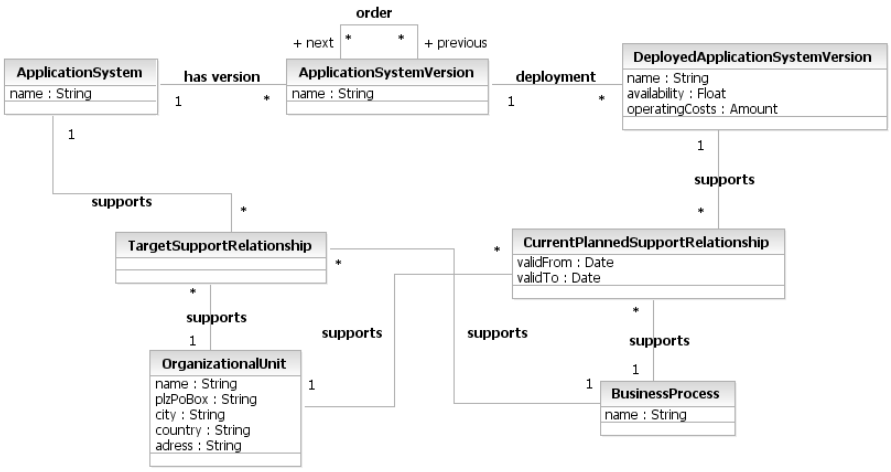


Abbildung 4.10.: Informationsmodellfragment für Prozessunterstützung – Variante 4

Sind die Interessen der Stakeholder derart, dass nicht nur Ist- und Plan-Zustände hinsichtlich der Prozessunterstützung bei Organisationseinheiten erfasst werden sollen, sondern auch Soll-Landschaften, so kann dies durch das Informationsmodellfragment in Abbildung 4.10 erreicht werden. Hierbei wird die Soll-Situation durch eine zweite ternäre Beziehung mittels der Klasse `TargetSupportRelationship` dargestellt und zur Darstellung der Ist- und Plan-Zustände tritt die Klasse `CurrentPlannedSupportRelationship` an die Stelle der einfachen `SupportRelationship`. Ein Soll-Zustand einer Unterstützung wird somit an ein Anwendungssystem gebunden, da Soll-Zustände einen langfristigen Charakter besitzen und somit im Gegensatz zu Ist- oder Plan-Zuständen keine Information über Versionen und/oder Betriebsorte existieren.

Derartige Varianten in der Gestaltung von Informationsmodellen existieren in verschiedenen Ausprägungen. Einige Beispiele, deren Aufzählung nicht als vollständig angesehen wird, sind wie folgt:

- Der Detailgrad bei der Modellierung von Kommunikationsbeziehungen zwischen Anwendungssystemen, wie bereits in den Abschnitten 3.3.1.1 und 3.3.1.2 vorgestellt, variiert. Dies reicht von der Modellierungsvariante auf Attributebene zur Abbildung von lesendem vs. schreibendem Zugriff sowie der Datenflussrichtung bis hin zu Modellen, in denen Informationen über die von Anwendungssystemen an ihren Schnittstellen angeboten Konnektoren existieren, wie sie von Matthes und Wittenburg [MW04b] beschrieben werden. Derartige Modelle können mit zusätzlichen Klassen und Assoziationen realisiert werden (siehe Abschitt 4.2.3).
- Im Kontext von Service-orientierten Architekturen wird ein Anwendungssystem als Service-Erbringer gegenüber einem Geschäftsprozess angesehen. Wird der *Service* in das Informationsmodell aufgenommen, so ändert sich die Modellierung der Geschäftsprozessunterstützung (siehe Abbildung 4.10) dahingehend, dass nicht das Anwendungssystem an der ternären Beziehung mit Geschäftsprozessen und Organisationseinheiten teilnimmt, sondern der Service. Ein Modell, welches sowohl

Anwendungssysteme als auch Services vorsieht, würde ggf. eine abstrakte Klasse einführen, um beide Varianten zu erlauben, und zusätzlich über eine Regel sicherstellen, dass keine Redundanzen oder Widersprüche auftreten.

- Ist die Geschäftsprozessunterstützung bei unterschiedlichen Organisationseinheiten niemals unterschiedlich, so würde die ternäre Beziehung aus Abbildung 4.10 zwischen Anwendungssystemen, Geschäftsprozessen und Organisationseinheiten wegfallen und nur eine Beziehung *supports* zwischen Anwendungssystemen und Geschäftsprozessen existieren.
- Sollen die von einem Anwendungssystem genutzten Infrastrukturelemente (Applikationsserver, Datenbankmanagementsysteme etc.) modelliert werden, so ist dies entweder durch einzelne Klassen für verschiedene Infrastrukturtypen zu erzielen, oder durch das Modellieren so genannter *Infrastruktur-Stacks*. Ein Infrastruktur-Stack ist eine Zusammenstellung von verschiedenen Infrastrukturelementen, die in bestimmten Versionsständen eine *gültige* Konfiguration ergeben. *Gültig* bedeutet, dass diese Konfiguration vom IT-Betrieb freigegeben wurde.

Um die für ein Informationsmodell relevanten Elemente besser einordnen zu können, zeigt Abbildung 4.11 ein Strukturmodell der zu betrachtenden Ebenen (engl. *Layers*) sowie der auf diese Ebenen einwirkenden Querschnittsfunktionen (engl. *Cross Functions*). Eine Modularisierung in verschiedene Pakete oder auch der Einsatz von Mustern, die wiederkehrende Problemstellungen mit Best-Practice-Ansätzen lösen (siehe Abschnitt 4.2.3), ermöglicht es, das Gesamtmodell zu strukturieren und unternehmensspezifisch zusammenzusetzen. Des Weiteren ergibt sich eine handhabbare Untermenge des Gesamtmodells, wobei sich einzeln beschriebene Module oder Muster aus Artefakten von einem oder mehreren Blöcken der Ebenen und Querschnittsfunktionen zusammensetzen.

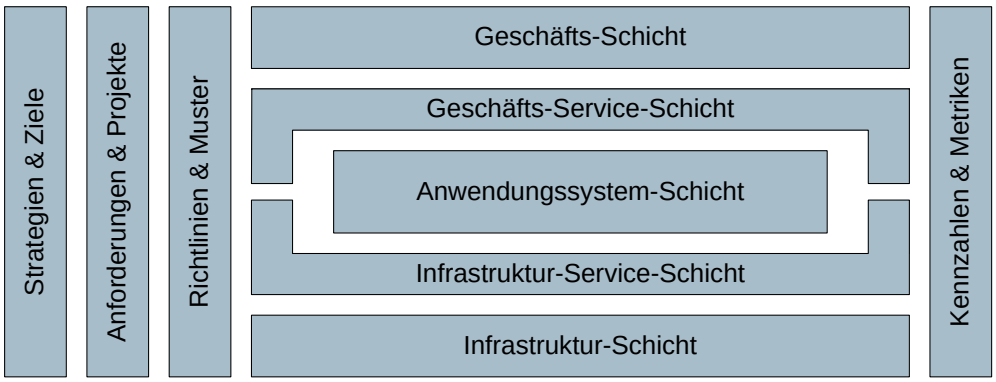


Abbildung 4.11.: Schichten und Querschnittsfunktionen als Struktur eines Informationsmodells

Die fünf Ebenen *Geschäfts-Schicht*, *Geschäfts-Service-Schicht*, *Anwendungssystem-Schicht*, *Infrastruktur-Service-Schicht* und *Infrastruktur-Schicht* umfassen die Informationsobjekte der Unternehmensarchitektur, die durch die Querschnittsfunktionen beeinflusst werden, welche ihrerseits wieder weitere Informationsobjekte enthalten.

Die Geschäfts-Schicht enthält Produkte, Geschäftsprozesse, Organisationseinheiten etc., die über die Geschäfts-Service-Schicht mit Artefakten der Anwendungssystem-Schicht verbunden sind. Die Geschäfts-Service-Schicht kapselt z. B. Geschäftsobjekte (engl. *Business Objects*), Geschäfts-Services (engl. *Business Services*) und zugehörige *Service Level Agreements* (SLAs), die mittels der Anwendungssysteme und anderer Elemente der Anwendungssystem-Schicht erfüllt werden.

Die Infrastruktur-Service-Schicht stellt die Verbindung zwischen den Artefakten der Anwendungssystem-Schicht und der Infrastruktur-Schicht her. Analog zur Geschäfts-Service-Schicht kapselt die Infrastruktur-Service-Schicht die Infrastruktur-Services und die zugehörigen SLAs (vgl. z. B. [OG00a, OG00b]). Die Infrastrukturelemente (z. B. Middleware- und Hardwaresysteme) der Infrastruktur-Schicht erbringen die notwendigen Services für die Artefakte der Anwendungssystem-Schicht.

Die Querschnittsfunktionen *Strategien & Ziele*, *Anforderungen & Projekte* und *Richtlinien & Muster* beeinflussen die Elemente der unterschiedlichen Ebenen. Strategien & Ziele generieren oder erzeugen Handlungsbedarfe, die in Form von Anforderungen in Projekten und Programmen (Anforderungen & Projekte) umgesetzt werden. Bei der Umsetzung der Anforderungen werden zusätzlich Richtlinien & Muster berücksichtigt, um z. B. Homogenität und Standardisierung im Unternehmen zu erreichen.

Die vierte Querschnittsfunktion *Kennzahlen & Metriken* bildet die Steuerungskomponente der Architektur und komplettiert das Modell: „If you can't measure it, you can't manage it!“[12]

Die Darstellung der Geschäfts-Service-Schicht und der Infrastruktur-Service-Schicht als *Kapsel* für die Anwendungssystem-Schicht begründet sich in der Idee von Service-orientierten Architekturen, bei denen nicht das Anwendungssystem sondern der *Service* im Vordergrund steht. Bei einer Service-orientierten Architektur entspricht der *Service* einem Geschäfts-Service, welcher für den operativen Betrieb verschiedene Infrastruktur-Services benötigt. Ebenso resultieren Anforderungen, die in Projekten mit einem Bezug zu einer Service-orientierten Architektur umgesetzt werden, in neuen oder geänderten Geschäfts-Services. Das Anwendungssystem tritt somit in den Hintergrund und ist als eine logische Zusammenfassung und Implementierung von Geschäfts-Services zu sehen.

Wie umfassend ein Informationsmodell aufgebaut ist und wie viele der Schichten und Querschnittsfunktionen adressiert werden, hängt von den Interessen der Stakeholder an der Unternehmensarchitektur ab. Sind *dynamische* Aspekte wie Anforderungen und Projekte nicht relevant, so würden diese entsprechend im Informationsmodell fehlen. Ziel der Strukturierung in Abbildung 4.11 ist es, aufzuzeigen, aus welchen Teilen ein Informationsmodell bestehen kann. Zusätzlich kann das Strukturmodell als Diskussionsbasis bei der Entwicklung eines Informationsmodells dienen.

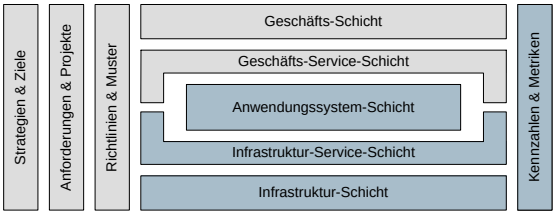
4.2.2. Existierende Informationsmodelle in Forschung und Praxis

Zielsetzung eines Informationsmodells zur Abbildung einer Unternehmensarchitektur ist es, die aus Sicht der Stakeholder relevanten Elemente zu erfassen und diese Sichten an

[12] Das Zitat wird Peter F. Drucker zugeschrieben.

den Interessen der Stakeholder auszurichten. In Forschung und Praxis existieren zahlreiche Informationsmodelle im Kontext von Unternehmensarchitekturen, die im Detail stark variieren. Im Folgenden werden drei Informationsmodelle entlang der Struktur aus Abbildung 4.11 vorgestellt. Diese Struktur aus Abbildung 4.11 wird für jedes Modell in verkleinerter Form dargestellt. Ein blauer Hintergrund bedeutet, dass das Modell Elemente dieser Schichten bzw. Querschnittsfunktionen enthält, ein grauer Hintergrund entsprechend das Gegenteil.

CIM – Common Information Model

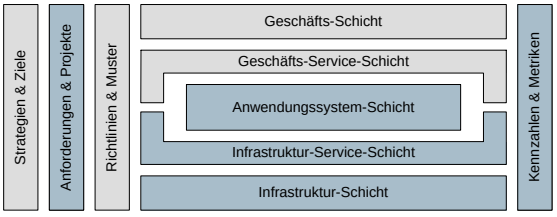


Das *Common Information Model* (CIM) wurde von der Distributed Management Task Force entwickelt [DM05] und fokussiert auf das Management von Infrastrukturelementen. Zum einen ist der Detailgrad des Modells für Unternehmensarchitekturen zu hoch und zum anderen werden für eine Unternehmensarchitektur relevante Modellelemente nicht berücksichtigt. Im CIM besitzt beispielsweise die Klasse NFS (*Network File System*) Attribute für *ReadBufferSize* oder *MountFailureRetries* [DM06b], welche für eine Unternehmensarchitektur irrelevant sind. Ein Ausschnitt des Informationsmodells ist in Abbildung B.1 im Anhang B enthalten.

Das Modell besitzt keine Elemente aus der Geschäfts-Schicht und Geschäfts-Service-Schicht, folglich sind Geschäftsprozesse oder Organisationseinheiten in dem Modell nicht bekannt. Gleichfalls nicht adressiert werden Strategien, Projekte, Musterarchitekturen etc., so dass die entsprechenden Querschnittsfunktionen keine Berücksichtigung finden. Als Informationsmodell für eine Unternehmensarchitektur ist das CIM somit nicht geeignet. Ist es jedoch im Interesse der Stakeholder, Informationen über Systemstatus etc. zu erlangen, so können Daten eines CIM-konformen Modells in aggregierter Form als Lieferant dienen.

CIM basiert auf einem eigenen Metamodell, dem *Management Object Format*[13] und definiert eine DTD für einen Datenaustausch basierend auf XML, so dass die *Implementierbarkeit* gewährleistet ist.

ITPMF – IT Portfolio Management Facility



Die IT Portfolio Management Facility (ITPMF) Spezifikation [OM06c] der OMG, welche sich in der Finalisierung befindet, wurde entwickelt, um das IT Portfolio, bestehend aus Anwendungssystemen

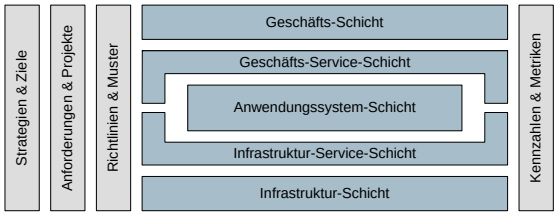
[13] Das *Management Object Format* wird von der DTMF als MOF abgekürzt, ist jedoch nicht mit der *Meta Object Facility* oder dem *Microsoft Operations Framework* verwandt.

und Infrastrukturelementen (u.a. Hardwaresysteme, Middlewaresysteme) zu dokumentieren und zu steuern. Der Detailgrad des Modells ist im Vergleich zum CIM gering, jedoch sind typisierbare Elemente vorhanden, welche es erlauben verschiedene Subtypen abzubilden. Ein Ausschnitt des Informationsmodells ist in Abbildung B.2 im Anhang B enthalten.

Wie CIM verfügt ITPMF über keine Elemente der Geschäfts-Schicht und der Geschäfts-Service-Schicht. Darüber hinaus werden die Querschnittsfunktionen, Strategien & Ziele sowie Richtlinien & Muster nicht adressiert und aus der Querschnittsfunktion Anforderungen & Projekte werden lediglich Anforderungen berücksichtigt. Eine zeitliche Versionierung von Elementen, um beispielsweise Ist-, Plan- und Soll-Landschaften abzubilden, erfolgt nicht. Das ITPMF ist somit als Informationsmodell für eine Unternehmensarchitektur nicht geeignet. Für die Infrastruktur-Schicht ist das Modell ein Startpunkt, welches mit den Interessen der Stakeholder abgeglichen werden muss.

Das Modell der ITPMF definiert als Serialisierungssyntax ein XMI-konformes Austauschformat und ist somit als *implementierbar* zu bezeichnen. Ein Werkzeug, welches das Modell ITPMF als konfiguriertes Metamodell enthält, ist der *Adaptive IT Portfolio Manager* von Adaptive, Inc.

ArchiMate



ArchiMate ist eine graphische Modellierungssprache für Unternehmensarchitekturen, die am Telematica Instituut entwickelt wurde [Jo06]. ArchiMate besitzt ein eigenes Schichtenmodell [Jo05, S. 86], welches sich aus den Schichten *Customers*, *Business Services*, *Business Processes*, *Application Services*, *Application Components*, *Infrastructural Services* und *Technical Infrastructure* zusammensetzt. Ein Ausschnitt des Informationsmodells ist in Abbildung B.3 im Anhang B enthalten.

In ArchiMate nicht berücksichtigt sind die Querschnittsfunktionen aus Abbildung 4.11. Darüber hinaus besitzen die Elemente der Schichten keine Informationen über den Lebenszyklus. Das Modell eignet sich somit nicht zur Abbildung von unterschiedlichen Ist-, Plan- und Soll-Landschaften, ist jedoch geeignet, um als Ausgangsbasis für ein Informationsmodell zu dienen, welches um die Querschnittsfunktionen und Lebenszyklusinformationen ergänzt werden muss.

Für die Modellierungssprache ArchiMate existieren verschiedene Implementierungen. Dies sind zum einen Eigenentwicklungen vom Telematica Instituut, die auf einem eigenen Metamodell basieren [Jo04], und zum anderen Implementierungen in kommerziellen Modellierungswerkzeugen, wie beispielsweise dem *ARIS Toolset* (IDS Scheer AG). Als Serialisierungssyntax wird von ter Doerst et al. [Do05] zwar XMI favorisiert, jedoch existiert bisher kein öffentlich verfügbares Schema.

Weitere Modelle

Neben den drei vorgestellten Informationsmodellen existieren zahlreiche weitere Modelle:

- Braun und Winter [BW05] haben im Rahmen eines Forschungsprojekts ein Informationsmodell für Unternehmensarchitekturen entwickelt, welches Elemente in den Schichten *Strategy Layer*, *Organization Layer* und *Application Layer*[14] definiert. Das Modell fokussiert sich ausschließlich auf die oberen drei Ebenen des Strukturmodells aus Abbildung 4.11. Da für das Modell keine Serialisierungssyntax oder eine detaillierte Beschreibung inklusive Attributen publiziert wurde, können keine Rückschlüsse gezogen werden, ob in dem Modell Informationen zum Lebenszyklus etc. enthalten sind. Nach Braun und Winter [BW05] wurde das Modell prototypisch mit dem Werkzeug *ADONIS* von BOC umgesetzt.
- Brendebach [Br05a] hat im Rahmen einer Diplomarbeit ein Informationsmodell für das Management von Anwendungslandschaften für die HVB Information Services[15] entwickelt. Dieses Modell wurde nur in Teilen implementiert, deckt jedoch alle Schichten und Querschnittsfunktionen aus Abbildung 4.11 ab. Die fehlende Umsetzung des Modells erlaubt jedoch keine Rückschlüsse auf eine Anwendbarkeit zur Modellierung von Unternehmensarchitekturen.
- Halbhuber [Ha04] hat im Rahmen einer Diplomarbeit ein Informationsmodell für das IT-Management der BMW Group erstellt. Dieses Modell deckt alle Schichten und Querschnittsfunktionen aus Abbildung 4.11 ab, jedoch fehlt eine vollständige Implementierung des Modells und damit die Möglichkeit, die Anwendbarkeit zu validieren.
- Lauschke [La05b] hat im Rahmen einer Bachelor-Arbeit ein Informationsmodell für das Management von IT-Landschaften bei Krones entwickelt. Das Modell deckt ausschließlich die Schichten des Strukturmodells in Abbildung 4.11 ab und hat seinen Schwerpunkt auf dem Management von Infrastrukturelementen und Anwendungssystemen. Das Modell wurde prototypisch mittels Microsoft Access umgesetzt.
- Kirchner [Ki03] hat im Rahmen eines Forschungsprojektes ein Informationsmodell zur Modellierung von IT-Landschaften entwickelt. Dies Modell beinhaltet nur Elemente der Schichten des Strukturmodells aus Abbildung 4.11.
- MEMO [Fr95, Fr02] (Multi Perspective Enterprise Modelling) wurde zur Modellierung von *Unternehmensmodellen* entwickelt und unterscheidet in einer Matrix die drei Perspektiven Strategie, Organisation und Informationssystem sowie die vier Aspekte Ressource, Struktur, Prozess und Ziel. Das Modell hat einen Fokus auf die Verknüpfung der betriebswirtschaftlichen Ebene mit der informationsverarbeitenden Ebene und bezieht nur die Querschnittsfunktion Strategien & Ziele sowie die oberen drei Schichten ein.

[14] Braun und Winter verwenden für den *Application Layer* auch den Begriff *System Layer*.
[15] ehemals HVB Systems

Weitere Informationsmodelle existieren in den Werkzeugen zum Management von Anwendungslandschaften und Unternehmensarchitekturen (siehe Abschnitt 5.2), diese Modelle sind jedoch zumeist nicht frei verfügbar und werden daher an dieser Stelle nicht tiefergehend betrachtet.

Vergleichbare Strukturmodelle, wie in Abbildung 4.11, existieren des Weiteren bei ARIS (siehe Abschnitt 2.3.4), der META Group [ME02], TOGAF [TOG02], Iyer und Gottlieb [IG04], Lichtenegger et al. [LRR03] etc. Wesentlich für das Strukturmodell in Abbildung 4.11 ist, dass die Elemente der Querschnittsfunktionen sich auf die Elemente aller Schichten auswirken und diese verändern bzw. verändert haben. Darüber hinaus existiert eine zusätzliche Querschnittsfunktion zur Steuerung.

Mit Ausnahme von ArchiMate fehlt es den vorgestellten Modellen an einer Unterstützung der Service-Schichten, die entweder von Anwendungssystemen für Geschäftsprozesse oder von Infrastrukturelementen für Anwendungssysteme erbracht werden.

Resümee

Als abschließende Aussage ist festzuhalten, dass *das eine* Informationsmodell nicht existiert. Durch die unterschiedlichen Interessen der Stakeholder ergeben sich verschiedene Strukturen in den Informationsmodellen. Um *dem einen* Informationsmodell möglichst nahe zu kommen, sind Adaptionstechniken wie auch bei Standardsoftwaresystemen vorzusehen, so dass beispielsweise im Rahmen eines *Customizing* aus den Interessen die richtigen Strukturen abgeleitet werden.

Eine Möglichkeit, um einen Fortschritt bei der Entwicklung von Informationsmodellen zu erreichen, stellt der folgende Abschnitt 4.2.3 vor. Dieser Ansatz verfolgt das Ziel, wiederkehrende Strukturen zu identifizieren, die sich an den Interessen von Stakeholdern orientieren und das ständige *Neuerfinden des Rades* vermeiden.

4.2.3. Muster für Informationsmodelle

Im Software-Engineering sind Muster (engl. *Patterns*) weithin bekannt. Die prominenteste Sammlung von Mustern für die Entwicklung von objekt-orientierter Software ist von Gamma et al. [Ga95]. Ein Muster wird von Gamma et al. wie folgt definiert:

> A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities. Each design pattern focuses on a particular object-oriented design problem or issue. It describes when it applies, whether it can be applied in view of other design constraints, and the consequences and trade-offs of its use. [Ga95, S. 3f]

Ein Muster stellt wiederverwendbare Strukturen dar und beschreibt den Aufbau, die Anwendungen, die resultierenden Einschränkungen und seine Konsequenzen. Gamma et al. nutzen als Motivation ein Zitat von Alexander, der ein Muster als Lösung eines wiederholt auftretenden Problems versteht und im Kern eine möglichen Lösung beschreibt [AIS77, S. x].

Im Kontext von Unternehmensarchitekturen wurde von Buckl et al. [Bu07] ein Vorschlag erarbeitet, wie sich der Musterbegriff auf Unternehmensarchitekturen übertragen lässt. Nach Buckl et al. setzt sich ein derartiges Muster aus drei Teilen zusammen:

Methode: Die Methode beschreibt die adressierten Interessen der Stakeholder und die Schritte, die bei der Anwendung des Musters ausgeführt werden.

Viewpoint: Der Viewpoint[16] beschreibt die Visualisierung, die in der Methode angewendet wird.

Informationsmodellmuster: Ein Informationsmodellmuster beschreibt ein Fragment eines Informationsmodells, welches die notwendigen Daten für den Viewpoint enthält.

Derzeit wird am Lehrstuhl für Informatik 19 (sebis) der TU München eine Studie zu diesen Mustern durchgeführt, um die prominentesten von diesen zu identifizieren und zu dokumentieren. Im Sinne des Ansatzes von Buckl et al. stellt beispielsweise das Informationsmodellfragment in Abbildung 4.10 einen Informationsmodellmuster dar. Ein Beispiel des zugehörige Viewpoints und die Methode sind in Abschnitt 3.3.1.5 enthalten.

Im Folgenden werden zwei weitere Beispiele für Informationsmodellmuster gegeben, die Kommunikationsbeziehungen zwischen Anwendungssystemen und Musterarchitekturen für Anwendungssysteme adressieren.

4.2.3.1. Informationsmodellfragment für Kommunikationsbeziehungen

Die Vernetzung der Anwendungslandschaft wird auf Softwarekarten unterschiedlich dargestellt und dokumentiert[17]. Werden die Darstellungsformen generalisiert, ergeben sich die folgenden Stufen (siehe Abbildung 4.12), die zu dem Informationsmodellmuster in Abbildung 4.13 führen.

Stufe 0 (engl. *Level 0*) visualisiert die Anwendungssysteme in Zusammenhang mit logischen Einheiten (Organisationseinheiten, Standorte etc.). Auf dieser Stufe werden keine direkten Verbindungslinien zwischen den einzelnen Anwendungssystemen gezeichnet.

Werden die Anwendungssysteme mittels Linien oder Pfeilen miteinander verbunden (*Stufe 1*), soll dies Schnittstellen dokumentieren, die einzelne Systeme miteinander verbinden. Die Pfeilenden der Linien stellen entweder die Datenflussrichtung oder die Rolle (Client vs. Server) dar und führen somit zu einer Überladung dieses Gestaltungsmittels.

In der betrieblichen Praxis wird der Begriff Schnittstelle auch zur Bezeichnung einer Kommunikationsbeziehungen zwischen zwei Anwendungssystemen verwendet, was nicht

[16] Zum Begriff *Viewpoint* siehe Abschnitt 3.1.1.1.
[17] Für Beispiele siehe Abbildung 3.3.1.1 und 3.3.1.2 in Abschnitt 3.3.

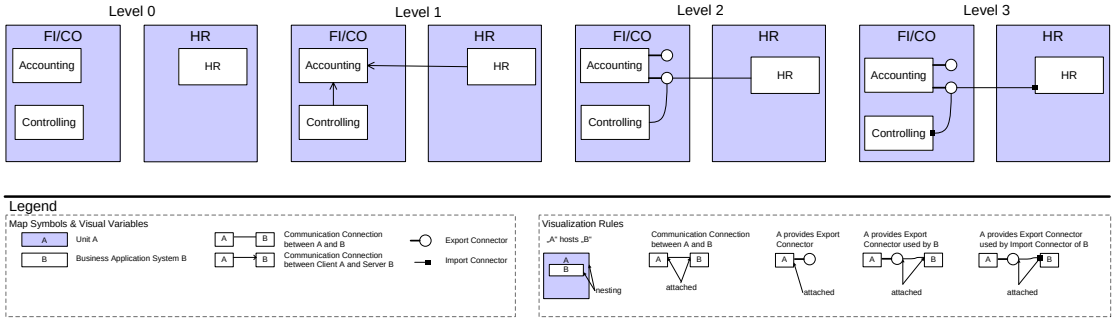


Abbildung 4.12.: Visualisierung von Kommunikationsbeziehungen mittels verschiedener Detaillierungsgrade

der im Software Engineering üblichen Terminologie entspricht. So definiert UML eine Schnittstelle als „An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. An interface specifies a contract; any instance of a classifier that realizes the interface must fulfill that contract." [OM05d, S. 82]; vergleichbares gilt für Definitionen bei CORBA [OM06b] oder Szyperski [Sz02]. Vor diesem Hintergrund wird in diesem Modell der Begriff *Kommunikationsbeziehung* verwendet, um den überladenen Begriff der Schnittstelle zu vermeiden. Eine Kommunikationsbeziehung definiert, dass zwei Anwendungssysteme in einer Beziehung stehen, bei der es sich um einen Daten- oder Kontrollfluss handeln kann.

Die existierenden Definitionen und Visualisierungen von Kommunikationsbeziehungen unterscheiden nicht zwischen einer technischen und fachlichen Sicht. Werden Schnittstellen im Kontext der Anwendungslandschaft analysiert, so treten die *exakten* Definitionen der einzelnen Methoden und deren Signaturen in den Hintergrund. Stattdessen sind die Konnektoren mit Kommunikationsart, Art des Zugriffs (lesend vs. schreibend) und die Technologie entscheidend. Wird von dem Vertrag, den ein Export-Konnektor schließt, abstrahiert, so sind der Geschäfts-Service und die betroffenen Geschäftsobjekte für das Netz in der Anwendungslandschaft von Bedeutung (siehe Abbildung 4.12).

Stufe 2 führt somit zu einer Betrachtung von Export-Konnektoren, die Geschäfts-Services für andere Anwendungssysteme anbieten, und einer Unterscheidung zwischen einer fachlichen und einer technischen Ebene. Ein Anwendungssystem kann hierbei den gleichen Service, der durch einen fachlichen Export-Konnektor angeboten wird, mittels verschiedener Technologien exportieren. Für einen fachlichen Export-Konnektor können somit mehrere technische Export-Konnektoren existieren (in Abbildung 4.12 nicht visualisiert).

Wird eine Verbindung mit Export-Konnektoren analysiert, so agiert das exportierende System als Server, welches Geschäfts-Services gegenüber anderen Systemen (den Clients) anbietet.

In einer *Stufe 3* werden auf Seite der Clients die Import-Konnektoren ergänzt. Diese Import-Konnektoren nutzen alle oder Teile der von den Export-Konnektoren angebotenen Funktionen. Import-Konnektoren greifen unter Umständen nur lesend auf einen Export-Konnektor zu, der sowohl lesenden als auch schreibenden Zugriff anbietet.

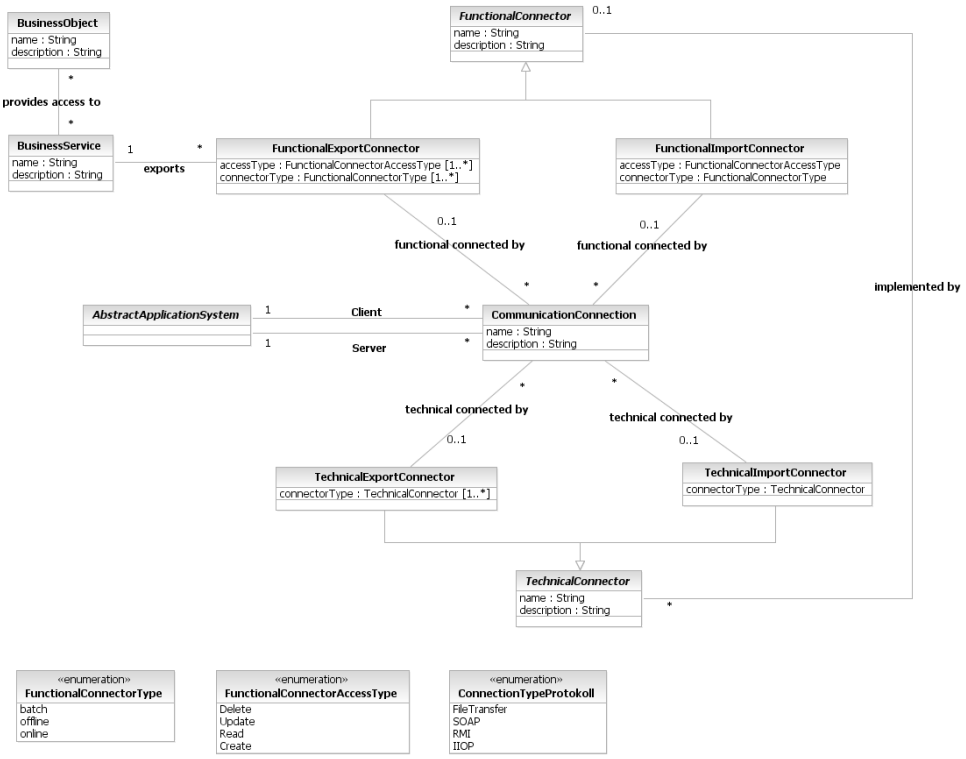


Abbildung 4.13.: Informationsmodellmuster für Kommunikationsbeziehungen zwischen Anwendungssysteme

Die Visualisierung der Export-Konnektoren ist an das aus UML [OM05d] und anderen graphischen Modellierungssprachen bekannte *Lollipop*-Symbol angelehnt, die Import-Konnektoren werden durch ein ausgefülltes Rechteck auf der Seite des Clients dargestellt (siehe Abbildung 4.12).

Die Berücksichtigung der Konnektoren und deren Visualisierung ermöglicht es beispielsweise, Transaktionsraten, laufende Projekte (welche Konnektoren verändern) oder Handlungsbedarf (Flaschenhälse etc.) darzustellen. Hierzu kann die Hintergrundfarbe der Konnektoren genutzt werden, um weitere Informationen darzustellen.

Das UML-Klassendiagramm in Abbildung 4.13 ist Teil eines Informationsmodellmusters zur Zusammenfassung der einzelnen Stufen und Konzepte, welches von Brendebach [Br05a] und Lauschke [La05b] angewendet wurde. In Abbildung 4.13 ist nicht berücksichtigt, ob Kommunikationsbeziehungen zwischen Anwendungssystemen, Anwendungssystemversionen oder installierten Anwendungssystemversionen erfasst werden (siehe hierzu auch Abschnitt 4.2.1); die abstrakte Klasse `AbstractApplicationSystem` dient als generisches Element.

Die Multiplizitäten an den Assoziationen lassen erkennen, dass die Erfassung der Konnektoren optional ist, da eine nachträgliche Aufnahme der Informationen möglich sein

soll. Für weitere Softwarekarten bzw. Viewpoints, die dieses Informationsmodellmuster anwenden, wird auf Matthes und Wittenburg [MW04b] verwiesen.

4.2.3.2. Informationsmodellfragment für Musterarchitekturen

Die große Zahl von verschiedenen Herstellern von Infrastrukturelementen (Datenbankmanagementsysteme, Applikationsserver, Webserver etc.) und die Zahl unterschiedlicher Softwarearchitekturen, die verschiedene Technologien einbeziehen, hat dazu geführt, dass Unternehmen ein Steuerungsinstrument benötigen, um die resultierende Komplexität zu managen.

In Abschnitt 2.1.5 wurde der Prozess *IT-Architekturmanagement* eingeführt, der mittels Musterarchitekturen und Musterlösungen Leitlinien und Unternehmensstandards für Anwendungssysteme definiert. Dabei werden die Begriffe Musterarchitektur und Musterlösung wie folgt verwendet:

Musterarchitektur : Eine Musterarchitektur (engl. *Architectural Blueprint*) beschreibt die abstrakte Architektur eines Anwendungssystems mittels verschiedener Sichten und Architekturbausteine. Die Sichten richten sich an verschiedene Stakeholder und dokumentieren unterschiedliche Elemente.

Musterlösung : Eine Musterlösung (engl. *Archtectural Solution*) instantiiert die Architekturbausteine einer Musterarchitektur durch Lösungsbausteine, die konkrete Produkte und Technologien referenzieren. Ein Anwendungssystem verwendet eine Musterlösung und die in ihr beschriebenen Lösungsbausteine.

Die Vielzahl von Produkten und Technologien, die für einen einzelnen Architekturbaustein (bspw. Applikationsserver) existieren, macht eine Unterscheidung zwischen Musterarchitekturen und Musterlösungen notwendig, um eine produkt- und technologieunabhängige Sicht – die Musterarchitektur – zu erhalten.

Unternehmen sind bestrebt, die Anzahl an Produkten für einen einzelnen Architekturbaustein gering zu halten, um den unnötigen Aufbau von Know-how in der Entwicklung sowie für Betrieb und Wartung zu reduzieren. Des Weiteren muss garantiert werden, dass die Kombination aus Lösungsbausteinen vom IT-Betrieb unterstützt wird. Ein Mittel, um diese Probleme zu adressieren, ist die Verwendung von Musterarchitekturen und Musterlösungen, die bewährte Architekturen und Kombinationen zusammenfassen.

Die Dokumentation von Musterarchitekturen erfolgt durch unterschiedliche Viewpoints, wie sie beispielsweise von Clements et al. [Cl02] oder Stangler [St03] beschrieben werden. Ein Beispiel für einen Viewpoint einer Musterarchitektur wurde in Abbildung 2.4 in Abschnitt 2.1.5 gegeben. Abbildung 2.4 zeigt die Stufensicht[18] einer Musterarchitektur.

[18] Im englischen wird zwischen *Tiers* und *Layers* unterschieden. Im deutschen Sprachgebrauch werden häufig beide Begriffe mit *Schicht* übersetzt. In dieser Arbeit wird *Tier* mit Stufe und *Layer* mit Schicht übersetzt. Zur Unterscheidung von *Tiers* und *Layers* wird auf Clements et al. [Cl02] verwiesen.

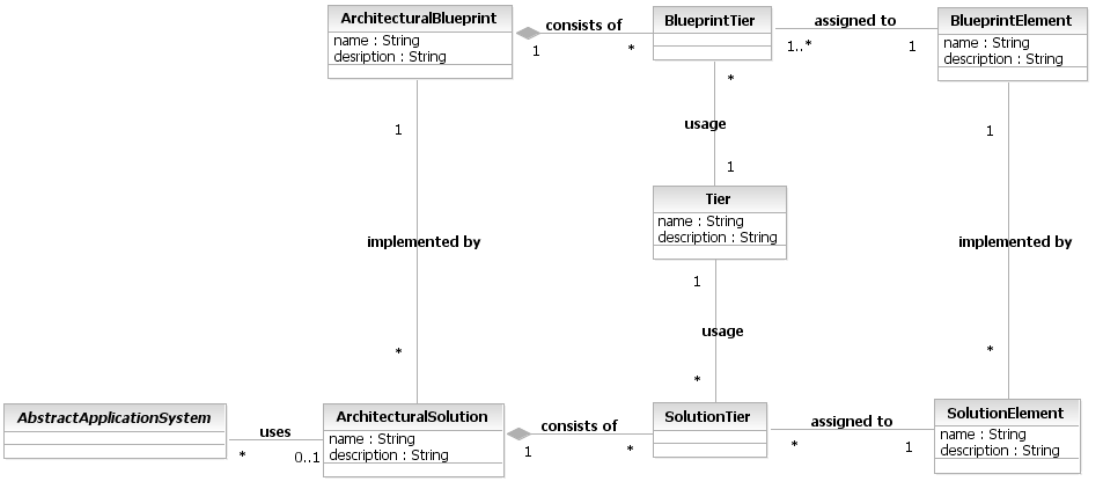


Abbildung 4.14.: Informationsmodellmuster für Musterarchitekturen

Weitere Viewpoints dokumentieren beispielsweise die Verteilung der Komponenten eines Anwendungssystems auf Hardwaresysteme inkl. der verwendeten Betriebssysteme oder die Verteilung der Hardwaresysteme auf unterschiedliche Netzwerksegmente.

Werden die wesentliche Bestandteile einer Stufensicht für Musterarchitekturen und Musterlösungen in einem Informationsmodellmuster zusammengefasst, ergibt sich das Modell in Abbildung 4.14: Eine Musterarchitektur (engl. *ArchitecturalBlueprint*) und eine Musterlösung (engl. *ArchitecturalSolution*) bestehen jeweils aus Stufen. Da für Musterarchitekturen und Musterlösungen nur eine gemeinsame Obermenge von Stufen existiert, nutzen beide die Klasse `Tier`, indem sie eigene Instanzen mit entsprechender Assoziationen über die Klassen `BlueprintTier` und `SolutionTier` bilden. Die Architekturbausteine (engl. *BlueprintElement*) und Lösungsbausteine (engl. *SolutionElement*) werden wiederum den einzelnen Schichten zugeordnet.

Beispiele für mögliche Stufen finden sich bei dem *Tiered Approach* von SUN Microsystems [SU01], der in Abbildung 4.15 dargestellt ist. Weitere Beispiele für mehrschichtige und mehrstufige Architekturen finden sich u. a. bei Bass et al. [BCK03].

Das Modell erlaubt es zu dokumentieren, welche Musterlösung von welchem Anwendungssystem genutzt wird. Durch das transitive Verfolgen der Assoziationen ist es möglich festzustellen, welche Lösungsbausteine von einem Anwendungssystem verwendet werden.

Werden die Elemente Musterarchitektur, Musterlösung, Architekturbaustein und Lösungsbaustein zusätzlich mit einem Versionsschlüssel, einer Vorgänger-Nachfolger-Beziehung und weiteren Informationen zum Lebenszyklus (Zeitraum Evaluierung, Zeitraum Produktion etc.) ergänzt, so können diese Informationen vom IT-Betrieb und IT-Projekten genutzt werden. Das Auslaufen des Supports für ein bestimmtes Datenbankmanagementsystem führt gleichzeitig zum Ende des Lebenszyklusses eines Lösungsbausteins und

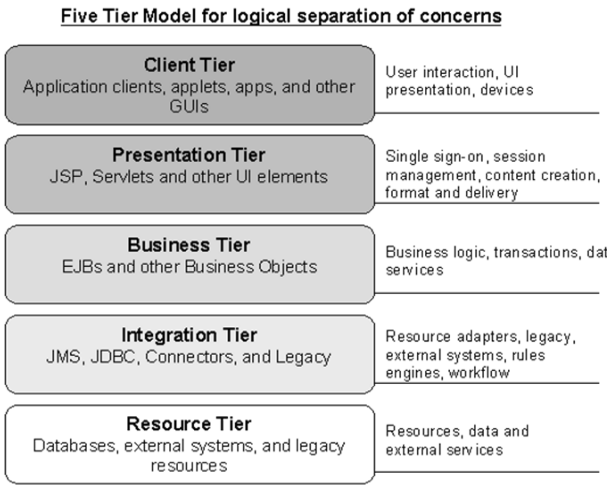


Abbildung 4.15.: *Tiered Approach* von SUN Microsystems [SU01]

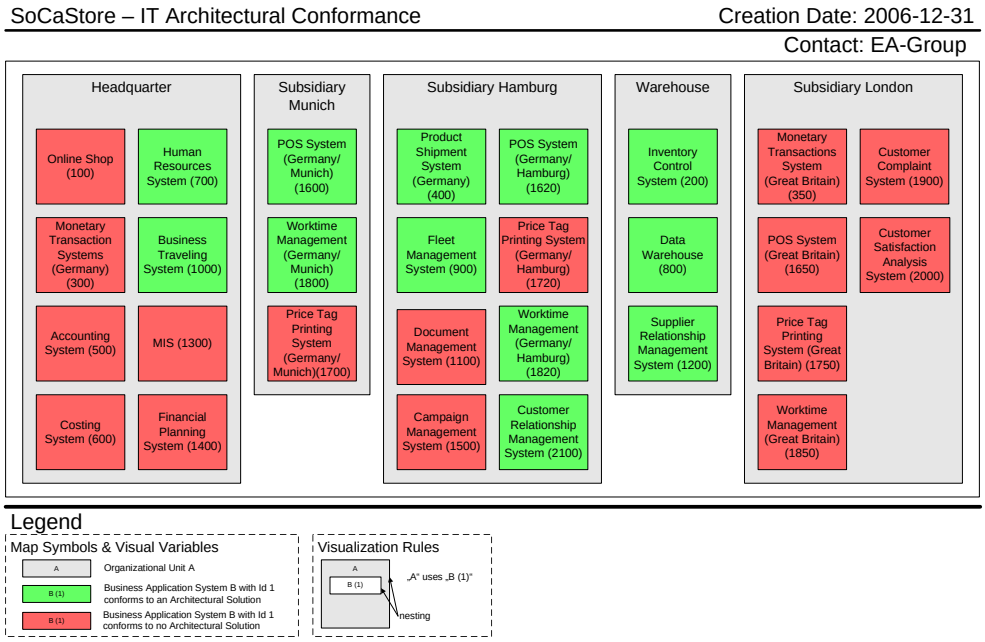


Abbildung 4.16.: Clusterkarte mit IT-Architekturkonformität von Anwendungssystemen

der verbundenen Musterlösung. Hieraus resultiert ein Handlungsbedarf, um rechtzeitig auf eine neue Version des Datenbankmanagementsystems oder einen anderen Hersteller zu wechseln. Intervallkarten (siehe Abschnitt 3.4.1.1.2) sind ein Hilfsmittel, um den Lebenszyklus einer Musterlösung und ihrer Lösungsbausteine adäquat darzustellen.

Ein weiteres Anwendungsbeispiel für die Softwarekartographie ist das Darstellen der IT-Architekturkonformität der Anwendungslandschaft entlang der Musterlösungen (siehe Abbildung 4.16). Hierbei wird die Konformität eines Anwendungssystems durch die Hintergrundfarbe (rot/grün) des korrespondierenden Rechtecks eines Anwendungssystems dargestellt.

Die beiden Beispiele für Muster in Informationsmodellen haben gezeigt, dass in Informationsmodellen Strukturen existieren, die flexibel eingesetzt werden können. Eine wesentlich Leistung des bzw. der Ersteller eines Informationsmodells liegt darin, die Interessen der Stakeholder zu analysieren und die entsprechenden Viewpoints, welche die Interessen adressieren, zu selektieren. Auf dieser Basis können die einzelnen Informationsmodellfragmente ausgewählt und anschließend in einem Gesamtmodell integriert werden.

4.3. Visualisierungsmodell für die Softwarekartographie

Nach der Diskussion des Informationsmodells als Metamodell des semantischen Modells wird in diesem Abschnitt das Visualisierungsmodell für Softwarekarten vorgestellt, welches das Metamodell für symbolische Modelle darstellt. Die in Abschnitt 3.3 vorgestellten Softwarekarten beinhalten durch die vorgestellten Gestaltungsmittel, -variablen und -regeln bereits Anforderungen an ein Visualisierungsmodell, die in diesem Abschnitt in einem geschlossenen Modell zusammengefasst werden.

Zusätzlich werden Anforderungen an die Visualisierung von Kennzahlen erhoben, die bisher nicht detailliert wurden. Abschnitt 4.3.1 beschreibt diese Anforderungen, bevor in Abschnitt 4.3.2 das Visualisierungsmodell vorgestellt wird. Abschnitt 4.3.3 zeigt abschließend, wie das Visualisierungsmodell für Softwarekarten angewendet wird.

4.3.1. Visualisierung von Kennzahlen

In Abschnitt 3.3.2 wurden Kennzahlen, die auf Softwarekarten visualisiert werden sollen, als ein Merkmal von Anwendungslandschaften vorgestellt. Dieser Abschnitt vervollständigt die Anforderungen hinsichtlich der Visualisierung von Kennzahlen an das Visualisierungsmodell, so dass mittels Gestaltungsmitteln und deren Gestaltungsvariablen Kennzahlen visualisiert werden können.

Zur Visualisierung von Kennzahlen bieten sich auf Softwarekarten zahlreiche Möglichkeiten an. Eine Kennzahl wird nach Kütz [Kü03, S. 41] als ein Merkmal verstanden, welches Sachverhalte in quantitativer und konzentrierter Form erfasst. Mögliche Kennzahlen, die auf Softwarekarten visualisiert werden, reichen von Kosten, über die Erfüllung von SLAs bis hin zu Transaktionsraten von Applikationen (siehe Anhang A).

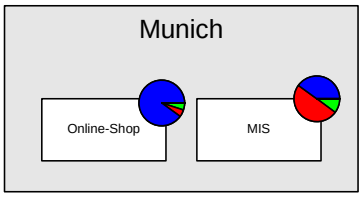


Abbildung 4.17.: Visualisierung der Implementierungssprache je Anwendungssystem

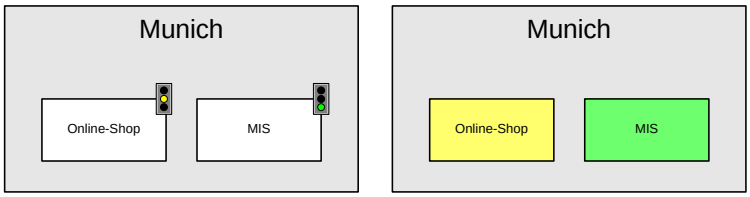


Abbildung 4.18.: Visualisierung der Erfüllung des SLAs je Anwendungssystem

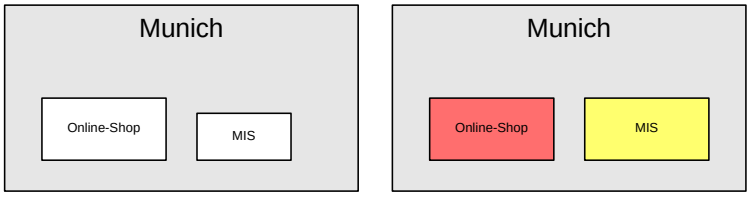


Abbildung 4.19.: Visualisierung der Betriebskosten je Anwendungssystem

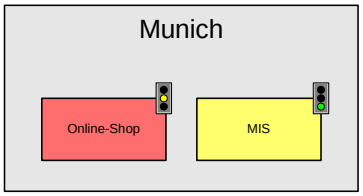


Abbildung 4.20.: Visualisierung der Betriebskosten und Erfüllung des SLAs je Anwendungssystem

Anwendungssystem	Java	C++	Pascal	SLA	Betriebskosten
Online-Shop	100%	5%	5%	95%	50.000
MIS	40%	50%	10%	99%	35.000

Tabelle 4.1.: Kennzahlen der Anwendungssysteme zur beispielhaften Visualisierung

Die Möglichkeiten zur Visualisierung von Kennzahlen unterscheiden sich zum einen nach dem Skalenniveau (Nominal, Ordinal, Intervall etc.) einer Kennzahl, der Skalenart (absolute Werte vs. relative Werte) und dem semantischen Gehalt einer Kennzahl. Im Folgenden sind einige Beispiele zur Visualisierung von Kennzahlen wiedergegeben, die vom Visualisierungsmodell unterstützt werden. Der Teil des semantischen Modells, welcher den Kennzahlen der Anwendungssysteme in den Abbildungen[19] zugrundeliegt, ist Tabelle 4.1 zu entnehmen.

Implementierungssprache je Anwendungssystem (Nominal- oder Verhältnisskala): Da ein Anwendungssystem eine logische Zusammenfassung von mehreren Komponenten darstellt, kann ein Anwendungssystem in mehreren Programmiersprachen implementiert sein. So kann ein Anwendungssystem zu 80% in Java, zu 15% in C++ und zu 5% in Pascal programmiert sein. Eine Kennzahl, welche diese Verteilung repräsentiert, kann beispielsweise durch ein Kreissektorendiagramm visualisiert werden. In diesem Kreissektorendiagramm spiegelt die Größe der einzelnen Kreissektoren den prozentualen Anteil einer Programmiersprache wider und die Hintergrundfarbe der Kreissektoren wird zur Unterscheidung der Programmiersprachen genutzt. Ein Beispiel für die Visualisierung dieser Kennzahl ist in Abbildung 4.17 dargestellt.

Erfüllung des SLAs je Anwendungssystem (Intervallskala): Die Erfüllung des Service-Level-Agreements eines Anwendungssystems wird typischerweise in einem Berichtszeitraum (z. B. Q1/2007) als prozentualer Wert angegeben. Zur Visualisierung dieser Kennzahl kann ein zusätzliches Symbol an einem Rechteck annotiert oder die Hintergrundfarbe eines Rechtecks, welches ein Anwendungssystem repräsentiert, verwendet werden. Als zusätzliches Symbol wird (häufig) eine Ampel verwendet, für die Intervalle definiert werden, um die Status Grün, Gelb und Rot zu unterscheiden. Bei der Nutzung des Hintergrunds bietet sich ebenfalls die Definition von Intervallen für die Hintergrundfarben Grün, Gelb und Rot an; alternativ kann ein Farbschema verwendet werden. Zur Erzeugung eines mehrstufigen Farbschemas kann beispielsweise der *ColorBrewer* von Brewer [Br05b] verwendet werden. Ein Beispiel für die Visualisierung dieser Kennzahl ist in Abbildung 4.18 dargestellt.

Betriebskosten je Anwendungssystem (Verhältnisskala): Die Betriebskosten von Anwendungssystemen können sowohl unter Anwendung der Hintergrundfarbe analog zur *Erfüllung des SLAs je Anwendungssystem* als auch unter Nutzung der Größe eines Rechtecks, welches ein Anwendungssystem symbolisiert, visualisiert werden. Ein größeres Rechteck repräsentiert hierbei höhere Betriebskosten. Ein Beispiel für die Visualisierung dieser Kennzahl ist in Abbildung 4.19 dargestellt.

Die Darstellung mehrerer Kennzahlen auf einer Softwarekarte ist möglich, indem die Kennzahlen auf mehrere Schichten verteilt werden, so dass ein *übereinanderlegen* von Schichten ermöglicht wird. Ein Beispiel, welches die Betriebskosten mittels der Hintergrundfarbe und die Erfüllung des SLAs mittels Ampeln visualisiert, findet sich in Abbildung 4.20. Weitere Möglichkeiten zur Visualisierung von Kennzahlen auf Softwarekarten werden von Beyer [Be04] beschrieben.

[19] In den Abbildungen wird auf eine vollständige Legende verzichtet, da sie Ausschnitte aus Softwarekarten darstellen.

4.3.2. Objektorientiertes Visualisierungsmodell

Ziel des objektorientierten Visualisierungsmodell ist es, eine objektorientierte Beschreibung der abstrakten Syntax von Softwarekarten zu erhalten. Die konkrete Syntax von Softwarekarten entsteht durch eine natürlichsprachliche Beschreibung der Anwendung bzw. Darstellung der Elemente der abstrakten Syntax, die im Weiteren die abstrakte Syntax ergänzt.

Die Modellierung des Visualisierungsmodells erfolgt, wie auch bei den Informationsmodellfragmenten, mittels UML Klassendiagrammen [OM05c], wobei die Elemente der Diagramme der Semantik von MOF [OM06d] entsprechen. Eine Klasse repräsentiert somit eine (linguistische) Instanz der Klasse `MOF::Constructs::Class`.

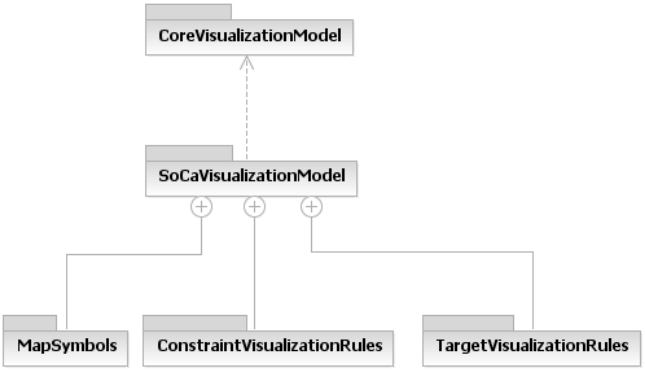


Abbildung 4.21.: Paketstruktur des Visualisierungsmodells

Die Paketstruktur des Visualisierungsmodells ist in Abbildung 4.21 dargestellt und besteht aus den folgenden Paketen, die im Anschluss detailliert werden:

CoreVisualizationModel: Das Paket `CoreVisualizationModel` definiert abstrakte Basiselemente des Visualisierungsmodells.

SoCaVisualizationModel: Das Paket `SoCaVisualizationModel` konkretisiert die Klassen des Pakets `CoreVisualizationModel` für Softwarekarten.

MapSymbols: Das Paket `MapSymbols` ist Teil des Paketes `SoCaVisualizationModel` und definiert die Gestaltungsmittel für Softwarekarten.

ConstraintVisualizationRules: Das Paket `ConstraintVisualizationRules` ist Teil des Paketes `SoCaVisualizationModel` und definiert die Muss-Gestaltungsregeln für Softwarekarten.

TargetVisualizationRules: Das Paket `TargetVisualizationRules` ist Teil des Paketes `SoCaVisualizationModel` und definiert die Ziel-Gestaltungsregeln für Softwarekarten.

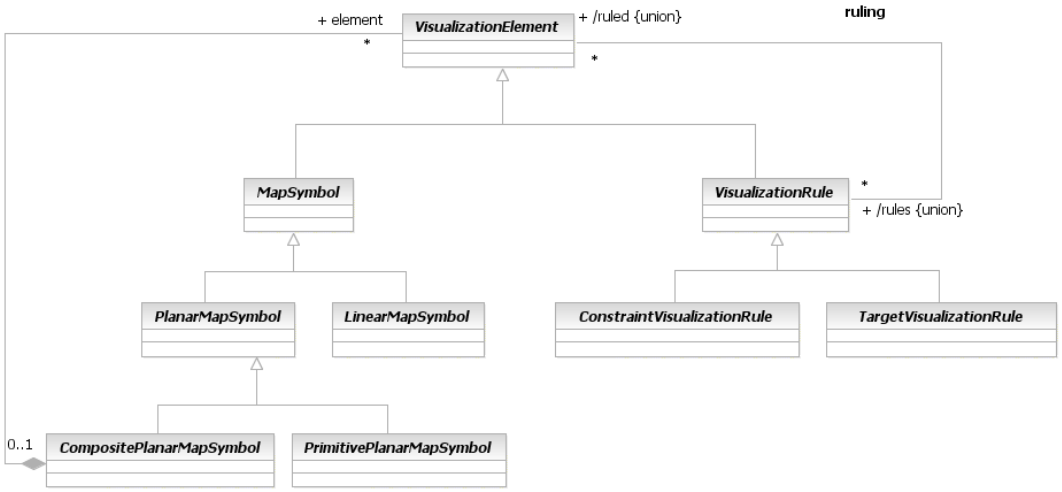


Abbildung 4.22.: Visualisierungsmodell: Paket `CoreVisualizationModel`

4.3.2.1. Visualisierungsmodell: Paket CoreVisualizationModel

Abbildung 4.22 zeigt die Elemente des Kernpakets `CoreVisualizationModel`, die alle *abstrakt* und somit nicht instantiierbar sind. Die Bedeutung der Elemente ist wie folgt:

`VisualizationElement` repräsentiert die Klasse von Visualisierungselementen, aus denen ein Diagramm zusammengesetzt wird. Dies schließt sowohl Gestaltungsmittel (`MapSymbol`) als auch Gestaltungsregeln (`VisualizationRule`) ein.

`MapSymbol` repräsentiert die Superklasse der graphischen und *erfahrbaren* Elemente einer Visualisierung, die Gestaltungsmittel.

`PlanarMapSymbol` repräsentiert die flächenartigen Gestaltungsmittel.

`LinearMapSymbol` repräsentiert die linienartigen Gestaltungsmittel. Instanzen sind beispielsweise Linien mit verschiedenen Pfeilenden und mehreren Knickpunkten.

`PrimitivePlanarMapSymbol` repräsentiert die einfachen, flächenartigen Gestaltungsmittel, die nicht zusammengesetzt werden. Instanzen sind je nach Implementierung beispielsweise Rechtecke, Kreise oder Chevren.

`CompositePlanarMapSymbol` repräsentiert die zusammengesetzten, flächenartigen Gestaltungsmittel und besteht aus einer Komposition von `VisualizationElements`. Ein Diagramm, welches im Visualisierungsmodell durch keine eigenständige Klasse repräsentiert wird, ist ebenso ein `CompositePlanarMapSymbol`.

`VisualizationRule` repräsentiert die Superklasse der Gestaltungsregeln.

`ConstraintVisualizationRule` repräsentiert die Klasse der Muss-Gestaltungsregeln (siehe auch Abschnitt 3.3). Muss-Gestaltungregeln müssen erfüllt sein, um die Semantik der Informationsobjekte durch das symbolische Modell korrekt darzustellen.

`TargetVisualizationRule` repräsentiert die Klasse der Ziel-Gestaltungsregeln (siehe auch Abschnitt 3.3). Ziel-Gestaltungsregeln optimieren die Darstellung der Informationsobjekte durch das symbolische Modell. Eine mögliche Subklasse minimiert beispielsweise die Größe eines flächenartigen Gestaltungsmittels.

`ruling` repräsentiert eine bidirektionale, abgeleitete (engl. *derived*) Assoziation. Diese Assoziation ermöglicht es, sowohl für eine Gestaltungsregel die beeinflussten Visualisierungselemente festzustellen als auch von einem Visualisierungselement die beeinflussenden Gestaltungsregeln zu erfragen. Die Ableitung entsteht durch die Realisierung dieser Beziehung in importierenden Paketen, die Untermengen dieser Assoziation bilden.

Eine Implementierung des Pakets `CoreVisualizationModel` ist nicht auf Softwarekarten eingeschränkt, andere Implementierungen für beispielweise UML sind denkbar, um die informelle Definition der abstrakten Syntax von UML zu verbessern[20]. Im Weiteren wird die Implementierung des `CoreVisualizationModel` für Softwarekarten gezeigt.

4.3.2.2. Visualisierungsmodell: Paket SoCaVisualizationModel

Zur Beschreibung von Softwarekarten dient das Paket `SoCaVisualizationModel`, welches das Paket `CoreVisualizationModel` importiert (siehe Abbildung 4.21). Die einzelnen Klassen des `SoCaVisualizationModel` erben[21] bei Namensgleichheit von den Klassen des `CoreVisualizationModel` und überschreiben diese.

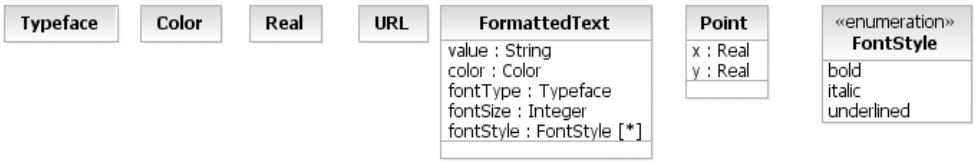


Abbildung 4.23.: Visualisierungsmodell: Datentypen

Für das `SoCaVisualizationModel` werden einige Datentypen benötigt (siehe Abbildung 4.23), die wie folgt definiert sind:

`Typeface` entspricht einer Schriftart, wie beispielsweise *Arial* oder *Times*.

`Color` entspricht einer Farbe, deren Repräsentation je nach der verwendeten Codierung (bspw. RGB oder CMYK) variieren kann.

`Real` entspricht der Klasse, deren Instanzen in der Menge der reellen Zahlen ($\mathbb{R}$) liegen.

`URL` entspricht dem Datentyp zur Beschreibung des Aufbaus von *Uniform Resource Locators*, wie es der RFC3986 [BLF05] definiert.

[20] Eine Diskussion der abstrakten und konkreten Syntax von UML folgt in Abschnitt 4.5.1.
[21] Diese Vererbung ist in den Abbildungen nicht enthalten.

`FormattedText` entspricht einem formatierten Text, dessen Attribute der Wert, die Farbe, die Schriftart, die Schriftgröße und der Schriftstil sind.

`Point` entspricht der Klasse deren Instanz ein Punkt in einer Fläche mit x- und y-Koordinaten ist. Die Koordinaten eines Punktes besitzen als Datentyp `Real`, um die Formulierung des Layouting als Optimierungsproblem zu ermöglichen, welches nicht in die Klasse der ganzzahligen Optimierung fällt (siehe Abschnitt 4.4.2). Würde `Integer` als Datentyp gewählt, so würde sich der Lösungsraum verändern.

`FontStyle` definiert als Enumeration den Stil eines formatierten Textes über die Literale fett (`bold`), kursiv (`italic`) und unterstrichen (`underlined`). Da Kombinationen der Literale zulässig sind, definiert `FormattedText` ein mehrdimensionales Attribut[22], welches die Kombinationen aufnimmt.

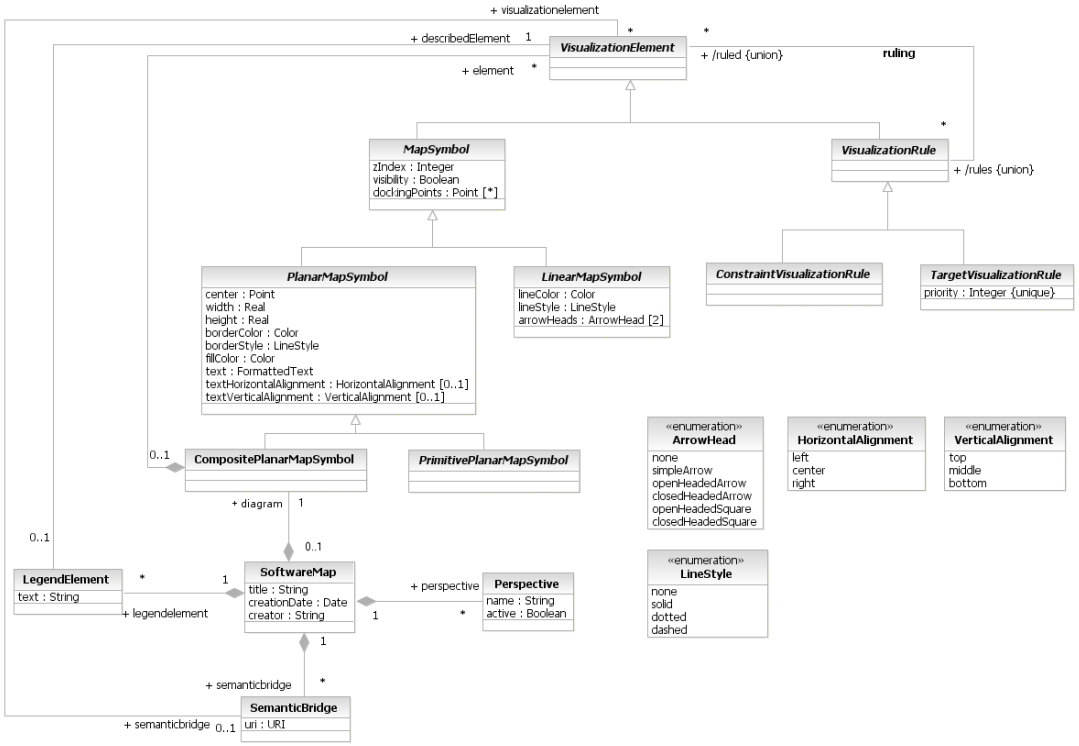


Abbildung 4.24.: Visualisierungsmodell: Paket `SoCaVisualizationModel`

Die Elemente des `SoCaVisualizationModel` sind in Abbildung 4.24 dargestellt und erweitern das `CoreVisualizationModel` wie folgt:

`MapSymbol` wird erweitert um die Attribute `zIndex`, `visibility` und ein mehrdimensionales Attribut `dockingPoints`. Der `zIndex` gibt die Reihenfolge der Elemente entlang der z-Achse einer Darstellung an und definiert somit, welches Element *vor*

[22] In UML wird die Multiplizität von Attributen in eckigen Klammen („[“, „]“) angegeben. Ist keine Multiplizität angegeben, so entspricht dies einer Multiplizität von 1.

bzw. *hinter* einem anderen liegt. Die `visibility` definiert, ob eine Gestaltungsmittelinstanz sichtbar oder ausgeblendet ist. Das mehrdimensionale Attribut `dockingPoints` definiert für eine Gestaltungsmittelinstanz die Verknüpfungspunkte, die relativ zum Mittelpunkt angegeben werden und von der Muss-Gestaltungsregel `Attachment` verwendet werden.

`PlanarMapSymbol` wird erweitert um die Attribute

- `center` für den Mittelpunkt eines flächenartigen Gestaltungsmittels,
- `width` und `height` für die Breite bzw. Höhe eines flächenartigen Gestaltungsmittels,
- `borderColor` und `fillColor` für die Rahmen- bzw. Füllfarbe,
- `borderStyle` für die Linienart des Rahmens sowie
- `text`, `textHorizontalAlignment` und `textVerticalAlignment` für eine Beschriftung und die Ausrichtung derselben.

`LinearMapSymbol` wird erweitert um die Attribute `lineColor` zur Definition der Linienfarbe, `lineStyle` zur Definition der Linienart und einem mehrdimensionalen Attribut (Multiplizität zwei) mit `ArrowHead`s, zur Definition der Pfeilenden.

`SoftwareMap` repräsentiert die Klasse für Softwarekarten und besitzt die Attribute `title` (Titel), `creationDate` (Erstellungsdatum) und `creator` (Ersteller). Eine Softwarekarte besteht des Weiteren aus einem `CompositePlanarMapSymbol`, welches das Diagramm als Inhalt des Kartenfelds (siehe Abschnitt 3.4.3) repräsentiert, und mehreren `LegendElement`s, welche die einzelnen Elemente einer Legende repräsentieren sowie mehreren `Perspective`s. Die zusätzlichen `SemanticBridge`s verbinden Visualisierungselementinstanzen mit den Informationsobjekten des semantischen Modells.

`LegendElement` repräsentiert die Klasse der Legendenelemente, die als Attribut einen `text` (Beschriftung) besitzen und Instanzen von Visualisierungselementen beschreiben.

`Perspective` repräsentiert die Klasse der Perspektiven einer Software, die definiert, welche Schichten (siehe Abschnitt 3.4.2) zu einem Viewpoint zusammengesetzt werden. Eine Perspektive besitzt als Attribute einen Namen (`name`) und die Eigenschaft, ob sie aktiv (`active`) ist. Eine aktive Perspektive zeigt die zugehörigen Schichten an, eine nicht-aktive blendet diese Schichten aus, wenn sie nicht Teil einer anderen Perspektive sind.

`SemanticBridge` repräsentiert eine *Brücke* zwischen einer Instanz eines Visualisierungselements und einem Informationsobjekt des semantischen Modells. Hierzu referenziert die `SemanticBridge` über das Attribut `uri`, ein Uniform Resource Identifier (RFC3986 [BLF05]), ein Informationsobjekt.

`TargetVisualizationRule` wird erweitert um das Attribut `priority`, welches es ermöglicht eine Reihenfolge der Ziel-Gestaltungsregeln hinsichtlich der Priorität festzulegen (siehe Abschnitt 4.3.2.5).

Die Enumerationen (Klassen mit dem Stereotyp `enumeration`) dienen zur Definition von Pfeilenden (`ArrowHead`) für Linien, zur horizontalen Ausrichtung (`Horizontal Alignment`) und zur vertikalen Ausrichtung (`VerticalAlignment`) von Texten sowie zur Definition von unterschiedlichen Linienarten (`LineStyle`).

4.3.2.3. Visualisierungsmodell: Paket MapSymbols

Das Paket `MapSymbols` konkretisiert die Klasse `CompositePlanarMapSymbol` sowie die abstrakten Klassen `PrimitivePlanarMapSymbol` und `LinearMapSymbol` für Gestaltungsmittel, so dass instantiierbare Klassen entstehen, die auf Softwarekarten verwendet werden. Die Unterscheidung von flächenartigen und linienartigen Gestaltungsmitteln begründet sich in der Definition der Gestaltungsregeln, die in den Abschnitten 4.3.2.4 und 4.3.2.5 beschrieben werden. Es ist beispielsweise nicht sinnvoll, ein Rechteck in einer Linie zu verschachteln, so dass diese Unterscheidung notwendig wird.

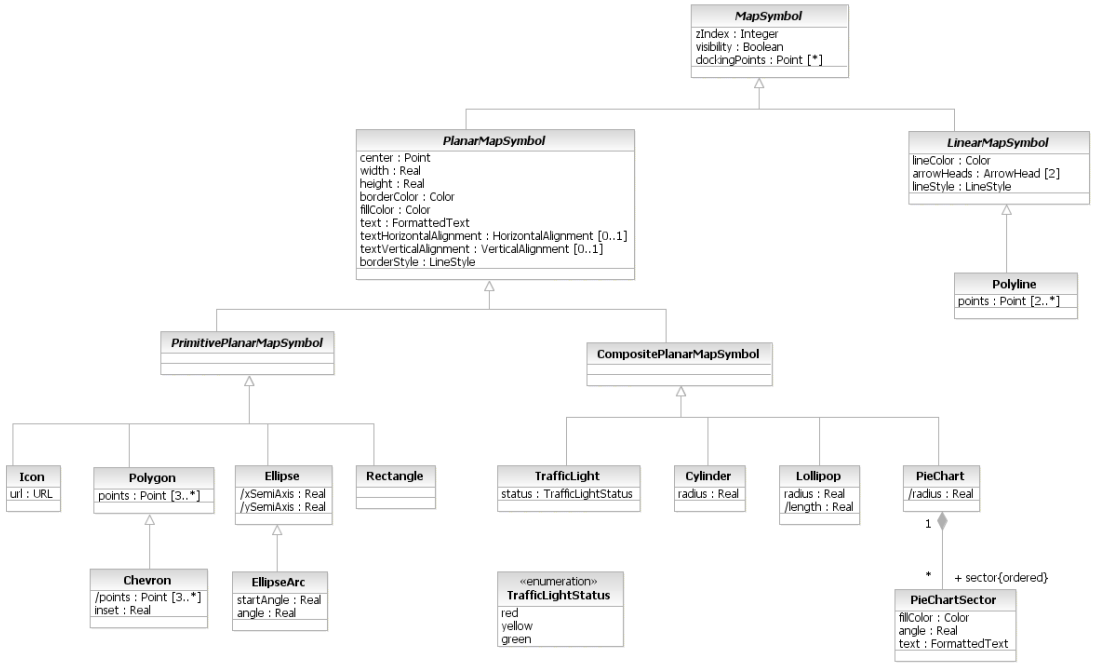


Abbildung 4.25.: Visualisierungsmodell: Paket `SoCaVisualizationModel`::`MapSymbols`

Die derzeit auf Softwarekarten verwendeten Gestaltungsmittel sind Abbildung 4.25 zu entnehmen. Prinzipiell sind weitere Gestaltungsmittel denkbar, die unter Umständen nicht durch das Zusammensetzen von existierenden Gestaltungsmitteln gebildet werden können. Hierzu muss dieses Paket entsprechend erweitert werden.

Für die im Paket `MapSymbols` eingeführten Gestaltungsmittel, die sich von `Primitive`

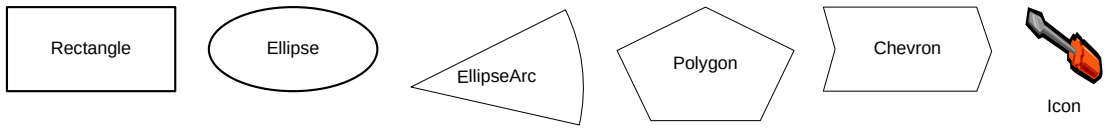


Abbildung 4.26.: Beispiele für primitive Gestaltungsmittel aus Abbildung 4.25

`PlanarMapSymbol` ableiten, sind in Abbildung 4.26 Beispiele zu finden. Diese einzelnen Gestaltungsmittel sind wie folgt definiert:

`Rectangle` repräsentiert ein Rechteck. Die von `PlanarMapSymbol` geerbten Attribute sind ausreichend, um ein Rechteck zu beschreiben, so dass `Rectangle` keine weitere Attribute definiert.

`Ellipse` repräsentiert eine Ellipse und definiert zusätzlich die abgeleiteten Attribute `xSemiAxis` (Halbachse in x-Richtung) und `ySemiAxis` (Halbachse in y-Richtung), die aus der Breite bzw. Höhe (geerbte Attribute) berechnet werden.

`EllipseArc` repräsentiert einen Ellipsenbogen, der sich von `Ellipse` ableitet, und zusätzlich die Attribute `startAngle` und `angle` definiert. Der Ellipsenbogen ergibt sich, wenn beginnend beim Startwinkel (`startAngle`) der Ellipsenbogen mit dem Winkel (`angle`) im Uhrzeigersinn aufgebaut wird.

`Polygon` repräsentiert ein Polygon, welches durch das mehrdimensionale und geordnete[23] Attribut von Punkten (`points`) aufgebaut wird.

`Chevron` ist eine spezielle Form eines Polygons, welches das von `Polygon` geerbte mehrdimensionale Attribut `points` als abgeleitetes Attribut überschreibt, da dies über die Breite (`width`), Höhe (`height`) und das neu definierte Attribut `inset` (*Knickgröße*) berechnet wird. Der `inset` beschreibt die Distanz von der gedanklichen vertikalen Linie an einer Seite eines Chevrons zum Knickpunkt, wodurch das Einknicken bzw. Herausknicken der Ecken entsteht.

`Icon` repräsentiert ein graphisches Symbol, welches durch eine URL (`url`) referenziert wird.

Neben den primitiven, flächenartigen Gestaltungsmitteln werden einige zusammengesetzte, flächenartige Gestaltungsmittel (`CompositePlanarMapSymbol`) definiert, die häufig in Softwarekarten verwendet werden. Das Einführen dieser Elemente vereinfacht die Beschreibung der Syntax einer Softwarekarte, da mehrfach verwendete Elemente nicht explizit zusammengesetzt werden müssen. Abbildung 4.27 zeigt diese zusammengesetzten flächenartigen Gestaltungsmittel, die wie folgt definiert sind:

`TrafficLight` repräsentiert das Symbol einer Ampel, welches in der einfachsten Form aus einem Rechteck und drei Kreisen zusammengesetzt ist, und dessen Attribut `status` mit einem Literal der Enumeration `TrafficLightStatus` (rot, gelb, grün) belegt werden kann (siehe auch Abschnitt 4.3.1).

[23] Die Eigenschaft `ordered` des Attributs `points` ist in Abbildung 4.25 nicht enthalten.

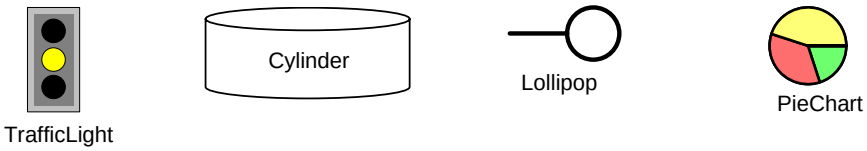


Abbildung 4.27.: Beispiele für zusammengesetzte flächenartige Gestaltungsmittel aus Abbildung 4.25

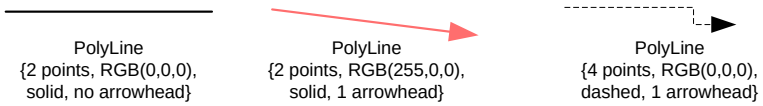


Abbildung 4.28.: Beispiele für linienartige Gestaltungsmittel aus Abbildung 4.25

`Cylinder` ist ein Zylinder, der in der Geometrie als gerader Kreiszylinder bezeichnet wird und das Attribut `radius` (Radius) definiert. Der `radius` beschreibt die zwei y-Halbachsen des Zylinders am oberen und unteren Ende.

`Lollipop` ist das Lollipop-Symbol, welches aus einem Kreis und einer Linie besteht, und definiert das Attribut `radius` (Radius des Kreises) und das abgeleitete Attribut `length` (Länge des Stiels). Das Attribut `length` wird aus dem geerbten Attribut `width` sowie dem Attribut `radius` berechnet. Für Beispiele der Verwendung dieses Gestaltungsmittels siehe auch Abschnitt 4.2.3.1.

`PieChart` ist ein Kreissektorendiagramm, welches das abgeleitete Attribut `radius` (Radius) definiert, welches aus den geerbten Attributen `width` und `height` berechnet wird, und aus einer geordneten Menge von Kreissektoren besteht. Die Kreissektoren (`PieChartSector`) besitzen die Attribute `fillColor`(Füllfarbe), `angle` (Winkel) und `text` (Beschriftung).

Das einzige für Softwarekarten verwendete linienartige Gestaltungsmittel ist die Polylinie (`Polyline`), die sich von `LinearMapSymbol` ableitet und zusätzlich als Attribut für die Knickpunkte ein mehrdimensionales und geordnetes[24] Attribut (`points`) definiert, wobei Start- und Endpunkt ebenso Knickpunkte repräsentieren. Abbildung 4.28 zeigt Beispiele für `LinearMapSymbol`s.

4.3.2.4. Visualisierungsmodell: Paket ConstraintVisualizationRules

In den Abschnitten 3.3 und 3.4 wurden Muss- und Ziel-Gestaltungsregeln exemplarisch eingeführt. Dieser Abschnitt zeigt, wie die Muss-Gestaltungsregeln in das Visualisierungsmodell einfließen. Eine Muss-Gestaltungsregel (engl. *Constraint Visualization Rule*) ist eine Gestaltungsregel, deren Instanzen eine graphische Beziehung zwischen Gestaltungsmittelinstanzen beschreiben, die erfüllt sein muss, um eine semantisch korrekte Darstellung zu erhalten (siehe Abschnitt 3.3).

[24] Die Eigenschaft `ordered` des Attributs `points` ist in Abbildung 4.25 nicht enthalten.

Die Muss-Gestaltungsregeln sind in vier UML-Klassendiagramme aufgeteilt. Ist in einem Diagramm ein Rollenende mit dem Zusatz `{subsets ruled}` und das andere Ende der binären Assoziation mit `{subsets rules}` beschriftet, so implementiert diese Assoziation die abgeleitete Assoziation `ruling` (siehe Abbildung 4.22).

Abbildung 4.29 enthält die Muss-Gestaltungsregeln, die in den einzelnen Softwarekartentypen unterschiedlich verwendet werden und wie folgt definiert sind:

`FullXSpecificIntersection` verlangt, dass die mit der Rolle `xIntersectedSymbol` assoziierte Instanz innerhalb der x-Koordinaten des `xIntersectingSymbol`s liegt. Bei einer Prozessunterstützungskarte wird beispielsweise das Rechteck für ein Anwendungssystem `InventoryControlSystem:ApplicationSystem` unterhalb des Chevrons für den Geschäftsprozess `Acquisition:BusinessProcess` positioniert werden, um die Prozessunterstützung auszudrücken.

`FullYSpecificIntersection` verlangt analog zur `FullXSpecificIntersection` eine Positionierung entlang der y-Achse. Die Positionierung auf einer Prozessunterstützungskarte von Rechtecken für Anwendungssysteme entsprechend der Prozessunterstützung bei bestimmten Organisationseinheiten wird beispielsweise durch diese Regel gewährleistet.

`Nesting` verlangt, dass die mit der Rolle `nested` assoziierte Gestaltungsmittelinstanz innerhalb der `nesting` Gestaltungsmittelinstanz liegt. Eine Clusterkarte nutzt diese Gestaltungsregeln, um beispielsweise das Rechteck für das Anwendungssystem `OnlineShop:ApplicationSystem` innerhalb des Rechtecks für den Standort `Munich:Location` zu positionieren, wodurch der Betriebsort ausgedrückt wird.

`XSequence` verlangt, dass die mit der geordneten (`ordered`) Rolle `ruledXSequence` assoziierten Gestaltungsmittelinstanzen von links nach rechts auf der Softwarekarte positioniert werden und die y-Position des Mittelpunkts der Gestaltungsmittelinstanzen gleich ist. Diese Gestaltungsregel wird beispielsweise angewendet, um die Geschäftsprozesse auf einer Prozessunterstützungskarte nebeneinander zu positionieren.

`YSequence` verlangt analog zur `XSequence` eine Ordnung entlang der y-Achse. Auf einer Prozessunterstützungskarte werden beispielsweise die Organisationseinheiten derart untereinander positioniert.

Abbildung 4.30 zeigt die Muss-Gestaltungsregeln, die genutzt werden, um bestimmte Attribute von Visualisierungselementen zu beeinflussen:

`IdentityOfProjection` verlangt, dass die mit der Rolle `ruledIdentical` assoziierten Instanzen von Visualisierungselementen bezüglich der in der Gestaltungsregel angegebenen Attribute (`constrainedAttributes`)[25] die gleichen Werte besitzen. Diese Regel wird beispielsweise angewendet, um Rechtecke für Anwendungssysteme mit der gleichen Breite darzustellen.

[25] Die `constrainedAttributes` sind vom Typ `MOF::Property` und werden einem `Classifier` über eine Assoziation mit der abgeleiteten Rolle `attribute` zugeordnet.

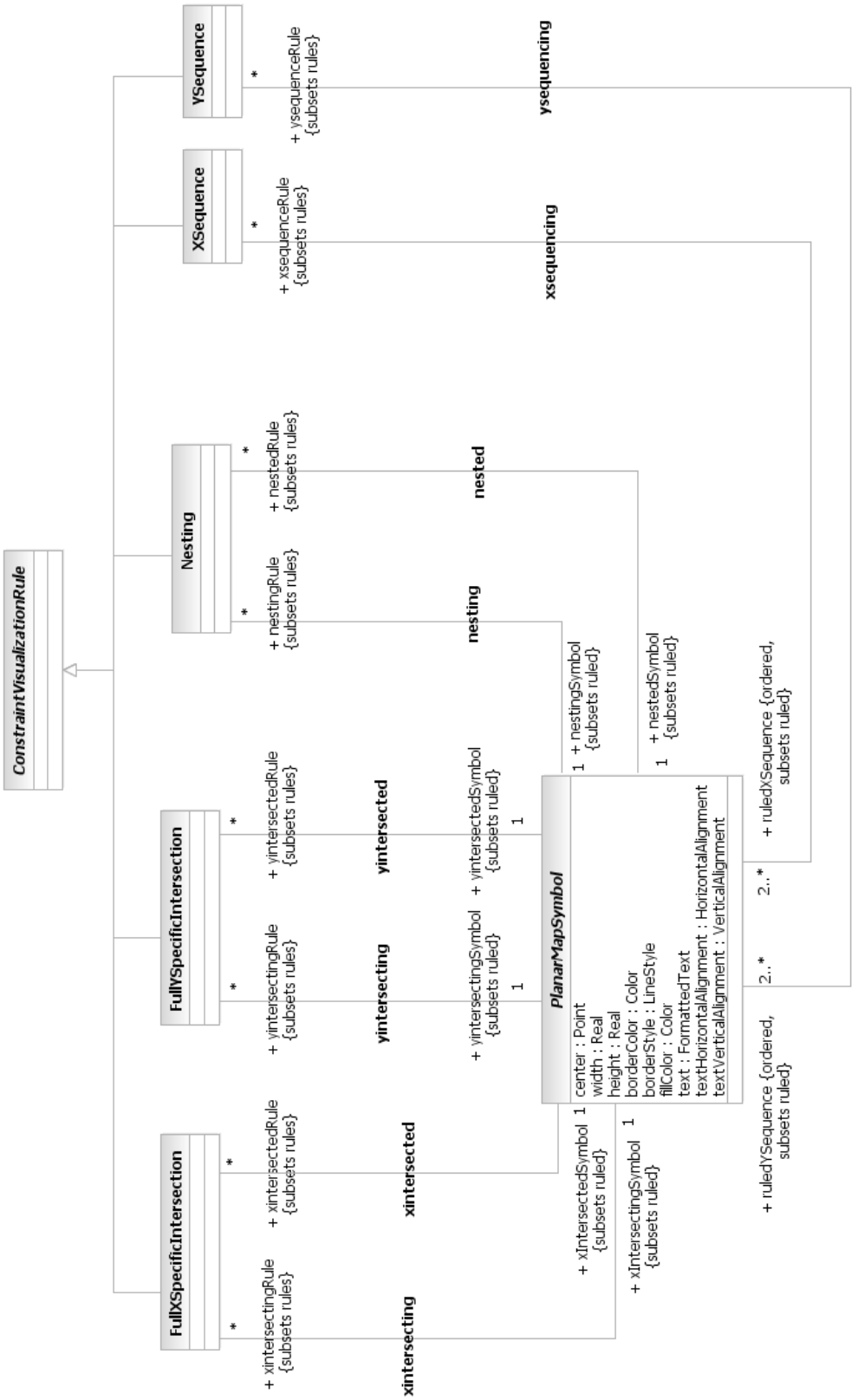


Abbildung 4.29.: Visualisierungsmodell: Paket `SoCaVisualizationModel::ConstraintVisualizationRules` – Diagramm 1

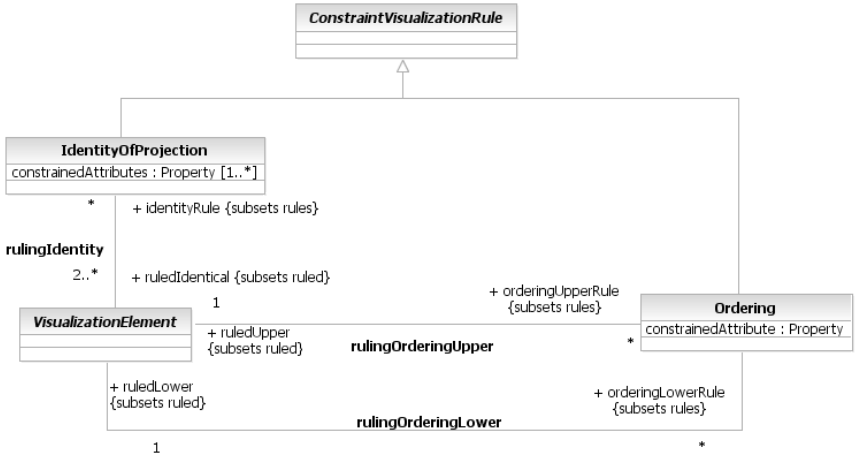


Abbildung 4.30.: Visualisierungsmodell: Paket `SoCaVisualizationModel`::`Constraint VisualizationRules` – Diagramm 2

`Ordering` verlangt, dass die mit der Rolle `ruledLower` assoziierte Instanz eines Visualisierungselements einen niedrigen Wert als die mit der Rolle `ruledUpper` assoziierte Instanz bezüglich eines in der Gestaltungsregel angegebenen Attributs (`constrainedAttribute`) besitzt. Diese Regel wird beispielsweise angewendet, um Rechtecke für Cluster in einer bestimmten Reihenfolge bezüglich der x-Achse zu positionieren, jedoch keine Reihung wie bei einer `XSequence` entstehen zu lassen.

Die Muss-Gestaltungsregeln, um das separierte Darstellen von Gestaltungsmitteln zu beschreiben und das Verknüpfen bzw. das Aneinanderhängen von Gestaltungsmitteln zu ermöglichen, sind in Abbildung 4.31 dargestellt:

`Separation` verlangt, dass die assoziierten Gestaltungsmittel voneinander separiert dargestellt werden. Separiert meint, dass

- bei zwei flächenartigen Gestaltungsmitteln die Flächen der Gestaltungsmittel höchstens einen gemeinsamen Punkt haben dürfen,
- bei zwei linienartigen Gestaltungsmittel keine Strecke innerhalb der Kantenzüge gleich sein darf, ein Kreuzen jedoch erlaubt ist und
- bei einem flächenartigen und einem linienartigen Element lediglich ein Berühren (wie bei einer Tangente) erlaubt ist.

Die Ausnahme, dass nur abzählbar viele Punkte zu beiden Gestaltungsmitteln gemeinsam gehören dürfen, beruht auf der Tatsache, dass ein Verknüpfen (siehe Regel `Attachment`) erlaubt sein muss, auch wenn die Regel `Separation` angewendet wird. Darüber hinaus sind kreuzungsfreie (planare) Graphen auf Grund der Position der Elemente häufig nicht möglich. Die Muss-Gestaltungsregel `Separation` wird angewendet, um beispielsweise von zwei Rechtecken für Anwendungssysteme, die innerhalb eines Cluster liegen, zu verlangen, dass diese sich nicht überschneiden.

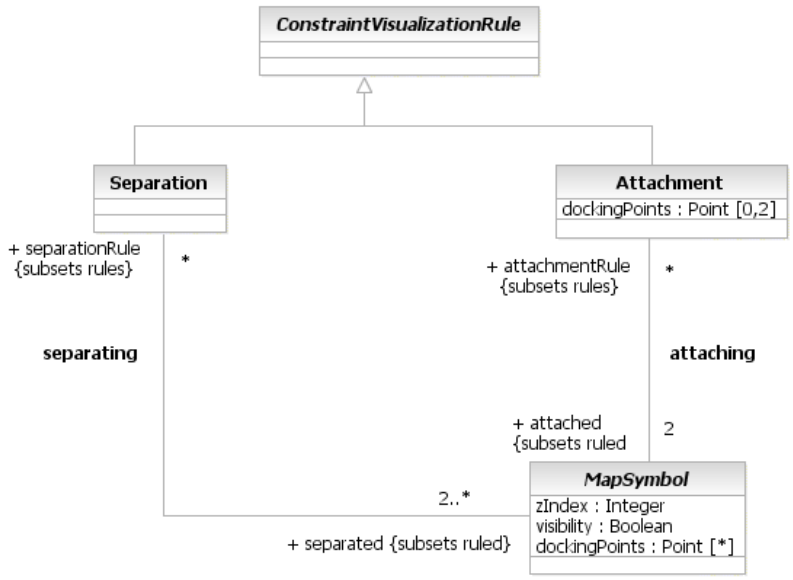


Abbildung 4.31.: Visualisierungsmodell: Paket `SoCaVisualizationModel`::`Constraint VisualizationRules` – Diagramm 3

`Attachment` wird genutzt, um Gestaltungsmittel miteinander zu verknüpfen. Hier wird entweder von jedem der beiden Gestaltungsmittel ein Verknüpfungspunkt angegeben, um damit die Position des Verknüpfens zu definieren, oder es wird ohne Verknüpfungspunkte verlangt, dass die beiden Gestaltungsmittel einen gemeinsamen Punkt in der Darstellung besitzen. Die Gestaltungsregel `Attachment` wird beispielsweise angewendet, um eine Linie für einen Datenfluss an ein Rechteck für ein Anwendungssystem oder einen Lollipop für einen Konnektor an das Rechteck des exportierenden Anwendungssystems zu knüpfen.

Das Schichtenprinzip für Softwarekarten (siehe Abschnitt 3.4.2) wird durch die Muss-Gestaltungsregeln aus Abbildung 4.32 realisiert:

`Layer` ist die abstrakte Basisklasse für die zwei abgeleiteten Klassen `ZLayer` und `VisibilityLayer`. Die Assoziation `layering` zwischen `Layer` und `MapSymbol` definiert die von einem `Layer` beeinflussten Gestaltungsmittel.

`ZLayer` verlangt, dass alle assoziierten Instanzen von Gestaltungsmitteln hinsichtlich des Attributs `zindex` den gleichen Wert besitzen. Die beeinflussten Gestaltungsmittelinstanzen befinden sich somit auf der gleichen *z-Schicht*.

`VisibilityLayer` verlangt, dass alle assoziierten Instanzen eines Gestaltungsmittels hinsichtlich des Attributs `visibility` den gleichen Wert besitzen. Die beeinflussten Gestaltungsmittelinstanzen sind somit alle entweder sichtbar oder ausgeblendet. Ist ein `VisibilityLayer` darüber hinaus einer oder mehreren Perspektiven (`Perspective`) zugeordnet, so wird die Sichtbarkeit zusätzlich über die Perspektive gesteuert.

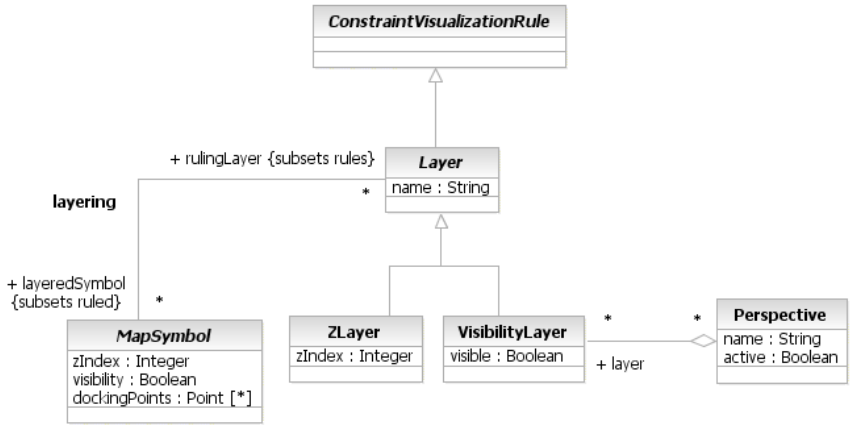


Abbildung 4.32.: Visualisierungsmodell: Paket `SoCaVisualizationModel`::`Constraint VisualizationRules` – Diagramm 4

Es ist möglich, dass eine Menge von Muss-Gestaltungsregeln eine nicht widerspruchsfreie Softwarekarte beschreiben. Beispielsweise kann von zwei Gestaltungsmittelinstanzen verlangt werden, dass sie sich gegenseitig verschachteln (`Nesting`), was nicht angezeigt werden kann. Im Visualisierungsmodell werden in dieser Version keine Beschränkungen eingeführt, um diesen Missstand zu beheben. Eine Erweiterung des Visualisierungsmodells kann diese Beschränkungen einführen, die beispielsweise in OCL [OM06e] formuliert werden. Die Anwendung des Modells ohne Beschränkungen wird in Abschnitt 5.3 gezeigt, wobei das Werkzeug Konflikte erkennt und diese gegebenenfalls einem Benutzer meldet.

4.3.2.5. Visualisierungsmodell: Paket TargetVisualizationRules

Nach der Einführung der Muss-Gestaltungsregeln im Abschnitt 4.3.2.4, werden im Weiteren die Ziel-Gestaltungsregeln im Visualisierungsmodell beschrieben. Die Unterscheidung dieser Arten von Regeln begründet sich in der Tatsache, dass eine Muss-Gestaltungsregel erfüllt sein muss, um die Semantik einer Visualisierung einzuhalten, eine Ziel-Gestaltungsregel hingegen die *ästhetischen* Aspekte einer Visualisierung adressiert. Abbildung 4.33 zeigt hierzu zwei Ausschnitte von Softwarekarten, wobei beide Ausschnitte alle Instanzen von Muss-Gestaltungsregeln hinsichtlich der Verschachtelung erfüllen, jedoch die Ausschnitte unterschiedlich ästhetisch wahrgenommen werden.

Um dem rechten Ausschnitt einer Softwarekarte möglichst nahe zu kommen, werden Ziel-Gestaltungsregeln eingeführt, die für ein Layouting, welches die Positionen von Gestaltungsmitteln errechnet, zusätzliche Ziele angeben. Die vom Visualisierungsmodell hierzu eingeführten Ziel-Gestaltungsregeln sind wie folgt beschrieben (siehe Abbildung 4.34):

`AreaMinimization` verfolgt das Ziel, die Fläche eines flächenartigen Gestaltungsmittels (`PlanarMapSymbol`) zu minimieren. Diese Gestaltungsregel wird beispielsweise angewendet, um die Fläche eines Rechtecks für einen Cluster möglichst klein zu halten. Wird diese Regel auf die Cluster `Org. Unit 1` und `Org. Unit 1.1` des linken

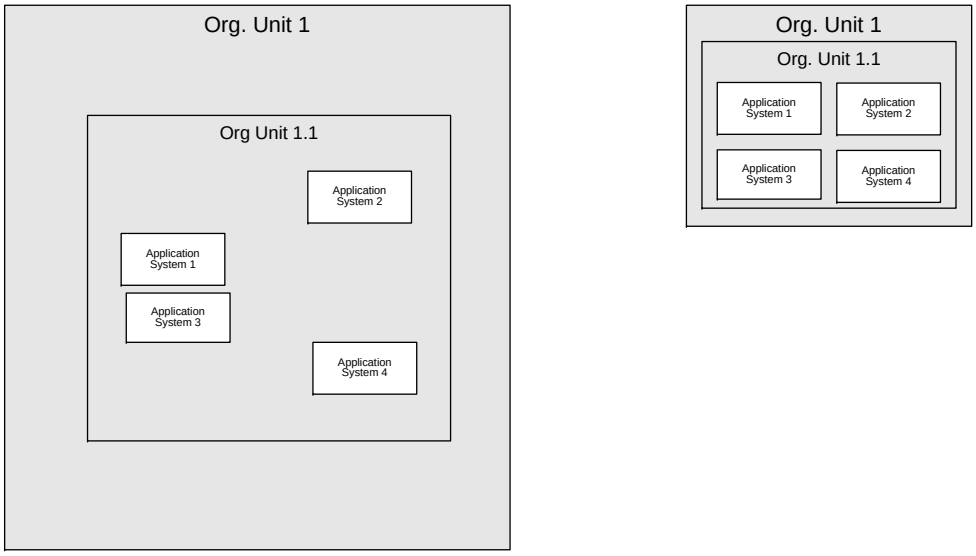


Abbildung 4.33.: Zwei semantisch äquivalente Softwarekarten mit unterschiedlicher, ästhetischer Erscheinung

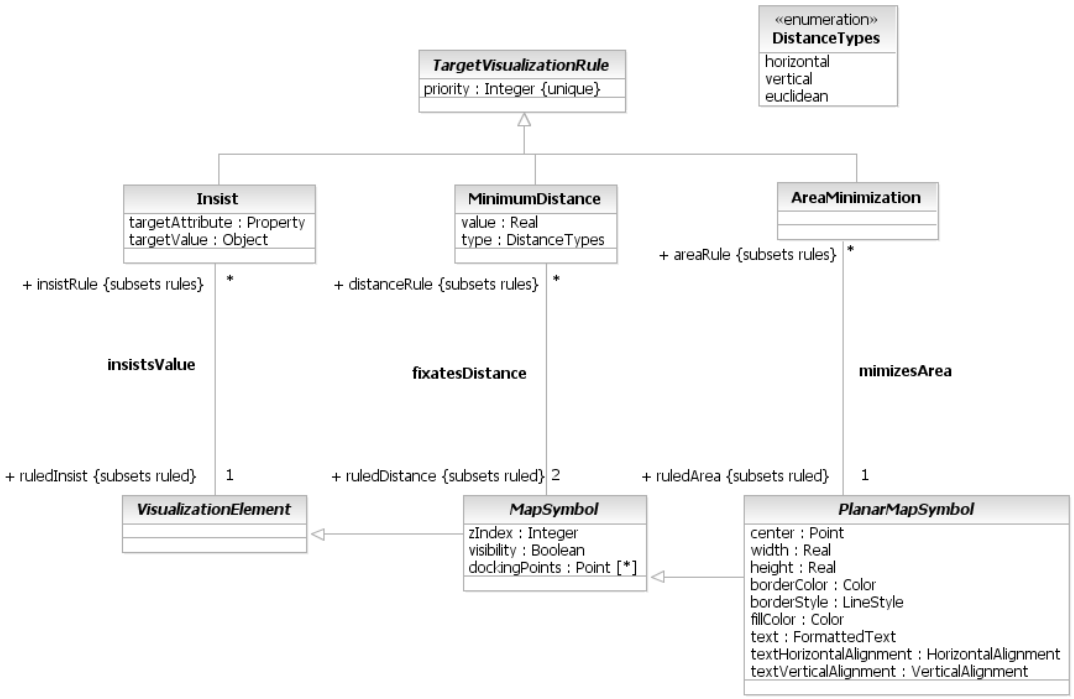


Abbildung 4.34.: Visualisierungsmodell: Paket `SoCaVisualizationModel::TargetVisualizationRule`

Ausschnitts der Softwarekarte in Abbildung 4.33 angewendet, so wäre ein Optimum für die Cluster der rechte Ausschnitt der Abbildung 4.33.

`Insist` verfolgt das Ziel, dass die Instanz eines Visualisierungselements (`Visualization Element`) bezüglich eines festgelegten Attributs (`targetAttribute`) einen bestimmten Wert (`targetValue`) erreicht, der in der Gestaltungsregel angegeben wird. Diese Gestaltungsregel wird beispielsweise bei einem erneuten Layouting nach einer Änderung genutzt, um die Position bestimmter Elemente zu erhalten. Wurden die Positionen von Elementen in einem Cluster manuell geändert, so sollen diese bei einer erneuten Generierung erhalten bleiben. Dies kann jedoch keine Muss-Gestaltungsregel verlangen, da sich gegebenenfalls das semantische Modell für die Softwarekarte geändert hat und einige Visualisierungselemente neu positioniert werden müssen.

`MinimumDistance` verfolgt das Ziel, dass zwei Instanzen von Gestaltungsmitteln (`MapSymbol`) mindestens den minimalen Abstand einhalten, der bei der Anwendung der Gestaltungsregel mittels des Attributs `value` angegeben wird. Die Art des gewünschten Abstands wird durch das Attribut `type` der Klasse `Distance Types` angegeben, die drei Typen von Distanzen unterscheidet: vertikale Distanz (`vertical`), horizontale Distanz (`horizontal`) und euklidische Distanz (`euclidean`). Diese Gestaltungsregel wird beispielsweise angewendet, um von zwei Rechtecken für Anwendungssysteme einen bestimmten minimalen Abstand zu erzielen.

Das Attribut `priority` in der Klasse `TargetVisualizationRule` ermöglicht es, eine Reihenfolge der Ziel-Gestaltungsregeln hinsichtlich der Priorität festzulegen. Da sich Ziel-Gestaltungsregeln gegenseitig beeinflussen können (beispielsweise eine `Area Minimization` und eine `MinimumDistance`), erlaubt es dieses Attribut, eine Priorisierung festzulegen.

Dieser Abschnitt 4.3.2 hat die abstrakte Syntax von Softwarekarten mit den objektorientierten Visualisierungsmodell dargelegt und exemplarisch die konkrete Syntax gezeigt. Eine mathematische Formalisierung dieses objektorientierten Visualisierungsmodells (in einer vorhergehenden Version) wird von Ernst et al. in [Er06] gezeigt. Ernst et al. nutzen hierzu eine denotationelle Semantik unter Anwendung der Prädikatenlogik. Nach der Einführung des objektorientierten Visualisierungsmodells wird im Folgenden Abschnitt 4.3.3 die Anwendung des Visualisierungsmodells beispielhaft illustriert.

4.3.3. Anwendung des Visualisierungsmodells

Die Anwendung des Visualisierungsmodells wird exemplarisch durch den Ausschnitt einer Softwarekarte in Abbildung 4.35 gezeigt. Diese Clusterkarte besteht aus mehreren

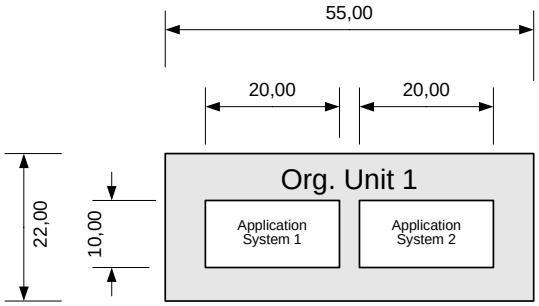


Abbildung 4.35.: Clusterkarte zur Serialisierung in Abbildung 4.36

Gestaltungsmitteln und Gestaltungregeln. Im Folgenden werden diese Visualisierungselemente aufgelistet und zusätzlich die `id`s der Serialisierungssyntax in Abbildung 4.36[26] referenziert:

`Rectangle`: Als Gestaltungsmittel werden ausschließlich Instanzen der Klasse `Rectangle` verwendet. Ein Rechteck (`id=6`) visualisiert eine Organisationseinheit und zwei Rechtecke (`id=13` und `id=20`) visualisieren Anwendungssysteme.

`Separation`: Die Muss-Gestaltungsregel `Separation` (`id=21`) wird angewendet, um die Rechtecke für Anwendungssysteme separiert darzustellen.

`Nesting`: Die Muss-Gestaltungsregel `Nesting` (`id=22` und `id=23`) wird zweimal instantiiert, um die Rechtecke für Anwendungssysteme in dem Rechteck für die Organisationseinheit zu verschachteln.

`IdentityOfProjection`: Die Muss-Gestaltungsregel `IdentityOfProjection` (`id=24`) wird angewendet, um die gleichen Werte für die Gestaltungsvariablen der Rechtecke für Anwendungssysteme hinsichtlich `width`, `height fillColor` und `border Color` zu erreichen.

`VisibilityLayer`: Die Muss-Gestaltungsregel `VisibilityLayer` (`id=25` und `id=26`) wird zweimal angewendet, um die Sichtbarkeit der Schichten für die Organisationseinheit und für die Anwendungssysteme auf sichtbar zu setzen.

`AreaMinimization`: Die Ziel-Gestaltungsregel `AreaMinimization` (`id=27`) wird angewendet, um die Größe des Rechtecks für die Organisationseinheit zu minimieren.

Die Gestaltungsvariablen der Gestaltungsmittel aus der Clusterkarte in Abbildung 4.35 spiegeln in der Serialisierung in Abbildung 4.36 die Eigenschaften der Gestaltungsmittelinstanzen wider. Es wird beispielsweise für die Gestaltungsmittelinstanz der Klasse `Rectangle` mit der `id=16` die Rahmenfarbe auf Schwarz gesetzt und die Füllfarbe auf einen Grauton (`RGB=230/230/230`). Auch die Position dieses Gestaltungsmittels wird über den Mittelpunkt (`center`) beschrieben sowie die Breite (`width`) und Höhe (`height`) dokumentiert.

[26] Die Serialisierung der Softwarekarte in Abbildung 4.35 ist nicht vollständig. Auslassungen werden durch [...] markiert. Die vollständige Serialisierung ist in Anhang C.1 enthalten.

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <SoCaVisualizationModel:SoftwareMap xmi:version="2.0"
3  [...]
4    title="Cluster Map" creationDate="2007-04-24T17:26:37.808+0200" creator="A. Wittenburg">
5  <diagram>
6      <element xsi:type="MapSymbols:Rectangle" id="6" rulingLayer="25" width="55.0" height
           ="22.0" textHorizontalAlignment="center" areaRule="27" nestingRule="22 23" borderStyle
           ="solid">
7        <center id="3" x="22.5" y="11.0"/>
8        <borderColor id="5" red="0" green="0" blue="0"/>
9        <fillColor id="4" red="230" green="230" blue="230"/>
10       <text id="2" value="Org. Unit 1" fontSize="12">
11         <color id="0" red="0" green="0" blue="0"/>
12         <fontType id="1" name="Arial"/>
13       </text>
14     </element>
15     <element xsi:type="MapSymbols:Rectangle" identityRule="24" id="13" rulingLayer="26"
           separationRule="21" width="20.0" height="10.0" textHorizontalAlignment="center"
           textVerticalAlignment="middle" nestedRule="22" borderStyle="solid">
16       <center id="10" x="16.0" y="12.0"/>
17       <borderColor id="12" red="0" green="0" blue="0"/>
18       <fillColor id="11" red="255" green="255" blue="255"/>
19       <text id="9" value="Application System 1" fontSize="6">
20         <color id="7" red="0" green="0" blue="0"/>
21         <fontType id="8" name="Arial"/>
22       </text>
23     </element>
24     <element xsi:type="MapSymbols:Rectangle" identityRule="24" id="20" rulingLayer="26"
           separationRule="21" width="20.0" height="10.0" textHorizontalAlignment="center"
           textVerticalAlignment="middle" nestedRule="23" borderStyle="solid">
25       <center id="17" x="39.0" y="12.0"/>
26       <borderColor id="19" red="0" green="0" blue="0"/>
27       <fillColor id="18" red="255" green="255" blue="255"/>
28       <text id="16" value="Application System 2" fontSize="6">
29         <color id="14" red="0" green="0" blue="0"/>
30         <fontType id="15" name="Arial"/>
31       </text>
32     </element>
33 [...]
34     <element xsi:type="ConstraintVisualizationRules:Separation" id="21" separated="13 20"/>
35     <element xsi:type="ConstraintVisualizationRules:Nesting" id="22" nestedSymbol="13"
           nestingSymbol="6"/>
36 [...]
37     <element xsi:type="TargetVisualizationRules:AreaMinimization" id="27" ruledArea="6"/>
38 [...]
39   </diagram>
40 </SoCaVisualizationModel:SoftwareMap>
```

Abbildung 4.36.: Ausschnitt der Serialisierung der Clusterkarte aus Abbildung 4.35

Das Beispiel zeigt die Anwendung des Visualisierungsmodells für eine Softwarekarte und führt zusätzlich die Serialisierungssyntax für Softwarekarten ein. Weitere Beispiele für Softwarekarten und ihre Serialisierungen, sowie die vollständige Serialisierung der Softwarekarte in Abbildung 4.35 sind im Anhang C enthalten.

4.4. Modelltransformation für Softwarekarten

Nachdem in den Abschnitten 4.1 bis 4.3 die Unterscheidung von semantischen und symbolischen Modellen sowie den korrespondierenden Metamodellen Informationsmodell und Visualisierungsmodell eingeführt wurde, wird im Folgenden dargestellt, wie diese Unterscheidung für Softwarekarten angewendet wird. Ziel dieser Anwendung ist, eine in Software implementierbare Lösung zu erhalten, um Softwarekarten zu generieren und Änderungen in Softwarekarten in ein Repository zurückzuspielen.

Hierzu wird der Ansatz der Modelltransformation vorgestellt, der es ermöglicht, ein oder mehrere Quellmodelle in ein oder mehrere Zielmodelle umzusetzen. Dieser Ansatz wird im Software-Engineering beispielsweise angewendet, um

- E/R-Modelle in SQL-Code umzusetzen (*Modell zu Quelltext Transformation*),
- UML-Modelle in einen Java oder C# Quelltext zu übersetzen (*Modell zu Quelltext Transformation*) oder
- ein UML-Modell ohne Profilerweiterung in ein UML-Modell mit Profilerweiterung (bspw. für Enterprise Java Beans) zu transformieren (*Modell zu Modell Transformation*).

Eine Standardisierungsbemühung im Kontext von Modelltransformationen stellt die *Model Driven Architecture* (MDA) der OMG [OM03] dar. Bei der MDA wird ein *Platform Independent Model* (PIM) als Quellmodell in ein *Platform Specific Model* (PSM) als Zielmodell transformiert. Bei einem PIM handelt es sich um ein Modell, welches keine Annahmen über die – zur späteren Implementierung verwendete – Plattform macht. Das PIM kann beispielsweise aus einem Modell bestehen, welches sich aus einer Menge von UML-Klassendiagrammen zusammensetzt.

4.4.1. Modelltransformation zur Kopplung von semantischen und symbolischen Modellen

Die Technik der Modelltransformation wird in dem hier vorgestellten Ansatz verwendet, um eine Beziehung zwischen dem semantischen und dem symbolischen Modell zu etablieren. Im Gegensatz zur MDA existiert kein PIM oder PSM, sondern lediglich Quell- und Zielmodelle. Die Transformation eines Elements des semantischen Modells (bspw. `AccountingSystem:ApplicationSystem`) zu einem Element des symbolischen Modells (bspw. `AccountingSystem:Rectangle`) wird durch eine Regel eines Transformationsmodells ermöglicht, die alle Anwendungssysteme in Rechtecke transformiert.

Die Anwendung der Modelltransformation für Softwarekarten ist in Abbildung 4.37 illustriert. Neben den Modellen *semantisches Modell*, *Informationsmodell*, *symbolisches Modell* und *Visualisierungsmodell* werden ein Metamodell für das Informationsmodell und Visualisierungsmodell sowie ein Transformationsmodell eingeführt. Die in Abbildung 4.37 gezeigte Transformation ist als bidirektionale Transformation vorgesehen, so dass sowohl semantische Modelle in symbolische Modelle transformiert werden können als auch umgekehrt.

Das Metamodell in Abbildung 4.37 ist die gemeinsame Sprache für das Informationsmodell und das Visualisierungsmodell und führt zu einer Vereinfachung der Transformation. Basieren die Modelle auf einem gemeinsamen Metamodell, ist bei der Transformation zu vernachlässigen, wie die in den Modellen verwendeten Konzepte (bspw. Klasse) ineinander transformiert werden.

Eine wichtige Voraussetzung eines gemeinsamen Metamodells für das Informationsmodell und das Visualisierungsmodell ist, dass die Mächtigkeit im Sinne der vorhandenen

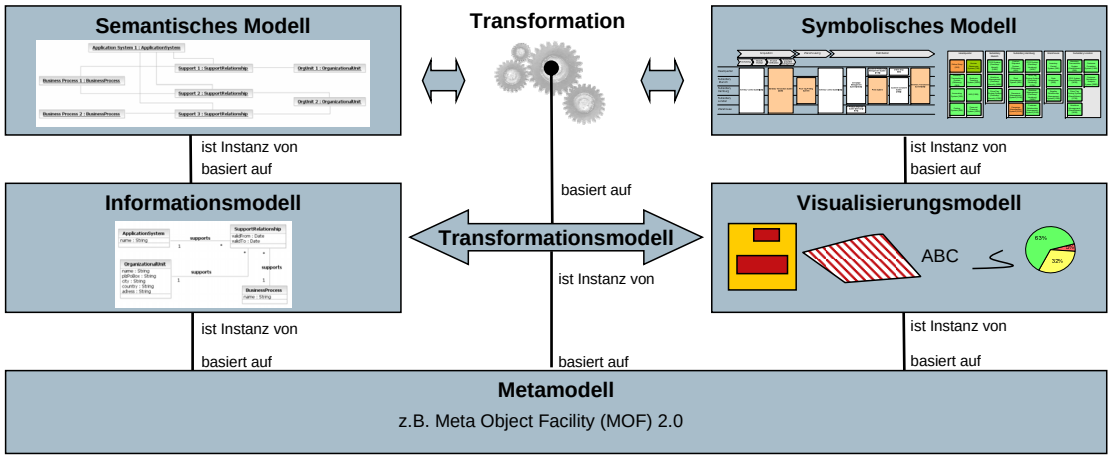


Abbildung 4.37.: Modelltransformation zur Kopplung von semantischen und symbolischen Modellen

Metamodellelemente für das Quell- *und* Zielmodell ausreichend ist. Für ein Informationsmodell hat Buckl [Bu05] gezeigt, dass diese Annahme für ein objekt-orientiertes Metamodell wie das der Meta Object Facility (MOF) [OM06d] gegeben ist, wenn Informationsmodelle wie bei Brendebach [Br05a], Halbhuber [Ha04] oder Lauschke [La05b] verwendet werden.[27] Dass diese Annahme ebenso für ein Visualisierungsmodell gegeben ist, ergibt sich aus dem in Abschnitt 4.3 entwickelten Visualisierungsmodell, welches Konzepte von MOF verwendet. Für ein Transformationsmodell, welches gleichfalls auf MOF basiert, wird derzeit der Standard *Query/View/Transformation* [OM05a] innerhalb der OMG entwickelt.

Um die Modelltransformation durchzuführen, werden Transformationsregeln benötigt, die definieren, wie die Elemente eines oder mehrerer Quellmodelle in Elemente eines oder mehrerer Zielmodelle überführt werden. Das Transformationsmodell mit diesen Regeln wird hierzu auf der Ebene des Informationsmodells und Visualisierungsmodells definiert und auf Instanzen dieser Modelle angewendet. Im Sinne des IEEE 1471 (siehe Abschnitt 3.1.1) definieren Transformationsregeln einen *Viewpoint*, dessen Anwendung zu einem *View*, bestehend aus *Models*, führt.

Das Konzept *Model* des IEEE 1471 korrespondiert mit dem eines symbolischen Modells der Softwarekartographie, eine Unterscheidung zwischen symbolischen und semantischen Modellen findet im IEEE 1471 nicht statt. Dies begründet sich in der Zielsetzung des Standards, der die Dokumentation von Architekturen, bestehend aus einer Menge von *erfahrbaren* und *kommunizierbaren* Modellen, in den Vordergrund stellt. Eine Teilmenge der Dokumentation von Anwendungslandschaften sind die symbolischen Modelle der Softwarekartographie.

Das Transformationsmodell definiert jedoch nicht nur, wie Elemente transformiert werden, sondern auch die Filter, welche angewendet werden, um Elemente auszuschließen,

[27] Siehe auch Abschnitt 4.2 für Modellelemente von Informationsmodellen.

```
1  rule Location2Rectangle {
2    from
3      infoObject : Semantic.Location
4    to
5      symbol : Symbolic.Rectangle (
6        text = infoObject.name,
7        backgroundColor = #CCCCCC
8      )
9  )
10
11 rule BusinessAppSys2Rectangle {
12   from
13     infoObject : Semantic.ApplicationSystem
14   to
15     symbol : Symbolic.Rectangle (
16       text = infoObject.name + '(' + infoObject.id + ')'
17       backgroundColor = #FFFFFF
18     ),
19     rule : Symbolic.Nesting (
20       inner = symbol,
21       outer = thisModule.resolveTemp(infoObject.hostedAt,
             'Location2Rectangle')
22     )
23 )
```

Abbildung 4.38.: Beispielhafte Regeln für eine Transformation zwischen Informationsmodell und Visualisierungsmodell

die für einen *Viewpoint* nicht relevant sind. Ein einfaches Beispiel mit zwei Transformationsregeln ist in Abbildung 4.38 dargestellt. Diese Transformation erzeugt aus dem semantischen Modell in Abbildung 4.2 das symbolische Modell in Abbildung 4.4.

Die erste Regel `Location2Rectangle` transformiert ein Informationsobjekt, welches eine Instanz der Klasse `Location` ist, in eine Instanz der Klasse `Rectangle`. Zusätzlich wird der Wert des Textfeldes `text` mit dem Wert des Attributs `name` des Standorts belegt und die Hintergrundfarbe auf `#CCCCCC` gesetzt. Die zweite Regel `BusinessAppSys2Rectangle` transformiert analog jedes Anwendungssystem in ein Rechteck mit der Hintergrundfarbe `#FFFFFF`. Darüber hinaus bildet sie die Beziehung `hostedAt` zwischen Standorten und Anwendungssystemen auf eine Instanz der Muss-Gestaltungsregel *Nesting* ab und assoziiert die entsprechenden Objekte im Zielmodell.

Im Bereich Software-Engineering existieren neben QVT [OM05a] verschiedene weitere Modelltransformationssprachen, wie beispielsweise ATL [AT06], die in Abbildung 4.38 verwendet wurde, oder BOTL [BM03]. Ebenso denkbar zur Definition der Modelltransformation ist der Einsatz von Programmiersprachen (bspw. Java). Für eine Diskussion der Implementierung der Modelltransformation zwischen semantischen und symbolischen Modellen wird auf Abschnitt 5.3 verwiesen.

4.4.2. Layouting des symbolischen Modells zur Erzeugung von Softwarekarten

Der Vorteil bei der Anwendung von Modelltransformationen zur Kopplung des semantischen Modells an das symbolische Modell liegt in der Flexibilität hinsichtlich des Informationsmodells und Visualisierungsmodells. Abschnitt 4.2 hat gezeigt, dass es *das eine* Informationsmodell nicht gibt und somit eine Flexibilität hinsichtlich des Informationsmodells benötigt wird, um adäquate Darstellungen (symbolische Modelle) zu erhalten. Ist es das Ziel, Softwarekarten für die Informationen des semantischen Modells zu erstellen, so ist keine Anpassung des Visualisierungsmodells aus Abschnitt 4.3 notwendig. Das Visualisierungsmodell stellt die notwendigen Konzepte zur Darstellung von Softwarekarten bereit, so dass die Definition einer Transformationsregel, welche die Informationsmodell-spezifischen Elemente verwendet, ausreichend ist, um Softwarekarten zu erzeugen.

Unter der Annahme, dass das Visualisierungsmodell wie in Abschnitt 4.3 definiert ist und dass effiziente und berechenbare Algorithmen existieren, um die Gestaltungsvariablen der Gestaltungsmittel zu kalkulieren (Position, Größe etc.), kann das symbolische Modell genutzt werden, um automatisch eine Visualisierung zu generieren.

Ansätze zur Kalkulation der Gestaltungsvariablen sind der Einsatz von Heuristiken, die beispielsweise Packing-Algorithmen [La07] verwenden, oder die Repräsentation als Optimierungsproblem. Dieser Ansatz basiert auf Ernst et al. [Er06] und wird im Folgenden vorgestellt.

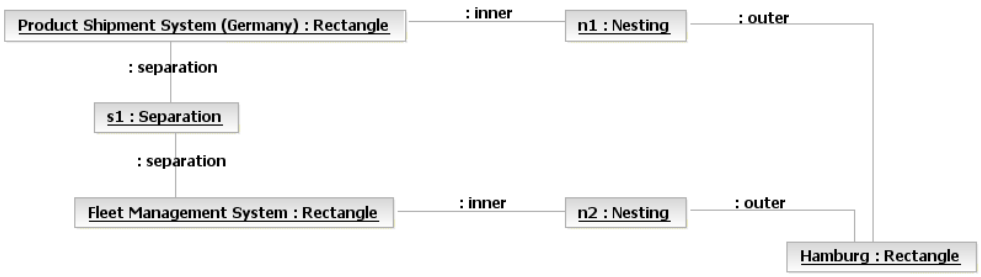


Abbildung 4.39.: Ausschnitt eines symbolischen Modells für Abbildung 4.1

Abbildung 4.39 zeigt hierzu ein symbolisches Modell, welches drei Gestaltungsmittelinstanzen über drei Instanzen von Gestaltunsregeln derart in Beziehung setzt, dass ein als Rechteck dargestellter Cluster für eine Organisationseinheit entsteht, der zwei weitere Rechtecke für Anwendungssysteme enthält, die separiert dargestellt werden müssen.

Ein aus dem symbolischen Modell abgeleitetes Ungleichungssystem ist in Abbildung 4.40 enthalten. Die Ungleichungen werden auf Basis des in Abbildung 4.41 dargestellten Aufbaus erstellt, um die Variablen der Gestaltungsmittel zu bestimmen. Es entsteht somit durch das Ableiten von Ungleichungen aus den Muss-Gestaltungsregeln der zulässige Bereich eines Optimierungsproblems. Eine Zielfunktion, die nicht in Abbildung 4.40 enthalten ist, minimiert beispielsweise die Größe des Rechtecks für den Cluster und wird aus Ziel-Gestaltungsregeln abgeleitet.

Ungleichungen für Muss-Gestaltungsregel `n1:Nesting`:

$$\left(x_{r2} - \frac{w_{r2}}{2}\right) - \left(x_{r1} - \frac{w_{r1}}{2}\right) \geq 0$$
$$\left(y_{r2} - \frac{h_{r2}}{2}\right) - \left(y_{r1} - \frac{h_{r1}}{2}\right) \geq 0$$
$$\left(x_{r1} + \frac{w_{r1}}{2}\right) - \left(x_{r2} + \frac{w_{r2}}{2}\right) \geq 0$$
$$\left(y_{r1} + \frac{h_{r1}}{2}\right) - \left(y_{r2} + \frac{h_{r2}}{2}\right) \geq 0$$

Ungleichungen für Muss-Gestaltungsregel `n2:Nesting`:

$$\left(x_{r3} - \frac{w_{r3}}{2}\right) - \left(x_{r1} - \frac{w_{r1}}{2}\right) \geq 0$$
$$\left(y_{r3} - \frac{h_{r3}}{2}\right) - \left(y_{r1} - \frac{h_{r1}}{2}\right) \geq 0$$
$$\left(x_{r1} + \frac{w_{r1}}{2}\right) - \left(x_{r3} + \frac{w_{r3}}{2}\right) \geq 0$$
$$\left(y_{r1} + \frac{h_{r1}}{2}\right) - \left(y_{r3} + \frac{h_{r3}}{2}\right) \geq 0$$

Ungleichungen für Muss-Gestaltungsregel `s1:Separation`:

$$\left(|x_{r2} - x_{r3}| \geq \frac{w_{r2} + w_{r3}}{2}\right) \vee \left(|y_{r2} - y_{r3}| \geq \frac{h_{r2} + h_{r3}}{2}\right)$$

Abbildung 4.40.: Abgeleitete Ungleichungen des Modells aus Abbildung 4.39

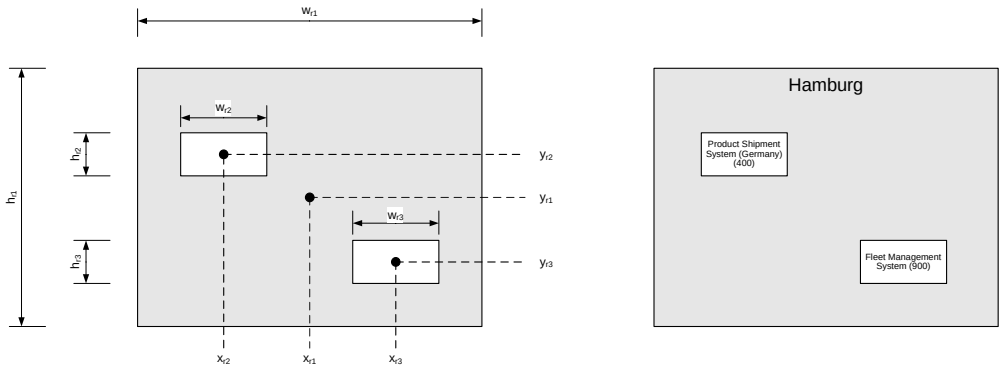


Abbildung 4.41.: Ableitung der Variablen für das Ungleichungssystem des Modells aus Abbildung 4.39

Eine im Sinne des symbolischen Modells aus Abbildung 4.39 korrekte Softwarekarte definiert nicht, wo die Rechtecke für Anwendungssysteme innerhalb des Clusters positioniert werden. Die Transformation, die aus dem semantischen das symbolische Modell erzeugt, enthält in diesem Fall keine Regeln, die den Transport einer Semantik durch die Position im Cluster definiert. Des Weiteren werden in der Transformation keine Vorgaben für die Größe der einzelnen Rechtecke gemacht. Soll die Größe eines Rechtecks für ein Anwendungssystem keine Semantik transportieren, so kann die Transformation weitere Muss-Gestaltungsregeln erzeugen, die verlangen, dass die Rechtecke für Anwendungssysteme hinsichtlich Breite und Höhe gleich sind (Muss-Gestaltungsregel `IdentityOf Projection`).

Die Komplexität des Optimierungsproblem hängt von den im symbolischen Modell enthaltenen Gestaltungsregeln ab, so dass unterschiedliche Klassen von Optimierungsproblemen entstehen können. Die Wahl der Algorithmen, um eine Lösung für das Optimierungsproblem zu erhalten, welches die Positionen und Größen der Gestaltungsmittel enthält, kann somit variieren.

Analog zu obigem Beispiel für eine Clusterkarte können für kartesische Karten Optimierungsprobleme gebildet werden. Problematisch ist, dass es sich bei den erzeugten Optimierungsproblemen zumeist um Probleme der nichtlinearen Optimierung handelt und somit der Einsatz von effizienten Algorithmen nur teilweise gegeben ist. Ein Versuch, eine (optimale) Lösung für das Optimierungsproblem mittels genetischer Algorithmen zu finden, hat sich als problematisch erwiesen, da die Dimension und Problemklasse zu ineffizientem und ineffektivem Verhalten geführt hat.

Eine Möglichkeit dieses Problem zu lösen ist die Problemklasse des mathematischen Modells zu verändern und beispielsweise ein Problem der linearen Programmierung zu erzeugen. Dies wäre möglich, wenn bestimmte Vereinfachungen angenommen werden. Ein mögliche Vereinfachung ist, dass eine Eigenschaft aus dem semantischen Modell (beispielsweise der Name eines Anwendungssystems) für eine Sortierung verwendet wird. Es werden somit die boolschen ODER-Operatoren aus Abbildung 4.40 eliminiert, die aus der Anwendung der Regel `Separation` für die Rechtecke der Anwendungssysteme entstanden sind.

Trotz der Schwierigkeit, dass für das Optimierungsproblem bisher keine effizienten Algorithmen identifiziert wurden, besitzt der Ansatz, aus Gestaltungsregeln Ungleichungen abzuleiten, eine Bedeutung für das Editieren von Softwarekarten. Wird beispielsweise ein Rechteck für ein Anwendungssystem in einer Clusterkarte aus einem Cluster entfernt, so können die Instanzen der Gestaltungsregeln für die Gestaltungsmittelinstanz identifiziert werden und die abgeleiteten Ungleichungen effektiv und effizient überprüft werden. Es ist somit möglich festzustellen, dass ein Objekt aus einem Cluster entfernt oder herausgeschoben wurde, woraus abgeleitet werden kann, dass diese Information in Veränderungen im semantischen Modell übersetzt werden muss, um die Konsistenz zwischen Daten und Visualisierung zu gewährleisten.

Da es jedoch auch Transformationsregeln gibt, die nicht bijektiv sind[28], ist es nicht immer möglich, Rückschlüsse auf die korrekte Änderung des semantischen Modells zu ziehen. In diesem Fall müsste die Änderung in der Softwarekarte entweder rückgängig gemacht oder das symbolische Modell von dem semantischen Modell entkoppelt werden.

4.5. Diskussion existierender Modellierungssprachen

Bevor in Kapitel 5 die Anwendung von Softwarekarten detailliert wird, werden im Folgenden existierende Modellierungssprachen aus dem Bereich der Informatik und Wirtschaftsinformatik diskutiert. Im Kontext des Managements von Anwendungslandschaften und Unternehmensarchitekturen sind insbesondere die Modellierungssprachen UML und ArchiMate hervorzuheben, die in den folgenden Abschnitten der Softwarekartographie gegenübergestellt werden.

Neben diesen Modellierungssprachen wurden in Abschnitt 2.3 bereits Modelle und Frameworks diskutiert, die jedoch keine ausgeprägte Modellierungssprache für Anwendungslandschaften inhärent beschreiben.

Die Werkzeugunterstützung zum Management von Anwendungslandschaften und Unternehmensarchitekturen wird eigenständig in Abschnitt 5.2 behandelt. Einige Werkzeuge nutzen die Modelle und Methoden der Softwarekartographie und werden im Kontext einer Werkzeugstudie in Abschnitt 5.2 betrachtet und in diesem Abschnitt nicht angeführt.

4.5.1. UML - Unified Modeling Language

Die *Unified Modeling Language*, die aus unterschiedlichen objektorientierten Modellierungssprachen (u. a. OMT und OOSE) für Analyse, Design und Entwurf von Softwaresystemen entstanden ist, versteht sich in Version 2.0 als eine vielfältig anwendbare, graphische Modellierungssprache:

> The Unified Modeling Language is a visual language for specifying, constructing, and documenting the artifacts of systems. It is a general-purpose modeling language that can be used with all major object and component methods, and that can be applied to all application domains (e.g., health, finance, telecom, aerospace) and implementation platforms (e.g., J2EE, .NET). [OM05c]

Der zweite Satz des obigen Zitats zeigt, auf welche Methoden UML den Schwerpunkt legt: objekt- und komponentenorientierte Methoden. Um die Anwendbarkeit von UML als Modellierungssprache für Anwendungslandschaften zu analysieren, ist der Aufbau von UML genauer zu betrachten.

UML 2.0 besteht aus der *UML Superstructure* [OM05d] und der *UML Infrastructure* [OM05c] sowie dem Metamodell *Meta Object Facility* [OM06d]. So besitzt UML 2.0

[28] Dies kann beispielsweise durch das Aggregieren und Filtern von Informationen auftreten, siehe auch Braun und Marschall [BM03].

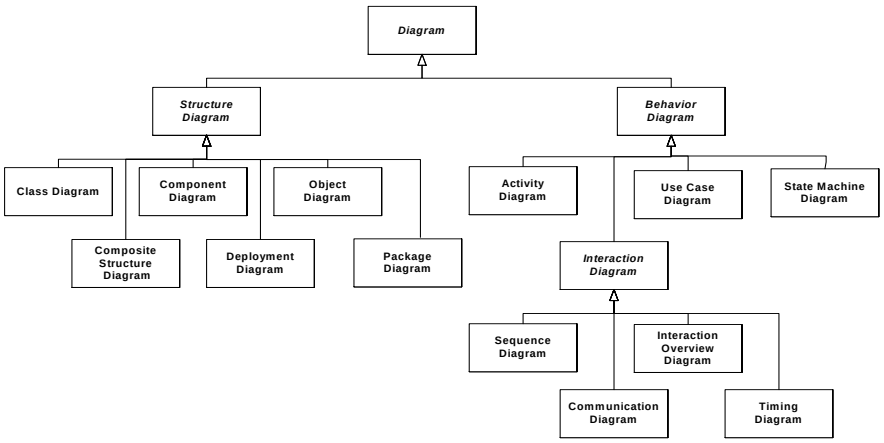


Abbildung 4.42.: Diagrammtypen UML 2.0 [OM05d]

ca. 260 Klassen und kennt 13 Diagrammtypen (siehe Abbildung 4.42). Für den Modellierer von besonderem Interesse ist die UML Superstructure, die u. a. den Aufbau der Diagramme definiert.

Die abstrakte Syntax wird in UML vorwiegend natürlichsprachlich beschrieben und vereinzelt durch die Verwendung der Backus-Naur-Form (BNF) und der Object Constraint Language [OM06e] ergänzt. Die konkrete Syntax (der Diagramme) wird mittels Notationstabellen definiert (siehe Abbildung 4.43), wobei u. a. die folgenden Probleme auftreten:

- Die Gestaltungsregeln werden nicht eingeführt. In Abbildung 4.43 wird beispielsweise das Verteilen (engl. *Deployment*) eines Artefakts auf einen Knoten durch

Node Type	**Notation**	**Reference**
Artifact	«artifact» ArtifactName	See “Artifact.”
Node	NodeName	See “Node.” Has keyword options «device» and «execution environment».
Artifact deployed on Node	«artifact» ArtifactName	See “Deployment.”
Node with deployed Artifacts	Node: «artifact» ArtifactName	See “Deployment.”

Abbildung 4.43.: Notationstabelle aus UML 2.0 [OM05d]

das Verschachteln der Elemente dargestellt. Es wird jedoch nicht definiert, ob das Artefakt vollständig innerhalb des Knotens liegen muss oder auch eine partielle Überschneidung ausreichend ist. Der Terminus Gestaltungsregel ist in UML nicht bekannt. Wie die Beispiele für Softwarekarten aus Abschnitt 3.3.1 gezeigt haben, wird häufig mit einer Positionierung der Elemente gearbeitet, um eine bestimmte Semantik zu transportieren. Dies spiegelt sich ebenso in den Softwarekartentypen in Abschnitt 3.4.1 wider.

- Die Gestaltungsmittel von UML werden ohne die möglichen Gestaltungsvariablen definiert. Variationen in der Größe der Elemente oder ihrer Farbgebung werden nicht berücksichtigt. Da in der Softwarekartographie Farben und Größen zum Transport von Semantik genutzt werden, stellt dies einen Missstand dar.

Die Semantik von UML wird vorwiegend natürlichsprachlich definiert. Da dies bei der Modellierung von Softwaresystemen oftmals nicht ausreichend ist, werden Teile der Semantik durch formale Ausdrücke mittels der *Object Constraint Language* (OCL) [OM06e] ergänzt. Dabei konzentriert sich die Semantik auf objekt- und komponentenorierentierte Systeme und besitzt zahlreiche Elemente zur Beschreibung von Struktur und Verhalten. Insgesamt genügt die Semantik von UML jedoch nicht der einer formalen Modellierungssprache, wie beispielsweise von Petrinetzen. Ansätze zur Formalisierung eines Teils der in UML verwendeten Sprachkonstrukte finden sich u. a. bei Broy et al. [BCR06].

Wird UML zur Modellierung von Anwendungslandschaften genutzt, so können hierzu die Erweiterungsmöglichkeiten von UML wie Stereotypen und Profile ihre Anwendung finden und beispielsweise folgende Diagrammtypen zum Einsatz kommen:

Anwendungsfalldiagramm: Das Anwendungsfalldiagramm stellt Beziehungen zwischen Akteuren und Anwendungsfällen dar. Anwendungssysteme werden durch die Systemgrenze beschrieben und die Anwendungsfälle den einzelnen Systemen zugeordnet. Akteure sind Nutzer (z. B. Personen oder Personengruppen) von Anwendungssystemen oder Organisationseinheiten, denen Anwendungssysteme zugeordnet werden.

Technische, wirtschaftliche oder planerische Aspekte, wie beispielsweise Programmiersprachen oder Betriebskosten, sind durch Anwendungsfälle nicht abbildbar. Anwendungslandschaften mit hundert oder mehr Anwendungssystemen mit Anwendungsfalldiagrammen zu modellieren, ist nicht zielführend, da die in Abschnitt 3.3.2 beschriebenen Merkmale nicht intuitiv abgebildet werden können.

Klassen-/Objektdiagramm: Die grundlegenden Elemente des Klassendiagramms wie Klasse, Assoziation, Vererbung und Rolle werden zur Modellierung von Anwendungslandschaften genutzt, wobei Stereotypen für Klassen und Assoziationen die Definition bestimmter Variationen ermöglichen. Der Stereotyp *ApplicationSystem* bei einer Klasse bedeutet, dass diese Klasse ein Anwendungssystem repräsentiert; *OrganizationalUnit* gibt einer Klasse die Bedeutung einer Organisationseinheit.

Kommunikationsbeziehungen zwischen Anwendungssystemen werden durch Assoziationen in Verbindung mit dem Lollipop-Symbol dargestellt. Ein zusätzlicher Ste-

reotyp an der Assoziation kennzeichnet, ob es sich beispielsweise um einen *Online-* oder *Batch-Konnektor* handelt.

Auf diesem Wege werden die relevanten Merkmale Anwendungssystem, Kommunikationsbeziehung und Organisationseinheit in einem Objektdiagramm visualisiert. Die begrenzte Anzahl graphischer Elemente des Klassen-/Objektdiagramms erlaubt es jedoch nicht, verschiedene Symbole für Geschäftsprozesse, Anwendungssysteme oder Organisationseinheiten zu verwenden, hierzu müsste ein entsprechendes Profil entwickelt werden.

Komponentendiagramm: Das Komponentendiagramm ist, wie das Klassendiagramm, eines der Strukturdiagramme von UML und visualisiert die Verteilung von Artefakten, die Softwarekomponenten repräsentieren, auf verschiedene Laufzeitumgebungen (z. B. Applikationsserver oder Datenbankserver). Da physikalische Geräte und Middleware-Komponenten relevante Merkmale für Softwarekarten sind, werden Komponentendiagramme genutzt, um auf einer detaillierten Ebene für einzelne Softwaresysteme Informationen über ihre physikalische Verteilung zu erhalten. Dabei ist es möglich, ein einzelnes Anwendungssystem zu visualisieren, jedoch nicht eine Menge von Anwendungssystemen.

In der Praxis sind Ansätze zur Modellierung von Anwendungslandschaften mittels Komponenten- oder Klassendiagrammen zu finden, jedoch wird zumeist auf die Verwendung eines eigenen Profils verzichtet und lediglich existierende Elemente umdefiniert. Eine Klasse stellt beispielsweise ein Anwendungssystem dar und ein Paket repräsentiert einen Funktionsbereich. Es wird somit eine eigene Semantik definiert, die nur Teile der UML-Notation verwendet. Dies ist jedoch als problematisch anzusehen, da der Umfang der Konzepte von UML erhalten bleibt und beispielsweise eine Assoziation zwischen zwei Anwendungssystemen mit einer Assoziationsklasse UML-konform zu modellieren ist, eine Semantik jedoch nicht definiert wäre.

Ein Ansatz zur Nutzung von UML zur Modellierung von Anwendungslandschaften, bei dem das UML Meta-Modell erweitert wird, ist ARCUS [HMT02, Te05b]. UML wird hierbei um spezifische Stereotypen, Einschränkungen und Notationen erweitert, um verschiedene Merkmale darzustellen. ARCUS definiert die vier Ebenen *Prozess-*, *Fachbegriffs-*, *Anwendungs-* und *Systemarchitektur*, die miteinander über Assoziationen verbunden sind und verschiedene Ebenen einer Anwendungslandschaft visualisieren. Im Gegensatz zu ARCUS werden in der Softwarekartographie nicht nur verschiedene Ebenen miteinander verbunden, sondern mehrere relevante Merkmale auf *einer* Softwarekarte dargestellt. ARCUS bietet beispielsweise nicht die Möglichkeit, Geschäftsprozesse und Anwendungssysteme auf einer Ebene zu visualisieren. Wirtschaftliche oder planerische/strategische Merkmale finden in ARCUS bei der Darstellung und im Modell keine Berücksichtigung.

4.5.2. ArchiMate

ArchiMate ist eine graphische Modellierungssprache, die am Telematica Instituut in den Niederlanden entwickelt wird. Das Informationsmodell von ArchiMate wurde bereits in Abschnitt 4.2.2 diskutiert. Die graphische Modellierung von ArchiMate ist hinsichtlich

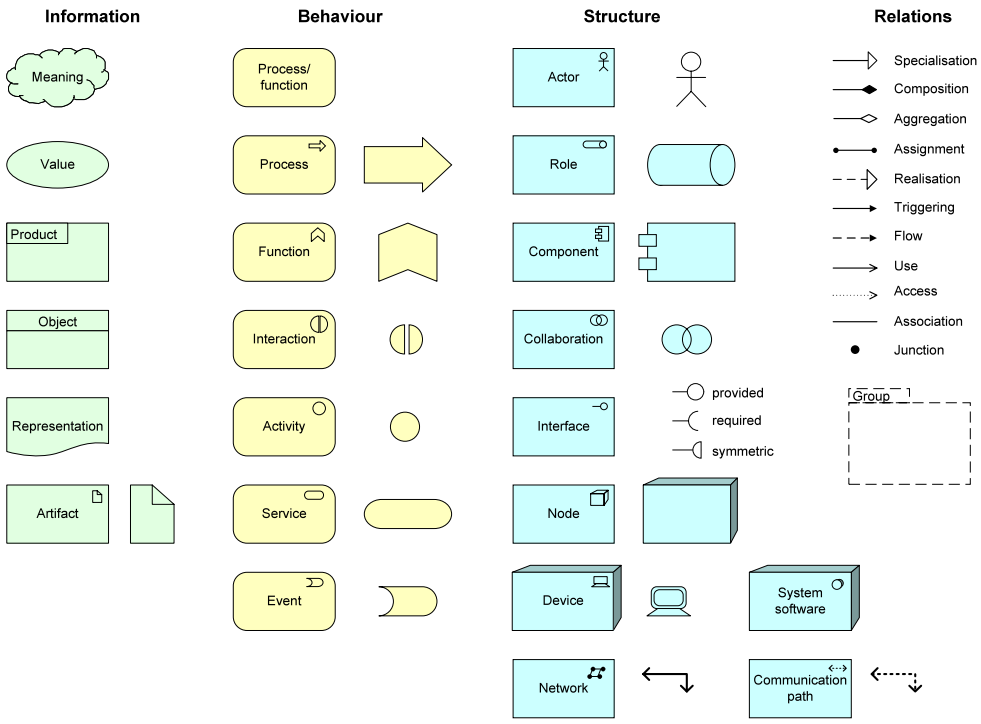


Abbildung 4.44.: Gestaltungsmittel von ArchiMate [Te05a]

des Aufbaus verwandt mit Modellierungssprachen, wie beispielsweise den ereignisgesteuerten Prozessketten, und definiert ein geschlossenes Informationsmodell.

Abbildung 4.44 zeigt die von ArchiMate definierten Gestaltungsmittel und Abbildung B.3 im Anhang B einen Ausschnitt des Informationsmodells. Die einzelnen Gestaltungsmittel, beispielsweise ein Rechteck mit einem Lollipop in der oberen rechten Ecke für eine Schnittstelle (engl. *Interface*), sind im Gegensatz zu den Gestaltungsmitteln der Softwarekartographie mit einem oder mehreren Elementen im Informationsmodell verbunden, beispielsweise mit einer Geschäftsschnittstelle (engl. *Business Interface*) oder mit einer Applikationsschnittstelle (engl. *Application Interface*). Eine Modelltransformation mittels Transformationsregeln, um Elemente des Informationsmodells an Elemente des Visualisierungsmodells zu koppeln, ist in ArchiMate nicht vorgesehen.

Ein Beispiel für die Anwendung der Modellierungssprache ArchiMate zeigt Abbildung 4.45, die Applikationen[29] nach Abteilungen gruppiert. Die Darstellung ist vergleichbar mit einer Clusterkarte, wobei ArchiMate jedoch keine Gestaltungsregeln kennt und die Gruppierung durch das Gestaltungsmittel *Group* (siehe Abbildung 4.44) erfolgt, welches beliebige Elemente gruppieren kann. Es ist somit nicht möglich, Elemente in einer Clusterkarte zu erzeugen, bei der Cluster durch ein wählbares Element des Informationsmodells (bspw. Organisationseinheit) gebildet und in dem Cluster Instanzen eines anderen Elements (bspw. Anwendungssystem) positioniert werden.

[29] Es werden an dieser Stelle die Namen aus dem Informationsmodell von ArchiMate verwendet.

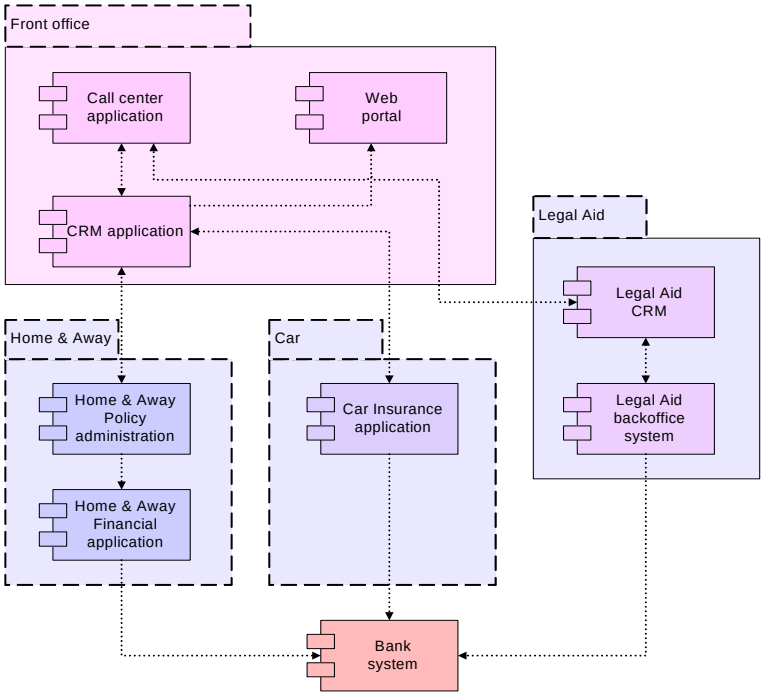


Abbildung 4.45.: Beispiel eines Modells in ArchiMate [La04]

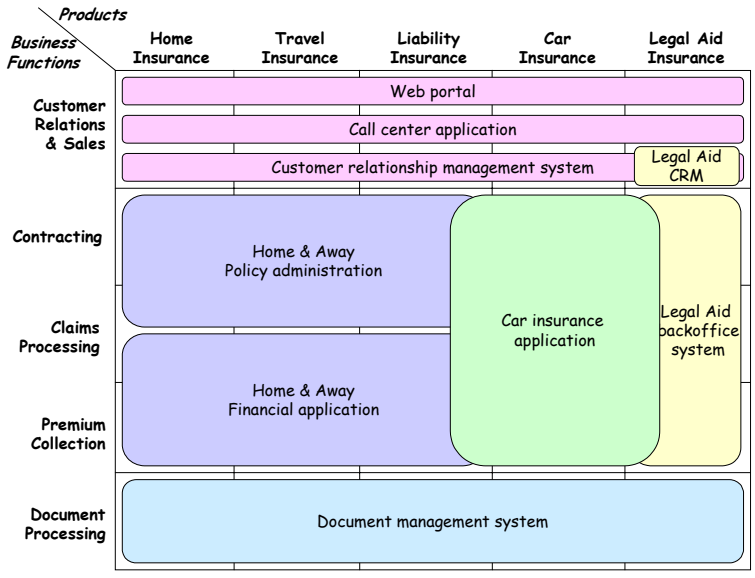


Abbildung 4.46.: *Landscape Map* nach van der Torre et al. [To05]

Eine Erweiterung von ArchiMate stellen die so genannten *Landscape Maps* dar, die von van der Torre et al. [To05] entwickelt wurden. Abbildung 4.46 zeigt eine *Landscape Map*, deren Aufbau verwandt mit den kartesischen Karten der Softwarekartographie ist. Beide Karten definieren einen Aufbau entlang zweier Achsen und sind hinsichtlich der Wahl der Achsen und den innerhalb der Achsen dargestellten Elementen flexibel.

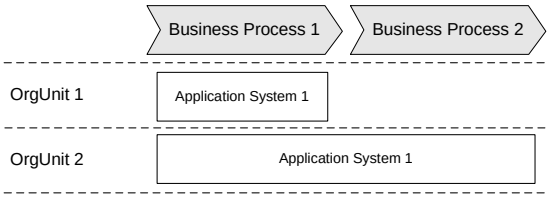


Abbildung 4.47.: Kartesische Softwarekarte zur Verdeutlichung der ternären Beziehung

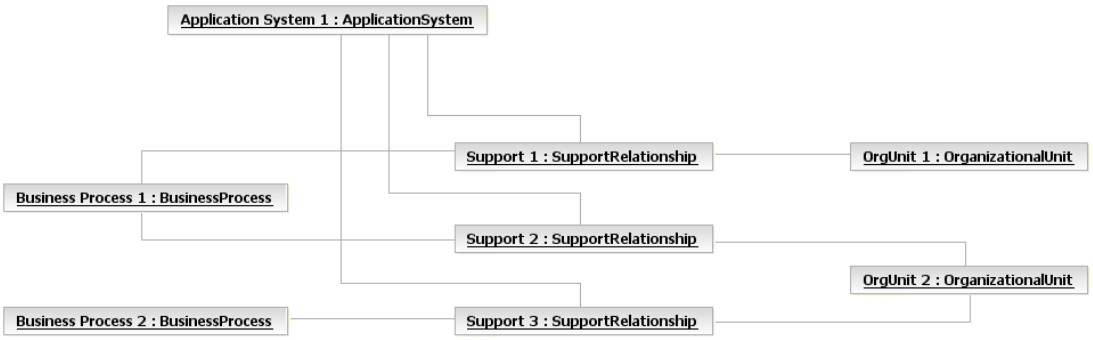


Abbildung 4.48.: Ausschnitt des semantischen Modells der Softwarekarte in Abbildung 4.47

Van der Torre et al. [To05] nutzen eine formale Syntax mittels Signaturen[30], um eine *Landscape Map* zu beschreiben und aufzubauen. In der formalen Semantik einer *Landscape Map* sehen van der Torre et al. eine ternäre Beziehung vor, die mit der ternären Beziehung aus Abschnitt 4.2.1 vergleichbar ist. Jedoch existiert im Informationsmodell von ArchiMate keine derartige ternäre Beziehung, so dass van der Torre et al. zur Visualisierung von Informationsobjekten eines ArchiMate-konformen Informationsmodells zwei binäre Beziehungen miteinander kombinieren [To06, S. 359]. Dies ist jedoch mit einem Informationsverlust verbunden, so dass eine Softwarekarte, wie als Ausschnitt in Abbildung 4.47 dargestellt, mit Informationsobjekten, wie in Abbildung 4.48 gezeigt, nicht möglich ist. Die ternäre Beziehung, welche die Position eines Elements in der Softwarekarte bestimmt, existiert in dem Informationsmodell nicht. Abbildung 4.49 zeigt das semantische Modell für die *Landscape Map* aus Abbildung 4.46 in ArchiMate [Do05], in welchem die ternäre Beziehung nicht vorhanden ist.

Zusammenfassend ist ArchiMate eine Modellierungssprache für Unternehmensarchitekturen, die, wie in Abschnitt 4.2.2 beschrieben, nur die Schichten *Geschäfts-Schicht*,

[30] Dies entspricht einer denotationellen Semantik.

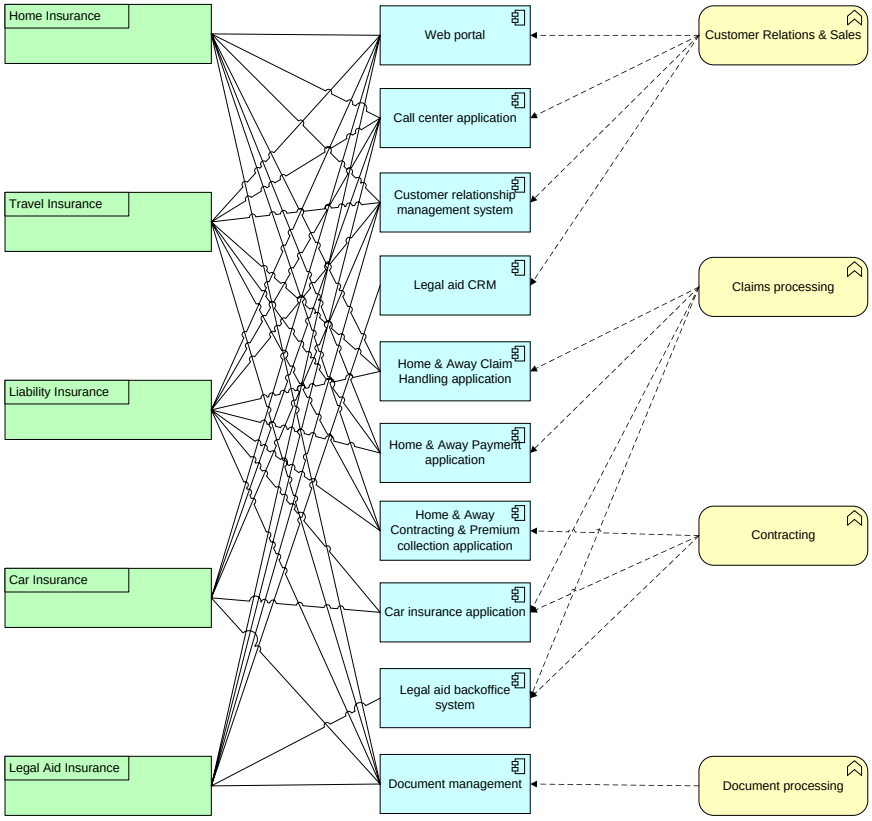


Abbildung 4.49.: Semantisches Modell in ArchiMate der Abbildung 4.46 nach [Do05]

Geschäfts-Service-Schicht, *Anwendungssystem-Schicht*, *Infrastruktur-Service-Schicht* und *Infrastruktur-Schicht* eines Informationsmodells adressiert, ohne die Querschnittsfunktionen (*Anforderung & Projekte* etc.) zu berücksichtigen.

Die *Landscape Maps* von van der Torre et al. sind nicht so flexibel, wie die kartesischen Karten der Softwarekartographie, da sie beispielsweise keine Schichten unterstützen und nicht auf einem Visualisierungsmodell beruhen, welches unter Anwendung einer Transformation unterschiedliche Visualisierungen für dasselbe Modell ermöglicht.

4.6. Bewertung

In diesem Kapitel wurde die Syntax und Semantik von Softwarekarten detailliert beschrieben. Abschnitt 4.1 hat den Zusammenhang von semantischen und symbolischen Modellen dargestellt, um Architekturdokumentation von Anwendungslandschaften und Unternehmensarchitekturen zu erhalten, die den Ansprüchen einer Modellierungssprache genügen.

Die Informationsmodelle, als Metamodelle der semantischen Modelle, waren Gegenstand von Abschnitt 4.2. Es wurde gezeigt, dass es das *eine* Informationsmodell, welches oh-

ne Adaptionen in verschiedenen Unternehmen verwendet werden kann, nicht gibt. Die Interessen der Stakeholder (siehe Abschnitt 3.1) bedingen, dass unterschiedliche Informationsmodelle entstehen, um die Anwendungslandschaft und auch die Unternehmensarchitektur zu dokumentieren. Das in Abschnitt 4.2.1 eingeführte Strukturmodell für Informationsmodelle wurde in den Kontext von existierenden Informationsmodellen gesetzt und es wurde gezeigt, dass Informationsmodelle im Kontext von Anwendungslandschaften und Unternehmensarchitekturen unterschiedliche Interessen bedienen. Der Einsatz von Mustern, wie in Abschnitt 4.2.3 dargestellt, zeigt einen Ansatz auf, um wiederkehrende Strukturen in Informationsmodellen zu identifizieren und die Konstruktion von Informationsmodellen zu vereinfachen.

Das Visualisierungsmodell für die Softwarekartographie, welches in Abschnitt 4.3 vorgestellt wurde, ermöglicht durch ein Modell die Beschreibung der Syntax von Softwarekarten. Die entlang der Praxisbeispiele in Abschnitt 3.3 eingeführten Gestaltungsmittel und Gestaltungsregeln wurden in dem Visualisierungsmodell zusammengefasst, um eine Beschreibung der abstrakten Syntax von Softwarekarten zu bilden. Diese abstrakte Syntax wurde anhand von Beispielen um eine konkrete Syntax ergänzt. Die vorgestellte Serialisierungssyntax für Softwarekarten ermöglicht des Weiteren den Austausch von Softwarekarten zwischen Werkzeugen.

Abschnitt 4.4 hat gezeigt, wie das Visualisierungsmodell für Modelltransformationen angewendet wird, um eine Kopplung des semantischen Modells und des symbolischen Modells zu ermöglichen. Mittels dieser Modelltransformation können aus semantischen Informationen Softwarekarten generiert werden. Ein Werkzeug für die Softwarekartographie, welches diesen Ansatz implementiert, wird in Abschnitt 5.3 vorgestellt.

Die abschließende Diskussion existierender Modellierungssprachen in Abschnitt 4.5 hat gezeigt, dass bisher keine vergleichbare Modellierungssprache für Anwendungslandschaften und Unternehmensarchitekturen bekannt ist. Die Anwendung von UML führt zu mehrdeutigen Modellen, da das vorhandene Metamodell nicht für die Architekturdokumentation von Anwendungslandschaften oder Unternehmensarchitekturen ausgelegt ist. ArchiMate als Modellierungssprache für Unternehmensarchitekturen und insbesondere die im Kontext von ArchiMate entwickelten *Landscape Maps* besitzen zwar Überschneidungen mit der Softwarekartographie, jedoch fehlt es ArchiMate an Visualisierungskonzepten, die in der Softwarekartographie obligatorisch sind. Darüber hinaus sind die *Landscape Maps* hinsichtlich ihrer Anwendung beschränkt.

Im folgenden Kapitel 5 wird die Anwendung von Softwarekarten zum Management von Anwendungslandschaften und Unternehmensarchitekturen aufgezeigt. Hierbei werden Gestaltungsprinzipien für Anwendungslandschaften diskutiert, die mittels Softwarekarten nicht nur visualisiert, sondern auch entwickelt werden. Werkzeuge, die das Management von Unternehmensarchitekturen unterstützen, werden in Zusammenhang mit der Vorstellung der „Enterprise Architecture Management Tool Survey 2005“ [se05a] beschrieben, um abschließend das Design eines Werkzeugs für die Softwarekartographie zu zeigen.

KAPITEL 5

Anwendung von Softwarekarten

Inhaltsverzeichnis

Die Anwendung der in den Abschnitten 3 und 4 aufgebauten Modellierungssprache mit dem Konzept der Softwarekarte wurde bereits in Kapitel 2 initial eingeführt und wird in diesem Abschnitt vertieft. Hierzu werden Techniken zur Gestaltung von Anwendungslandschaften erarbeitet, eine Studie zu Werkzeugen zum Management von Unternehmensarchitekturen vorgestellt sowie abschließend das Design eines Werkzeugs für die Softwarekartographie gezeigt.

Die Gestaltung von Anwendungslandschaften in Abschnitt 5.1 diskutiert verschiedene Zielsetzungen, die Unternehmen bei der Weiterentwicklung ihrer Anwendungslandschaft verfolgen, und welche Auswirkungen unterschiedliche Strategien besitzen. Da der Aufbau eines Unternehmens, verteilte Produktionsstandorte, unterschiedliche Produktlinien etc. eine wesentliche Rolle bei der Gestaltung der Anwendungslandschaft spielen, wird dies bei den einzelnen Gestaltungsprinzipien berücksichtigt. Abschließend soll Abschnitt 5.1 die Frage beantworten „Kann es die perfekte Anwendungslandschaft geben?“.

Der Abschnitt 5.2 fasst die Resultate der „Enterprise Architecture Management Tool Survey 2005“ [se05a] zusammen, im Rahmen derer anhand eines Fragenkataloges und mehrerer Szenarios neun Werkzeuge evaluiert worden sind. Es wird die Vorgehensweise der Studie geschildert, die Resultate dargestellt und Schwächen der Werkzeuge aufgezeigt.

Das Design eines Werkzeugs für die Softwarekartographie, welches die vorgestellte Modellierungssprache umsetzt, wird in Abschnitt 5.3 präsentiert. Das Werkzeug beruht auf der in Abschnitt 4.4 vorgestellten Modelltransformation zur Kopplung des semantischen und des symbolischen Modells und zeigt die Implementierbarkeit des Visualisierungsmodells.

5.1. Gestaltung von Anwendungslandschaften

Methoden zur Gestaltung von Anwendungslandschaften und Unternehmensarchitekturen befinden sich in einer frühen Phase der Entwicklung, wird von der reinen Konsolidierung von Infrastrukturelementen und -services abgesehen. Diese Konsolidierung von Infrastrukturelementen und auch Infrastruktur-Services hat zur Konzentration und Homogenisierung von beispielsweise Hardwaresystemen, Datenspeichersystemen, Middlewaresystemen und E-Mail-Services geführt.

Dieser Abschnitt vernachlässigt die Ebenen der Infrastrukturelemente und -services, da es für diese bereits zahlreiche Produkte von Beratungshäusern, Outsourcing-Dienstleistern, Softwareproduktherstellern etc. gibt[1], und beschäftigt sich ausschließlich mit Methoden zur Gestaltung von Anwendungslandschaften und Unternehmensarchitekturen, die nach Abbildung 4.11 oberhalb der *Infrastruktur-Service-Schicht* liegen.

Im Folgenden werden zwei Methoden vorgestellt, die bei der Gestaltung von Anwendungslandschaften eingesetzt werden können. Diese sind zum einen die vertikale & horizontale Integration (siehe Abschnitt 5.1.1) und zum anderen das Fassadenprinzip für Domänen (siehe Abschnitt 5.1.2). Abschließend folgt in diesem Abschnitt eine Diskussion der Frage „Kann es die *optimale* Anwendungslandschaft geben?“ (siehe Abschnitt 5.1.3).

[1] Beispielsweise von Fujitsu Siemens Computers zur Konsolidierung von E-Mail-Services (`http://www.fujitsu-siemens.de/solutions/it_infrastructure_solutions/mail_consolidation/index.html`, abgerufen am 2007-05-28).

5.1.1. Vertikale & horizontale Integration

Die Gestaltungsprinzipien *vertikale Integration* und *horizontale Integration* stützen sich auf kartesische Softwarekarten, die entlang der x-Achse Geschäftsprozesse (Prozessunterstützungskarte), Geschäfts-Services oder Produkte visualisieren. Die Informationsobjekte der y-Achse sind von der Art des Unternehmens und den zu adressierenden Interessen abhängig.

Handelt es sich um ein Unternehmen, welches die gleichen Produkte in verschiedenen Ländergesellschaften vertreibt und/oder das gleiche Produkt an verschiedenen Standorten produziert, so sind sowohl die Vertriebsorganisationen als auch die Produktionsstandorte als Informationsobjekte für die y-Achse relevant. Teilweise spiegeln sich die Strukturen in der Produktentwicklung, der Produktfertigung und dem Produktvertrieb in der Organisationsstruktur eines Unternehmens wider, so dass die Organisationseinheiten als Informationsobjekte für die y-Achse verwendet werden.

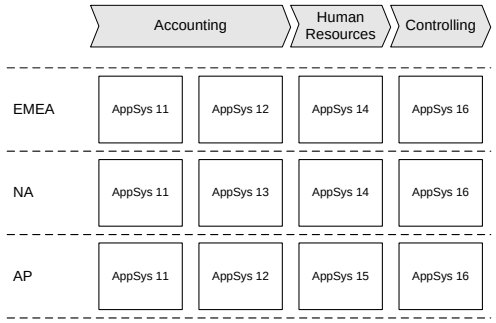


Abbildung 5.1.: Prozessunterstützungskarte ohne Darstellung der vertikalen Integration

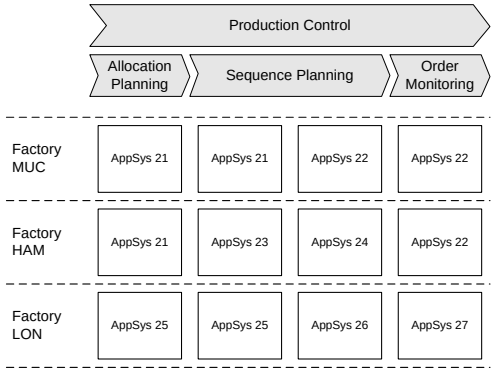


Abbildung 5.2.: Prozessunterstützungskarte ohne Darstellung der horizontalen Integration

Um die Gestaltungsprinzipien für vertikale und horizontale Integration zu erläutern, werden die kartesischen Karten in den Abbildungen 5.1 und 5.2 herangezogen[2], die beide Ist-Landschaften visualisieren:

- Abbildung 5.1 zeigt anhand einer Prozessunterstützungskarte die Unterstützung von Anwendungssystemen in der Kombination sekundäre Geschäftsprozesse (*Accounting*, *Human Resources* und *Controlling*) mit den Organisationseinheiten *EMEA* (*Europe*, *Middle East* und *Africa*), *NA* (*North America*) und *AP* (*Asia Pacific*).
- Abbildung 5.2 zeigt anhand einer Prozessunterstützungskarte die Unterstützung von Anwendungssystemen in der Kombination des primären Geschäftsprozesses *Produktionssteuerung* (engl. *Production Control*) inkl. Subprozessen mit den Organisationseinheiten *Factory MUC*, *Factory HAM* und *Factory LON*.

[2] Auf das Titelfeld und die Legende wurde in diesen Softwarekarten verzichtet.

Die einzelnen kartesischen Karten zeigen ausschließlich Anwendungssysteme, jedoch keine Anwendungssystemversionen. Sind die im Einsatz befindlichen Anwendungssystemversionen an verschiedenen Standorten unterschiedlich, ist die Analyse um Anwendungssystemversionen zu erweitern, so dass ein höherer Detaillierungsgrad entsteht.

5.1.1.1. Vertikale Integration

Wird von der Prozessunterstützungskarte in Abbildung 5.1 eine zweite Variante erstellt, welche die Geschäftsprozessunterstützung nicht bei jeder Organisationseinheit durch eine einzelne Gestaltungsmittelinstanz darstellt, sondern ein Rechteck vertikal gestreckt, so entsteht Abbildung 5.3. Aus dieser Abbildung ist im Vergleich zu Abbildung 5.1 leichter zu entnehmen, dass nicht alle Organisationseinheiten zur Unterstützung desselben Geschäftsprozesses dasselbe Anwendungssystem einsetzen.

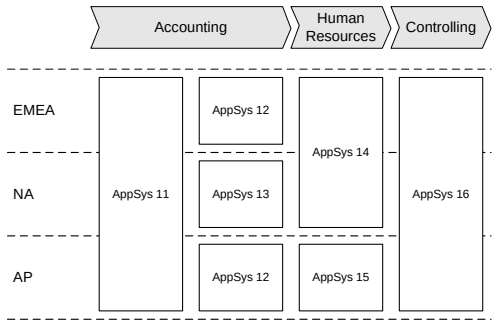


Abbildung 5.3.: Prozessunterstützungskarte mit Darstellung der vertikalen Integration

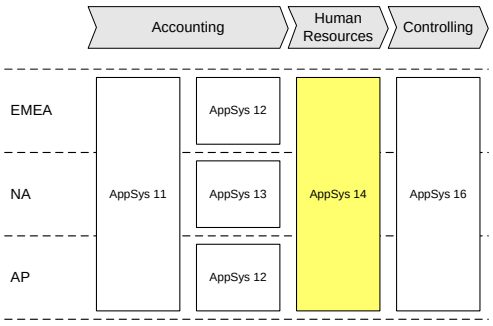


Abbildung 5.4.: Prozessunterstützungskarte mit erhöhtem Grad der vertikalen Integration

Vertikale Integration meint bei einer Prozessunterstützungskarte den Grad, zu welchem die Prozessunterstützung über mehrere, verschiedene Organisationseinheiten, Produkte oder Standorte gleich ist. In Abbildung 5.3 ist beispielsweise die vertikale Integration für den Geschäftsprozess *Human Resources* nicht vollständig. Die Organisationseinheit *AP* nutzt ein anderes Anwendungssystem als die Organisationseinheiten *EMEA* und *NA*.

Ziele beim Erhöhen des Grades der vertikalen Integration sind

- das Erzielen von Skaleneffekten bei verschiedenen Organisationseinheiten, Produkten, Standorten etc. und/oder
- das Reduzieren von Betriebs- und Wartungskosten durch das Reduzieren der Zahl von Anwendungssystemen und das Zusammenlegen von Ressourcen.

Kritisch bei einem Erhöhen der vertikalen Integration ist das Verringern von Flexibilität bei den Elementen auf der y-Achse der kartesischen Karte; im Beispiel aus Abbildung 5.3 ist dies die Flexibilität der einzelnen Organisationseinheiten. Neben einer Prozessunterstützungskarte kann auch eine kartesische Karte mit Geschäfts-Services auf der x-Achse

verwendet werden, um die vertikale Integration zu analysieren. Sowohl die obigen Ziele als auch der obige Kritikpunkt sind für eine derartige Variante der vertikalen Integration zutreffend.

Inwieweit eine vertikale Integration sinnvoll ist, sowie die Frage, welches der Anwendungssysteme abgelöst werden soll, kann durch eine Analyse der betroffenen Anwendungssysteme erfolgen. Hierzu eignen sich beispielsweise Portfoliomatrizen, bei denen auf der x-Achse die Betriebskosten und auf der y-Achse die Wartungskosten aufgetragen werden (siehe auch Abschnitt 2.1.3). Als zusätzliche Dimensionen für Größe und Hintergrundfarbe der Kreise eignen sich die Anzahl der Nutzer und der Grad der Erfüllung des SLAs. Derartige und weitere Portfoliomatrizen zum Vergleich mehrerer Anwendungssysteme finden sich bei Keller [Ke06] und Niemann [Ni05].

Wird eine vertikale Integration bei dem Geschäftsprozess *Human Resources* verfolgt, so kann eine Soll-Landschaft, wie in Abbildung 5.4 gezeigt, erstellt werden, wenn das Anwendungssystem *AppSys 14* für die Integration favorisiert wird. Die Unterschiede zur Ist-Landschaft sind zusätzlich farblich hervorgehoben.

5.1.1.2. Horizontale Integration

Analog zur vertikalen Integration bezieht sich die horizontale Integration auf den Grad der Unterstützung bei einer kartesischen Karte in x-Richtung. Abbildung 5.5 zeigt die horizontale Integration durch Ausdehnung der Rechtecke in x-Richtung für die Prozessunterstützungskarte in Abbildung 5.2. Das Anwendungssystem *AppSys 21* unterstützt beispielsweise die Geschäftsprozesse *Allocation Planning* und *Sequence Planning* bei der Organisationseinheit *Factory MUC*. Wie auch bei der vertikalen Integration tritt das Problem in der Darstellung auf, dass die Integration nur bei in der Visualisierung benachbarter Elemente der x-Achse dargestellt werden kann.

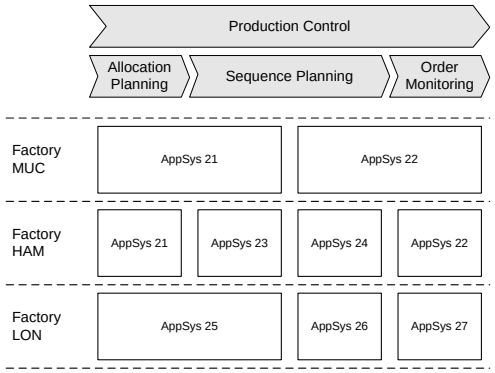


Abbildung 5.5.: Prozessunterstützungskarte mit Darstellung der horizontalen Integration

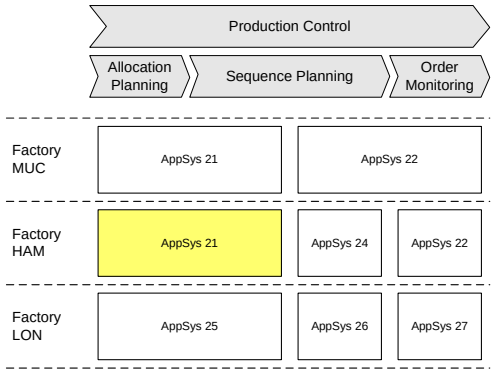


Abbildung 5.6.: Prozessunterstützungskarte mit erhöhtem Grad der horizontalen Integration

Die Ziele beim Erhöhen der horizontalen Integration sind, wie bei der vertikalen Integration, das Erzielen von Skaleneffekten und das Reduzieren von Betriebs- und Wartungskosten. Das Erzielen von Skaleneffekten betrifft jedoch bei der horizontalen Integration die gleichartige Unterstützung von Geschäftsprozessen oder Geschäfts-Services.

Die horizontale Integration ist kritischer zu betrachten als eine vertikale Integration, da beispielsweise die Unterstützung mehrerer Geschäftsprozesse durch das gleiche Anwendungssystem nicht nur die Flexibilität beschränkt, sondern ebenso das Prinzip der Trennung von Interessen (engl. *Separation of Concerns*) verletzt. Werden Geschäftsprozesse in einem hohen Detaillierungsgrad (beispielsweise Ebene 3 oder 4) betrachtet, so werden wenige, stark vernetzte und aufeinanderfolgende Geschäftsprozesse oftmals durch ein gleichartiges Anwendungssystem unterstützt. Betrifft dies jedoch auch Geschäftsprozesse, die nicht aufeinanderfolgen, so können zwar Medienbrüche und die Anzahl von Kommunikationsbeziehungen zwischen Anwendungssystemen durch eine horizontale Integration verringert werden, dies aber zu Lasten der Trennung von Interessen und einem Sinken der Kohäsion.

Eine Soll-Landschaft, die bei der Organisationseinheit *Factory HAM* die horizontale Integration erhöht, ist in Abbildung 5.6 dargestellt. Diese Art der horizontalen Integration erhöht gleichzeitig die vertikale Integration, da das Anwendungssystem *AppSys 21* für den Geschäftsprozess *Sequence Planning* ebenso von der Organisationseinheit *Factory MUC* eingesetzt wird.

5.1.1.3. Vertikale Integration vs. horizontale Integration

Inwieweit eine vertikale Integration einer horizontalen Integration oder andersherum vorzuziehen ist, kann nicht pauschal beantwortet werden. Die vertikale Integration eignet sich insbesondere für Geschäftsprozesse, die kein Diversifikationsmerkmal für das Unternehmen besitzen. Prozesse mit einem Diversifikationsmerkmal benötigen eine hohe Flexibilität, um Innovationen abbilden zu können und flexibel auf Änderungsanforderungen zu reagieren. Eine vertikale Integration bedingt einen erhöhten Abstimmungsbedarf zwischen verschiedenen Organisationseinheiten, Standorten etc. und reduziert somit die Flexibilität der einzelnen Einheit.

Für die vertikale Integration eignen sich somit insbesondere sekundäre Geschäftsprozesse, die einen Bezug zur Buchhaltung, Fakturierung, Personalwesen etc. besitzen. Derartige Prozesse sind im Vergleich zu primären Geschäftsprozessen *starr* und unterliegen geringem Flexibilitätsbedarf. In diesen Prozessen werden oftmals betriebliche Standardsoftwaresysteme eingesetzt, die sich auch für den Einsatz über Organisationseinheiten hinweg eignen.

Die Entscheidung für oder gegen eine horizontale Integration muss mit entsprechenden Bedarfsanalysen gestützt werden. Primäre Prozesse sollen schnell an neue Anforderungen angepasst werden, so dass eine prozessübergreifende Unterstützung die Flexibilität verringert. Vorteilhaft bei der horizontalen Integration ist das Reduzieren von Medienbrüchen zwischen aufeinanderfolgenden Geschäftsprozessen. Dieser Problematik, die auch durch standardisierte Austauschformate adressiert wird, wirken Initiativen im

Kontext der Implementierung von Service-orientierten Architekturen, wie BPEL, entgegen [An03].

Das Einführen einer Middlewaretechnologie, welche die Kommunikation zwischen Anwendungssystemen standardisiert und eine Rekombination von Services verbessert, stellt zwar kein Erhöhen der horizontalen Integration dar, jedoch werden dieselben Ziele verfolgt. Aus diesem Grunde ist die Betrachtung eines Clusters von aufeinanderfolgenden Geschäftsprozessen mit einer engen Kopplung in der horizontalen Ebene sinnvoll. Der Cluster stellt eine logische Zusammenfassung von Anwendungssystemen dar und sollte derart gebildet werden, um neue Anforderungen im Kontext des Clusters zu betrachten. Kennzahlen des Clusters, wie beispielsweise Betriebskosten, Wartungskosten und Kritikalität der Systeme, ermöglichen eine zusammenhängende Bewertung der eng gekoppelten Anwendungssysteme.

5.1.2. Fassadenprinzip für Domänen

Das Bilden von logischen Gruppen in der Anwendungslandschaft wurde bereits entlang der Softwarekarten in Abschnitt 3.3 gezeigt. In den beiden Softwarekarten der Versicherungskonzerne (siehe Abbildungen 3.10 und 3.12) wurden die Anwendungssysteme verschiedenen (Geschäfts-) Bereichen zugeordnet. Im Folgenden werden diese Gruppierungen als *Domänen* bezeichnet, um auszudrücken, dass die Anwendungssysteme in einer Domäne einen fachlichen Zusammenhang besitzen. Des Weiteren liegt die fachliche Verantwortung von Anwendungssystemen einer Domäne häufig bei einer bestimmten Fachabteilung. Das Bilden von Domänen für Anwendungssysteme wird auch von Laartz et al. [LSV00] beschrieben. Laartz et al. ziehen dabei eine Analogie zum Städtebau und vergleichen eine Domäne mit einem Stadtviertel.

Laartz et al. führen des Weiteren aus, dass die Kommunikationsbeziehungen zwischen Anwendungssystemen eine höhere Lebensdauer als die Anwendungssysteme selbst haben. Die Lebensdauer bezieht sich dabei auf die fachliche Funktionalität der Anwendungssysteme, welche Versionen von Anwendungssystemen oder auch die Migration der Funktionalität in ein anderes Anwendungssystems überdauern.

Hierdurch kommt den Kommunikationsbeziehungen in einer Anwendungslandschaft eine besondere Bedeutung zu. Das Planen von Kommunikationsbeziehungen innerhalb einer Domäne hat dabei eine gegenüber den domänenübergreifenden Kommunikationsbeziehungen untergeordnete Rolle. Wie auch beim Städtebau, ist das Planen einer neuen Straße in einem Neubaugebiet im Vergleich weniger komplex und priorisiert, als die Anbindung des Neubaugebietes an die gesamte Infrastruktur (gemeint sind hier Straßen) der Stadt. Diese *großen* Hauptverkehrswege sind das Analogon zu Kommunikationsbeziehungen zwischen Domänen. Die Kommunikationsbeziehungen innerhalb einer Domäne sind die kleinen Straßen eines Stadtteils und können lokal betrachtet und verändert werden.

Das Fassadenprinzip für Domänen verlangt, dass die Domänen nach außen eine einheitliche Schnittstelle für andere Domänen definieren. Dieses Muster ist im Software-Engineering von Gamma et al. [Ga95] unter dem gleichen Namen *Facade* eingeführt wor-

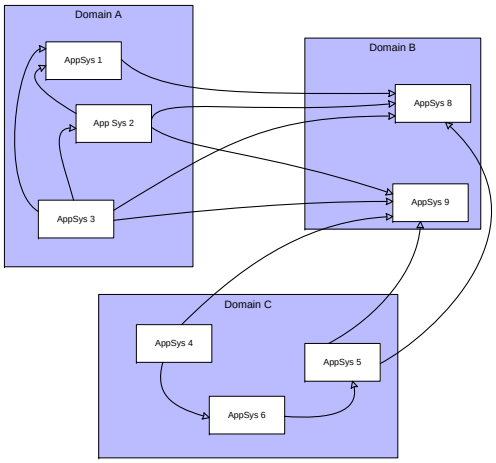


Abbildung 5.7.: Clusterkarte ohne Verwendung des Fassadenprinzips für Domänen

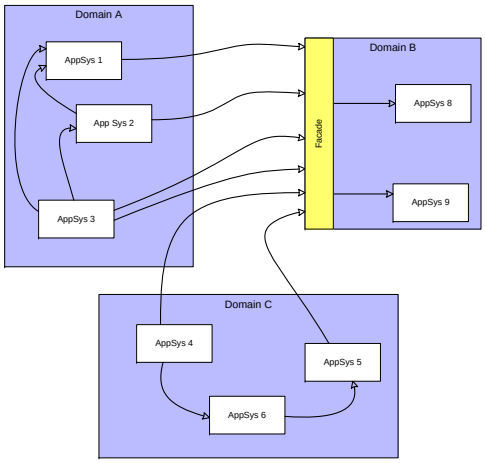


Abbildung 5.8.: Clusterkarte mit Verwendung des Fassadenprinzips für Domänen

den und wird beim Fassadenprinzip für Domänen auf ein höheres Abstraktionsniveau – auf die Anwendungslandschaft – übertragen.

Abbildung 5.7 zeigt einen Ausschnitt einer Anwendungslandschaft ohne die Anwendung des Fassadenprinzips, wobei die Kommunikationsbeziehungen derart zu verstehen sind, dass keine einheitliche Schnittstelle existiert. Unter Anwendung des Fassadenprinzips entsteht Abbildung 5.8, in der eine einheitliche Schnittstelle für die anderen Domänen definiert und die Fassade (engl. *Facade*) eingeführt wird. Die Analyse der Kommunikationsbeziehungen ohne Betrachtung der einheitlichen Schnittstellen ist ebenso wichtig, da nur so zu erkennen ist, welche Anwendungssysteme einen Daten- oder Kontrollfluss besitzen.

Der Vorteil von Fassaden liegt in der Reduktion der Schnittstellenkomplexität. Die einzelnen Anwendungssysteme in einer Domäne verändern sich durch neue oder geänderte Anforderungen. Wird ein Anwendungssystem geändert, so müssen gegebenenfalls alle Anwendungssysteme, die eine Kommunikationsbeziehung zu dem Anwendungssystem besitzen, angepasst werden. In Abbildung 5.8 ist erkennbar, dass eine Fassade die Bündelung von Kommunikationsbeziehungen ermöglicht. Der Zugriff des Anwendungssystems *AppSys 1* auf ein Anwendungsystem der Domäne *Domain B* erfolgt ausschließlich über die Fassade, so dass ein Zugriff auf die einzelnen Anwendungssysteme (auch mehrere) transparent erfolgt. Die Stabilität der Fassade erhöht hierbei auch die Stabilität der Zugriffschnittstelle für das Anwendungssystem *AppSys 1*, da domäneninterne Änderungen nicht zu einer Veränderung an der Fassade führen müssen.

Dieses Konzept findet beim Aufbau einer Service-orientierten Architektur seine Anwendung, bei dem grobgranulare Geschäfts-Services gebildet werden, die eine hohe Stabilität besitzen sollen. Die Realisierung von Fassaden kann sowohl auf rein konzeptueller Ebene durch das Einführen von Richtlinien und Standards erfolgen oder durch Middleware-

systeme gestützt werden, wie sie beim *Enterprise Application Integration* [Ke02] zum Einsatz kommen.

5.1.3. Diskussion zur optimalen Anwendungslandschaft

Eine *optimale* Anwendungslandschaft wird in dieser Arbeit mit einem Optimum hinsichtlich Effektivität und Effizienz gleichgesetzt. Das Optimum hinsichtlich Effektivität wird erreicht, wenn die Anwendungssysteme die Geschäftsziele des Unternehmens bestmöglich unterstützen. Dies bedeutet beispielsweise, dass die Anwendungssysteme der Anwendungslandschaft das Produzieren der Güter eines Produktionsunternehmens derart unterstützen, dass die Vorgaben der Fachabteilungen hinsichtlich Entwicklung, Produktion, Vertrieb etc. entsprechend der Anforderungen umgesetzt sind.

Ein Optimum hinsichtlich Effizienz bedeutet, dass die Kosten der Anwendungssysteme minimal sind. Die Effizienz der IT ist derzeit immer noch schwer messbar und wird ausschließlich auf die IT-Kosten zurückgeführt. Jedoch stellt die adäquate Verrechnung von IT-Kosten ein Problem dar, da eine Prozesskostenrechung für die IT, die beinhaltet, dass die Entwicklungs-, Betriebs- und Wartungskosten eines Anwendungssystems einer Geschäftsprozessunterstützung zugerechnet werden können, noch am Anfang steht. Das Thema IT-Controlling [Kü03, Kr05], welches diese Thematik adressiert, ist derzeit in Unternehmen nicht so etabliert, wie z. B. die Kostenrechnung bei den Prozess- und Produktionskosten in den Wirtschaftswissenschaften.

Des Weiteren ist die Flexibilität und Anpassbarkeit der IT an neue Zielvorgaben, dies sowohl aus dem Geschäft als auch aus der IT, nur unzureichend quantifizierbar. Die Effizienz und auch Effektivität der IT, auf wechselnde Anforderungen adäquat zu reagieren, ist eine stete Herausforderung. Das Konzept der Service-orientierten Architekturen versucht dies zu adressieren und verspricht hierbei, mittels einer (Re-) Konfigurierbarkeit von (Geschäfts-) Services eine Verbesserung. Ob dies tatsächlich eine Verbesserung darstellt und inwieweit dies auch zu messbaren Veränderungen führt, ist abzuwarten.

Die Wechselbeziehungen von IT-Effizienz und IT-Effektivität stellt Helbig [He07] in Abbildung 5.9 dar. Helbig fokussiert hierbei auf die Kosten und stellt diese in Zusammenhang mit den Unternehmensergebnissen.

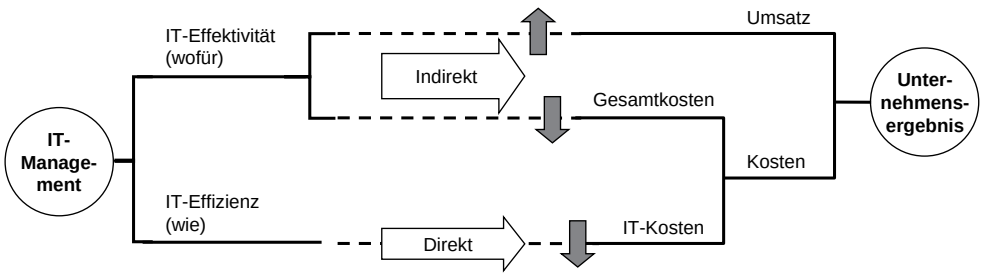


Abbildung 5.9.: Effektivität und Effizienz als Steuerungsgrößen des IT-Managements [He07]

Die Frage nach der *optimalen* Anwendungslandschaft im Hinblick auf optimale Effektivität und optimale Effizienz der Anwendungslandschaft kann für eine Momentaufnahme beantwortet werden. Existieren keine neuen oder geänderten Anforderungen an die Anwendungslandschaft und wurden die Anforderungen aus der Vergangenheit vollständig umgesetzt, ist die Anwendungslandschaft optimal effektiv. Sind die Kosten für den Betrieb der Anwendungslandschaft minimal, so ist diese gleichzeitig optimal effizient.

Da diese Situation aber nicht der Realität entspricht und die Anforderungen an die IT einem steten Wandel unterliegen, muss die Frage nach der optimalen Anwendungslandschaft zeitbezogen beantwortet werden. Eine optimale Anwendungslandschaft müsste somit auf den Wandel optimal effektiv und optimal effizient reagieren können. Da das Vorhersehen aller neuen Anforderungen an die IT einem Blick in eine *Glaskugel* gleichkommt, folgt die These: „Die *optimale* Anwendungslandschaft existiert nicht."

In der Situation von Zusammenschlüssen (*Mergers*), Zukäufen (*Acquisitions*), Auslagerungen (*Outsourcings*) und Ausgliederungen (*Carve Outs*) müsste eine optimale Anwendungslandschaft dies vorhersehen können. Dass jedoch beispielsweise Zusammenschlüsse und Zukäufe in unterschiedlichen Folgerungen für die Anwendungslandschaft resultieren, beschreibt auch Keller [Ke06]. Keller unterscheidet hier zwischen *Neubau*, *Cherry Picking* und *Dampfwalze*, wobei jede Variante unterschiedlich (politisch) motiviert ist und verschiedene Folgen besitzt.

Ziel des IT-Managements sollte es sein, eine Balance zwischen Effektivität und Effizienz zu erreichen, die in einer flexiblen Anwendungslandschaft mündet, um der optimalen Anwendungslandschaft möglichst nahezukommen. Das IT-Management kann hierbei beispielsweise durch Werkzeuge zum Management der Unternehmensarchitektur unterstützt werden, die Gegenstand des folgenden Abschnitts 5.2 sind.

5.2. Werkzeuge zum Management von Anwendungslandschaften und Unternehmensarchitekturen

Grundlage für diesen Abschnitt ist eine im Jahr 2005 durchgeführte Studie zu Werkzeugen des Managements von Unternehmensarchitekturen mit dem Titel „Enterprise Architecture Management Tool Survey 2005" (im Folgenden EAMTS2005). Die Studie wurde mit Unterstützung der folgenden Industriepartner durchgeführt, die der Studie als Informationsgeber zur Verfügung standen: AMB Generali Informatik Service, BMW Group, Deutsche Börse Systems, Deutsche Post, HVB Systems, Kühne + Nagel, Münchener Rück, Siemens, T-Com und TUI.

Die Studie hat neun Werkzeuge zum Management von Unternehmensarchitekturen untersucht, wobei entsprechend des Schaubilds in Abbildung 2.2 für Prozesse zum Management von Anwendungslandschaften nicht nur der Prozess *Management der Unternehmensarchitektur* untersucht wurde. Auch die Werkzeugunterstützung für die anderen Managementprozesse in Abbildung 2.2 wurde teilweise untersucht, wie die folgenden Absätze zeigen werden.

Hersteller	Werkzeug(e)	URL
AB+ Conseil	SOLU-QIQ	http://www.abplusconseil.com
Adaptive, Ltd.	Adaptive EAM	http://www.adaptive.com
Agilense, Inc.	EA WebModeler	http://www.agilense.com
alfabet AG	planningIT	http://www.alfabet.de
ASG, Inc.	ASG-Rochade	http://www.asg.com
BOC GmbH	ADOit	http://www.boc-eu.com
Casewise Ltd.	Corporate Modeler Suite & ITAA	http://www.casewise.com
Flashline, Inc.	Flashline 4	http://www.flashline.com
GoAgile	GoAgile MAP	http://www.goagile.com
IDS Scheer AG	ARIS Toolset	http://www.ids-scheer.com
LogicLibrary	LogiScan & Logidex	http://www.logiclibrary.com
MEGA International SA	MEGA	http://www.mega.com
Mercury Interactive Corp.	IT Governance Center	http://www.mercury.com
Orbus Software iServer for EA	iServer	http://www.orbussoftware.com
process4.biz[a]	process4.biz	http://www.process4.biz
Proforma Corp.	ProVision Modeling Suite	http://www.proformacorp.com
Telelogic AB[b]	System Architect	http://www.telelogic.com
Troux Technologies, Inc.[c]	Metis	http://www.troux.com
Visible Systems Corporation	Visible Advantage	http://www.visible.com

Tabelle 5.1.: EAMTS2005: Lange Liste von Werkzeugen [se05a]

[a] process4.biz GmbH firmierte früher unter EiTAM, Inc.; das Produkt *EiTAM* wurde in *process4.biz* umbenannt.
[b] Telelogic AB hat Popkin Software, Inc. übernommen
[c] Troux Technologies, Inc. hat die Produktgruppe *Metis* von Computas AS übernommen.

Zu Beginn der Studie wurde anhand einer Internetrecherche und der Sichtung von Quellen (bspw. Gartner [Ja05], Forrester Research [Pe04]) eine Liste von möglichen zu analysierenden Werkzeugen erstellt (die so genannte *Long List*), die in Tabelle 5.1 wiedergegeben ist. Diese Liste wurde in Abstimmung mit den Industriepartnern gekürzt (die so genannte *Short List*), um die Anzahl der zu analysierenden und zu evaluierenden Werkzeuge zu reduzieren (siehe Tabelle 5.2). Eine Analyse, wie sie im Folgenden beschrieben wird, konnte aus Gründen der Projektlaufzeit und von Ressourcen nicht mit allen Werkzeugen der *Long List* durchgeführt werden.

Hersteller	**Werkzeug(e) (Version)**
Adaptive, Ltd.	Adaptive EAM (v3.0.306)
alfabet AG	planningIT (v1.0)
BOC GmbH	ADOit (v2.0)
Casewise, Inc.	Corporate Modeler Suite (v9.1) & IT Architecture Accelerator (v3.6)
IDS Scheer AG	ARIS Toolset (v6.2.3)
MEGA International SA	MEGA (v6.1)
process4.biz GmbH	process4.biz (v4.07)
Telelogic AB	System Architect (v10.1.11)
Troux Technologies, Inc.	Metis (v3.6)

Tabelle 5.2.: EAMTS2005: Gekürzte Liste von Werkzeugen inkl. Versionen [se05a]

Die folgenden Abschnitte fassen den Aufbau der Studie und das Vorgehen bei der Analyse (Abschnitt 5.2.1) sowie die Ergebnisse (siehe Abschnitt 5.2.2) und die Erkenntnisse (siehe Abschnitt 5.2.3) der EAMTS2005 zusammen.

5.2.1. Aufbau und Vorgehen der Werkzeugstudie

Im Gegensatz zu Studien von Gartner [Ja05] oder Forrester Research [Pe04] war es das Ziel der EAMTS2005, eine tiefgehende Analyse von Werkzeugfunktionalitäten und ihre Anwendung im Management von Unternehmensarchitekturen durchzuführen. Die EAMTS2005 ist hierbei nicht als Konkurrenz zu den Studien von Gartner oder Forrester Research zu sehen, sondern als Ergänzung.

Grundsätzlich baute das Vorgehen der EAMTS2005 auf einen zweistufigen Ansatz, der einerseits einen Fragenkatalog und andererseits simulierbare Szenarios beinhaltet. Anhand des Fragenkatalogs wurden die Werkzeuge hinsichtlich bestimmter Anforderungen analysiert, die an ein derartiges Werkzeug gestellt werden. Da die Ansätze der Werkzeuge zur Unterstützung des Managements von Unternehmensarchitekturen unterschiedlich sind, wurde die Analyse mittels des Fragenkatalogs durch die Simulation von Szenarios unterstützt, durch welche sowohl notwendige Funktionalitäten als auch typische Aufgaben beim Management von Unternehmensarchitekturen simuliert wurden.

Die Industriepartner fungierten sowohl bei dem Fragenkatalog als auch bei den Szenarios als Informationsgeber, um den Praxisbezug zu gewährleisten und die Anwenderperspektive adäquat abzubilden.

Ziel der Evaluation nach der Analyse war keine einfache Rangordnung der Werkzeuge, sondern das Aufzeigen ihrer Charakteristika und ihrer Fähigkeiten bei der Unterstützung für unterschiedliche Aufgaben im Management von Unternehmensarchitekturen und verwandten Managementprozessen. Die EAMTS2005 wurde im Zeitraum Januar bis August 2005 durchgeführt, die Publikation des 338-seitigen Ergebnisberichts erfolgte im September 2005.

5.2.1.1. Fragenkatalog

Der Fragenkatalog der EAMTS2005 besteht aus ca. 400 Fragen, die in mehrere Kategorien aufgeteilt sind. Der Fragenkatalog wurde den Herstellern über eine Webanwendung zur Verfügung gestellt, wobei die Anwendung dynamisch Abhängigkeiten von Fragen auflöst und den Workflow des antwortenden Herstellers anpasst. Besitzt ein Werkzeug beispielsweise keine Weboberfläche, so wurden die Folgefragen ausgeblendet, die sich mit weiteren Web-spezifischen Technologien (Plugins, Webbrowserkompatibilität etc.) beschäftigen.

Die Hauptkategorien unterteilen den Fragenkatalog in funktionale Anforderungen (siehe Abschnitt 5.2.1.1.1) sowie technische und weitere Anforderungen (siehe Abschnitt 5.2.1.1.2).

5.2.1.1.1. Funktionale Anforderungen

Die funktionalen Anforderungen analysieren die Werkzeuge aus einer Perspektive, die losgelöst sind von ihrer Umsetzung und ihrer Anwendung in Managementprozessen. Diese Anforderungen wurden mittels der folgenden Kategorien gruppiert:

- Methode (engl. *Methodology*)
- Visualisierung (engl. *Visualization*)
- Integration mit verwandten Domänen (engl. *Integration with related Domains*)
- Integration in andere Werkzeuge (engl. *Integration with other Tools*)
- Berichterstellung (engl. *Reporting*)
- Unterstützung von kollaborativem Arbeiten (engl. *Collaboration Support*)
- Metadaten zur Architekturdokumentation (engl. *Metadata for Architectural Description*)
- Internationalisierung/Lokalisierung (engl. *Internationalization/Localization*)
- Navigation im Modell (engl. *Navigation in the Model*)

Mit der Kategorie *Methode* wird evaluiert, in welchem Umfang in den Werzeugen Methoden zum Management von Unternehmensarchitekturen implementiert sind und inwieweit die vordefinierten Methoden an unternehmensspezifische Anforderungen angepasst werden können. Des Weiteren wurden in dieser Kategorie die Metamodelle[3] der Werk-

[3] Im Begriffsapparat der Softwarekartographie das Informationsmodell.

zeuge betrachtet, die in einer Methode verwendet werden. Fragen bezüglich der Größe der Metamodelle und die Anwendung von Frameworks (bspw. Zachman [Za06] oder DoDAF [Do04]) sind enthalten. Hinsichtlich der Anpassbarkeit der Metamodelle wurden des Weiteren die Metamodellierungsfähigkeiten der Werkzeuge (bspw. Unterstützung von Multiplizitäten bei Assoziationen, Vererbung von Attributen etc.) hinterfragt.

Die Kategorie *Visualisierung* beinhaltet Fragen, die sich mit den Fähigkeiten der Werkzeuge beschäftigen, Informationen mittels definierter Diagrammtypen in einer verständlichen Form zu visualisieren. Hierzu zählen keine tabellarischen Berichte, sondern beispielsweise die BCG-Matrix und die Softwarekarten. Erfragt wurden auch die Werkzeugfähigkeiten, komplexe Darstellungen zu vereinfachen, indem beispielsweise Informationen gefiltert werden. In dieser Kategorie wurden sowohl die Standardimplementierungen (*Out of the Box*) von Diagrammtypen als auch die Adaptionsfähigkeiten der Diagrammtypen berücksichtigt.

Die Fragen der Kategorie *Integration mit verwandten Domänen* befassen sich mit den Fähigkeiten der Werkzeuge, Aufgaben aus verwandten Domänen wahrzunehmen. Hierzu zählten die objektorientierte Modellierung mit UML [OM05d], die Modellierung von Datenbanken mittels E/R-Diagrammen [Ch76] (oder verwandter Notationen) und die Geschäftsprozessmodellierung mittels EPKs [KNS92] oder der BPMN [OM06a].

Die Kategorie *Integration in andere Werkzeuge* evaluiert, inwieweit eine Einbindung der analysierten Werkzeuge mittels Technologien, wie *Java Portlets*, *Microsoft Windows Sharepoint Service*, *SAP Portals* etc., möglich ist. Im Vordergrund steht die Integration der Oberflächen sowie der Authentifizierungs- und Autorisierungsfunktionen.

Berichterstellung beinhaltet Fragestellungen, welche die Werkzeugfähigkeiten zum Erstellen von tabellarischen Auswertungen sowie der graphischen Aufbereitung dieser Tabellen durch Kiviat-Diagramme (auch Spinnen- oder Netzdiagramm), Kreissektorendiagramme etc. evaluiert.

Die Kategorie *Unterstützung von kollaborativem Arbeiten* beinhaltet Fragen zur Mehrbenutzerfähigkeit der Werkzeuge, die Abbildung und Konfigurierbarkeit von Rollen und Rechten inkl. dem Detaillierungsgrad sowie die Unterstützung des parallelen Arbeitens auf einer Datenbasis. Zusätzlich wurden Möglichkeiten zum Offline-Arbeiten (Import/-Export von Teilen der Datenbasis) und die hiermit verbunden Fähigkeiten zur Konfliktlösung bei konkurrierenden Änderungen einzelner Datensätze erfragt.

Metadaten zur Architekturdokumentation fragt nach Fähigkeiten, welche es erlauben, die Viewpoints (siehe Abschnitt 3.1), die Stakeholder, die Interessen etc. mit Metadaten zu ergänzen. Vor dem Hintergrund der Konzepte des IEEE 1471 [IE00] können diese Metadaten einzelne Elemente (Stakeholder, Interessen Viewpoints, Modelle etc.) dokumentieren und mit zusätzlichen Informationen versehen.

Die Kategorie *Internationalisierung/Lokalisierung* beinhaltet Fragen zu verschiedenen Zahlenformaten, Datumsformaten, Definieren von mehrsprachigen Texten etc.

Navigation im Modell hinterfragt die Möglichkeiten zur Navigation in Diagrammen zu detaillierten Beschreibungen der Informationsobjekte, das Navigieren in baumartigen

Strukturen oder das Navigieren von einem Informationsobjekt zu seiner Visualisierung etc.

5.2.1.1.2. Technische und weitere Anforderungen

Ein Werkzeug zum Management von Unternehmensarchitekturen muss sich in die existierende IT-Infrastruktur eines Unternehmens einbetten, um beispielsweise existierende Informationsquellen zu integrieren und die Datensicherheit zu gewährleisten. Die technischen Anforderungen adressieren dies und beinhalten Fragen der folgenden Kategorien:

Architektur (engl. *Architecture*) bestehend aus Fragen zu Softwarearchitektur, Komponenten, Programmiersprachen, Netzwerkanforderungen, Programmierschnittstellen etc.

Benötigte Infrastruktur (engl. *Required Infrastructure*) bestehend aus Fragen zu Middleware, Hardware etc.

Sicherheit (engl. *Security*) bestehend aus Fragen zu verschlüsselter Kommunikation, Integration in Single-Sign-On-Systeme, Protokollierung etc.

Performanz (engl *Performance*) bestehend aus Fragen zu Netzbandbreite etc.

Visualisierungstechniken (engl. *Visualization Techniques*) bestehend aus Fragen zu verwendeten Formaten, wie SVG, Microsoft PowerPoint, Microsoft Visio etc.

Wartung (engl. *Maintenance*) bestehend aus Fragen zu Sicherungs- und Wiederherstellungsverfahren etc.

Import, Export und Einbinden externer Datenquellen (engl. *Import, Export and Linking external Datasources*) bestehend aus Fragen zu Import-/Export-Formaten für Geschäftsprozessmodellierungswerkzeuge, Systemsmanagementsysteme, Datenbanksysteme etc.

Drucken (engl. *Printing*) bestehend aus Fragen zum Druck auf Standardpapierformaten.

Die weiteren Anforderungen beziehen sich auf Fragen zu dem Unternehmen des anbietenden Werkzeugherstellers, der Lizenzpolitik, den Support-Optionen etc. Diese Daten wurden in den Evaluierungsergebnissen der EAMTS2005 nicht berücksichtigt und wurden ausschließlich den Industriepartnern der Studie zur Verfügung gestellt.

5.2.1.2. Szenarios

Da eine alleinige Evaluierung der Werkzeuge aufgrund des Fragenkatalogs für nicht ausreichend erachtet wurde, um detaillierte Aussagen über die Eignung eines Werkzeugs zum Management von Unternehmensarchitekturen treffen zu können, wurden neben dem Fragenkatalog Szenarios zur Simulation mit den Werkzeugen erarbeitet.

Die Szenarios, die in Zusammenarbeit mit den Industriepartnern entwickelt wurden, sind in zwei Kategorien aufgeteilt. Die erste Kategorie von Szenarios analysiert die Werkzeuge hinsichtlich spezifischer Funktionalitäten, wie beispielsweise Visualisierungstechniken, Metamodellierungsfähigkeiten etc. (siehe Abschnitt 5.2.1.2.1). Diese Szenarios verifizieren und validieren die Aussagen der Werkzeughersteller im Fragenkatalog und stellen diese in einen Anwendungskontext.

Die zweite Kategorie von Szenarios analysiert die Unterstützung der Werkzeuge für bestimmte Aufgaben beim Management von Unternehmensarchitekturen und verwandten Managementprozessen (siehe Abschnitt 5.2.1.2.2).

In der EAMTS2005 [se05a] ist jedes Szenario detailliert beschrieben. Die Beschreibung der einzelnen Szenarios beginnt mit den adressierten Interessen, den aus diesen abgeleiteten, feingranulareren Fragestellungen sowie den Arbeitsschritten zur Simulation. Abschließend werden die zu erzielenden Ergebnisse in jedem Szenario dokumentiert, die sich bei der Simulation ergeben sollen.

Die zu erzielenden Ergebnisse werden als Leitlinien erachtet, die nicht im exakten Maße erfüllt werden müssen. Jedoch müssen die Werkzeuge die Erstellung semantisch äquivalente Ergebnisse ermöglichen, um die Interessen des Szenarios korrekt zu adressieren. Es wurde somit alternativen Lösungen verschiedener Werkzeuge Rechnung getragen.

5.2.1.2.1. Szenarios zur Analyse spezifischer Funktionalitäten

Die Szenarios zur Analyse spezifischer Funktionalitäten in der EAMTS2005 sind:

- Visualisierung der Anwendungslandschaft (engl. *Visualization of the Application Landscape*)
- Visualisierung von Kennzahlen (engl. *Visualization of Measures*)
- Vereinfachter lesender Zugriff (engl. *Simplified Access for Readers*)
- Editieren eines Modells unter Verwendung eines externen Editors (engl. *Editing Model Data using an External Editor*)
- HTML-Export (engl. *HTML Export*)
- Adaption des Metamodells (engl. *Metamodel Adaptation*)
- Visualisierung einer großen Anwendungslandschaft (engl. *Large Scale Application Landscape Visualization*)

Im Folgenden wird das Szenario *Visualisierung der Anwendungslandschaft* detailliert dargestellt, da es der wesentliche Einstiegspunkt für die weiteren Szenarios ist. Bei diesem Szenario werden initial die Informationen über die Anwendungslandschaft in das Werkzeug eingegeben, die von den anderen Szenarios verwendet werden. Der Aufbau der weiteren Szenarios ist analog zu den im Folgenden vorgestellten.

Die Interessen des Szenario *Visualisierung der Anwendungslandschaft* sind wie folgt formuliert:

> Das Kaufhaus SoCaStore will einen Überblick seiner Anwendungslandschaft und Unternehmensarchitektur erhalten. Dies soll erreicht werden durch die Erstellung von drei Softwarekarten, die verschiedene Merkmale visualisieren. Hierzu sind eine Clusterkarte, eine Prozessunterstützungskarte und eine Zeitintervallkarte zu erstellen.

Durch die Simulation sollen die folgenden Fragestellungen beantwortet werden:

- Stellt das Werkzeug ein Metamodell bereit, welches alle notwendigen Informationen über die Anwendungslandschaft zur Adressierung der Interessen abbilden kann?
- Stellt das Werkzeug geeignete und flexible Import-Mechanismen bereit und welche Datenformate werden hierbei unterstützt?
- Welche Möglichkeiten zur Datenmanipulation werden von dem Werkzeug bereitgestellt und welche Mechanismen (Editieren von graphischen Diagrammen, Editieren von Tabellen etc.) werden unterstützt?
- Stellt das Werkzeug Mechanismen zur Erzeugung von Visualisierungen bereit, wie sie in den Interessen (Clusterkarte, Prozessunterstützungskarte, Zeitintervallkarte) gefordert werden und können diese Darstellungen automatisiert erzeugt werden?
- Erlaubt das Werkzeug manuelle Veränderungen der Visualisierungen und ist das Werkzeug in der Lage, manuelle Veränderungen zu erhalten, wenn die Darstellung erneut erzeugt wird?

Die Simulation des Szenarios erfolgt mittels der folgenden Schritte, welche die zu erreichenden Ergebnisse referenzieren:

1. Das Metamodell des Werkzeugs wird überprüft, um festzustellen, ob die Informationen des Kaufhauses SoCaStore zur Simulation mit Anwendungssystemen, Organisationseinheiten, Geschäftsprozessen, Anwendungssystemversionen und den weiteren Elementen abgebildet werden können. Ist das vorkonfigurierte Metamodell nicht ausreichend, ist eine Anpassung (soweit möglich) vorzunehmen.
2. Die Informationen von SoCaStore sind aus Microsoft Excel zu importieren, ggf. ist eine Konvertierung der Daten nach XML oder CSV durchzuführen.
3. Das Editieren von Informationen im Werkzeug ist durch das Löschen, Neuanlegen und Ändern von Elementen zu prüfen.
4. Visualisierungen entsprechend der Abbildung 5.10 (Clusterkarte), Abbildung 5.11 (Prozessunterstützungskarte) und Abbildung 5.12 (Zeitintervallkarte) sollen erzeugt werden und von dem Werkzeug aufgrund der eingegebenen Informationen *generiert* werden. Anschließend sollen Veränderungen in der Visualisierung vorgenommen (Visualisierungselemente verschieben, Größe von Visualisierungselementen ändern) und die Darstellung erneut generiert werden, um festzustellen, ob die manuellen Änderungen erhalten bleiben.

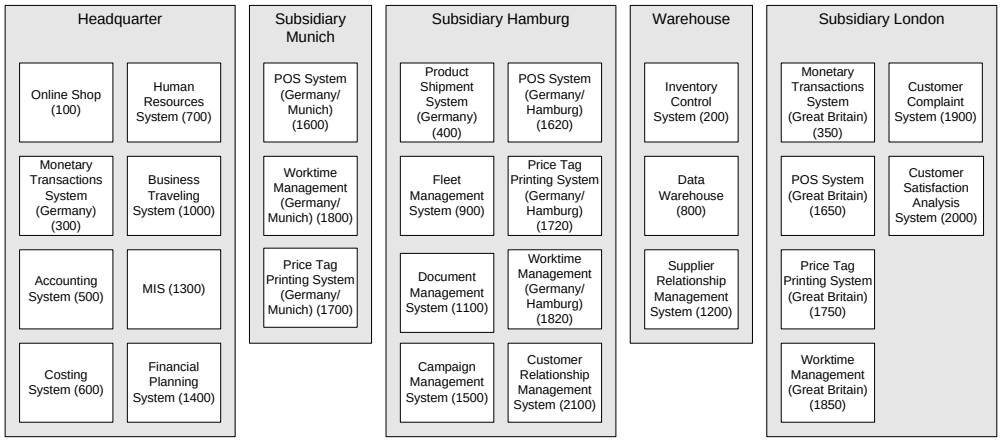


Abbildung 5.10.: EAMTS2005: Clusterkarte als Ergebnis des Szenarios *Visualisierung der Anwendungslandschaft* [se05a]

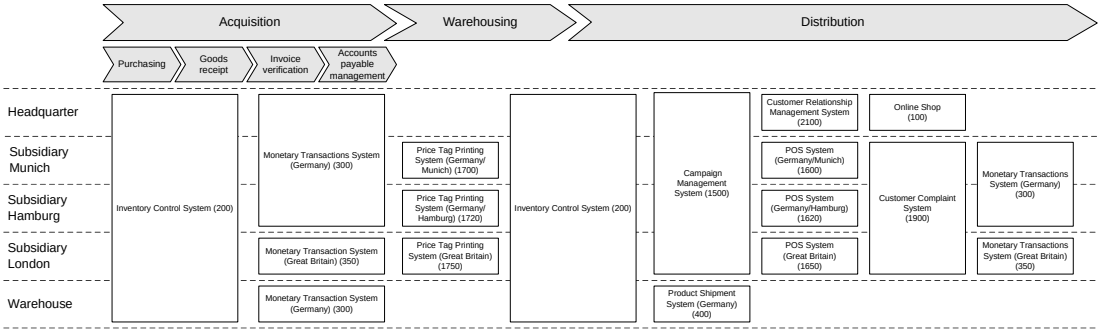


Abbildung 5.11.: EAMTS2005: Prozessunterstützungskarte als Ergebnis des Szenarios *Visualisierung der Anwendungslandschaft* [se05a]

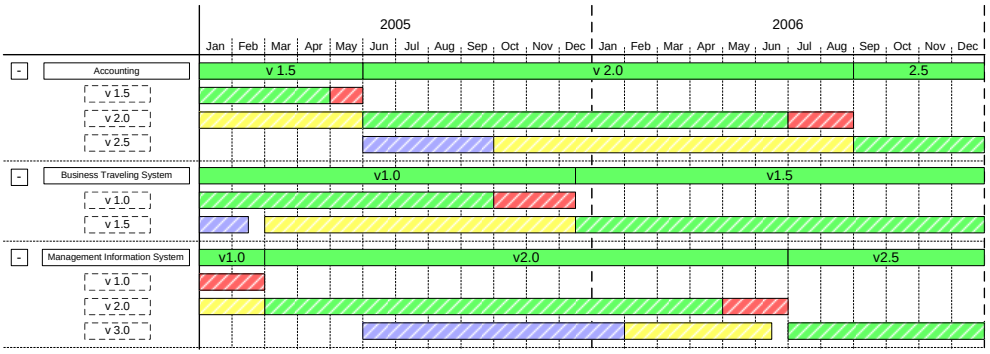


Abbildung 5.12.: EAMTS2005: Zeitintervallkarte als Ergebnis des Szenarios *Visualisierung der Anwendungslandschaft* [se05a]

5.2.1.2.2. Szenarios zur Analyse von Managementaufgaben

Die zweite Kategorie von Szenarios dient zur Analyse der Unterstützung spezifischer Managementprozesse durch die Werkzeuge. Die EAMTS2005 hat hierzu nicht nur das Management von Unternehmensarchitekturen analysiert, sondern auch andere Managementprozesse betrachtet (siehe auch Abschnitt 2.1).

Das Szenario *Management von Ist-, Plan- und Soll-Landschaften*[4] wird an dieser Stelle detailliert. Dies Szenario simuliert Aufgaben, die zum Management der Weiterentwicklung und Evolution der Anwendungslandschaft notwendig sind. Es besteht somit ein enger Bezug zum Projektportfoliomanagement, welches das Projektportfolio zusammenstellt und budgetiert. Die Interessen dieses Szenarios sind wie folgt formuliert:

> Um die Weiterentwicklung und Evolution der Anwendungslandschaft im Werkzeug abzubilden, muss das Werkzeug verschiedene Versionen und Szenarios[5] erzeugen sowie verwalten können.

Um die Interessen zu adressieren, müssen die bereits in Abschnitt 2.1.7 eingeführten Ist-, Plan- und Soll-Landschaften unterstützt werden. Die Fragestellungen, die während der Simulation dieses Szenarios beantwortet werden müssen, sind die folgenden:

- Unterstützt das Werkzeug die Visualisierung der Ist-Landschaft?
- Unterstützt das Werkzeug das Erzeugen, Visualisieren und Speichern von zukünftigen Versionen und Szenarios der Anwendungslandschaft (Plan- und Soll-Landschaften)?
- Unterstützt das Werkzeug das graphische Hervorheben von Unterschieden zwischen verschiedenen Versionen und Szenarios der Anwendungslandschaft?
- Unterstützt das Werkzeug das Verfolgen von Unterschieden zwischen den Anwendungslandschaften, die aus geplanten, laufenden oder abgeschlossenen Projekten entstehen bzw. entstanden sind?

Die Simulation des Szenarios erfolgt mittels der folgenden Schritte, welche die zu erreichenden Ergebnisse referenzieren:

1. Importiere oder gebe die Informationen zu Geschäftsprozessen, Anwendungssystemen, Organisationseinheiten und die Beziehungen zwischen diesen ein.
2. Importiere oder gebe die Informationen zu dem Projekt *Konsolidierung der Zahlungsverkehrssysteme* ein, welches das Anwendungssystem *Monetary Transaction System (Great Britain)* ablöst. Das Anwendungssystem *Monetary Transaction System (Germany)* soll diese Aufgabe übernehmen[6].

[4] In der EAMTS2005 [se05a] wurde dieses Szenario mit *Management der Anwendungslandschaft* bezeichnet.

[5] Szenario meint hier verschiedene Versionen einer Plan-Anwendungslandschaft und ist nicht mit den Szenarios der EAMTS2005 zu verwechseln.

[6] Diese Änderung kann durch das Vergleichen der Ist-Landschaft in Abbildung 5.13 und der Plan-Landschaft in Abbildung 5.14 nachvollzogen werden.

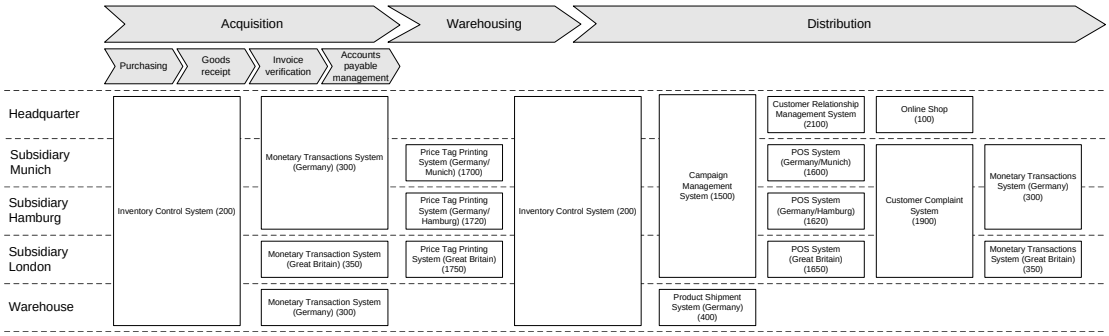


Abbildung 5.13.: EAMTS2005: Visualisierung der Ist-Landschaft als Ergebnis des Szenarios *Ist-, Plan- und Soll-Landschaften* [se05a]

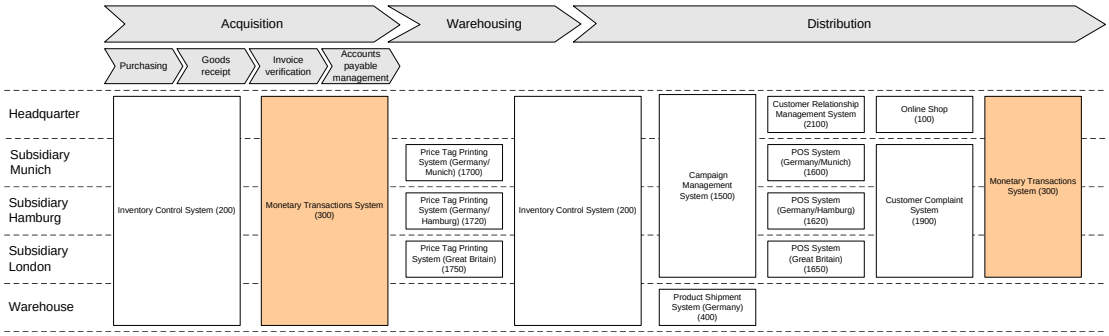


Abbildung 5.14.: EAMTS2005: Visualisierung der Plan-Landschaft als Ergebnis des Szenarios *Ist-, Plan- und Soll-Landschaften* [se05a]

3. Gebe die Informationen der Soll-Landschaft ein, welche die vertikale Integration der Anwendungslandschaft entlang der Organisationseinheiten erhöhen soll (siehe Abbildung 5.15). Hierzu sollen neben den Zahlungsverkehrssystemen ebenso die Preisetikettendrucksysteme (*Price Tag Printing Systems*) und die Kassensysteme (*POS*[7] *Systems*) homogenisiert werden.
4. Visualisiere die folgenden Anwendungslandschaften in einer graphischen Form:
   - Ist-Landschaft (2005-01-01), siehe Abbildung 5.13
   - Plan-Landschaft (2006-01-01) mit Hervorhebung der Unterschiede zur Ist-Landschaft, siehe Abbildung 5.14
   - Soll-Landschaft mit Hervorhebung der Unterschiede zur Ist-Landschaft, siehe Abbildung 5.15
5. Stelle die Unterschiede zwischen den Anwendungslandschaften tabellarisch dar, siehe Abbildung 5.16.

[7] POS - Point of Sale

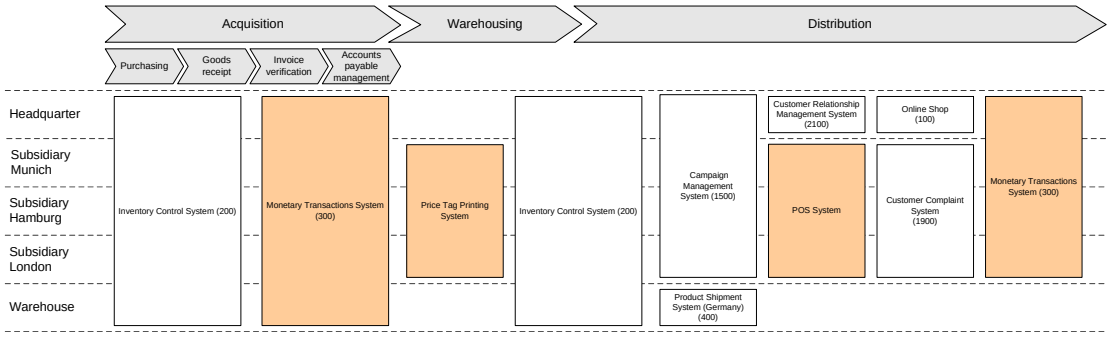


Abbildung 5.15.: EAMTS2005: Visualisierung der Soll-Landschaft als Ergebnis des Szenarios *Ist-, Plan- und Soll-Landschaften* [se05a]

Nr	Name	Current	Planned	Target
		2005-01-01	2006-01-01	
100	Online Shop			
200	Inventory Control System			
300	Monetary Transactions System (Germany)		x	x
350	Monetary Transactions System (Great Britain)		x	x
400	Product Shipment System (Germany)			
1500	Campaign Management System			
1600	POS System (Germany/Munich)			x
1620	POS System (Germany/Hamburg)			x
1650	POS System (Great Britain)			x
1700	Price Tag Printing System (Germany/Munich)			x
1720	Price Tag Printing System (Germany/Hamburg)			x
1750	Price Tag Printing System (Great Britain)			x
1900	Customer Complaint System			
2100	Customer Relationship Management System (CRM)			

x - changed

Abbildung 5.16.: EAMTS2005: Tabellarischer Bericht als Ergebnis des Szenarios *Ist-, Plan- und Soll-Landschaften* [se05a]

Um einen Überblick über die weiteren Szenarios in dieser Kategorie zu geben, werden diese kurz erklärt:

- Die Szenarios *Projektportfoliomanagement* (engl. *Project Portfolio Management*) und *Synchronisationsmanagement* (engl. *Synchronization Management*) simulieren Aufgaben, wie sie in den Abschnitten 2.1.3 und 2.1.6 beschrieben wurden. Ziele der Simulation sind u. a. Projekte mit Anwendungssystemen zueinander in Beziehung zu setzen, Abhängigkeiten von Projekten zu identifizieren und auf Änderungen in Projekten geeignet zu reagieren.
- Das Szenario *Nachvollziehbarkeit und Strategiemanagement* (engl. *Traceability and Strategy Management*) adressiert Probleme, die beim Verknüpfen von (IT-) Strategien mit anderen Elementen der Unternehmensarchitektur auftreten. Die Nachvollziehbarkeit von Zielen, die Strategien zugeordnet sind, welche in Projekten resultieren und ggf. neue Anwendungssystemversionen erzeugen, wird simuliert.
- Das Szenario *Management von Geschäftsobjekten und -services* (engl. *Management*

of Business Objects and Business Services) untersucht die Fähigkeiten der Werkzeuge im Umgang mit Geschäftsobjekten und -services. Neben dem Erfassen der Elemente sollen Informationsflüsse zwischen Anwendungssystemen, die über Services einen Zugriff auf Geschäftsobjekte ermöglichen, visualisiert werden.

- Das Szenario *Anwendungsarchitekturmanagement* (engl. *Application Architecture Management*) simuliert die Fähigkeiten der Werkzeuge im Umgang mit Musterarchitekturen und Musterlösungen, wie sie im Prozess IT-Architekturmanagement (siehe Abschnitt 2.1.5) verwendet werden.
- Das Szenario *Infrastrukturmanagement* (engl. *Infrastructure Management*) adressiert Problemstellungen, die beim Management des Lebenszyklus von Infrastrukturelementen auftreten, wie das Auslaufen der Support-Verträge für Datenbankmanagementsysteme etc.

5.2.1.2.3. Szenariosimulation

Zur Simulation der einzelnen Szenarios wurden Daten eines fiktives Kaufhauses *SoCaStore* genutzt, die aus Namen, Eigenschaften und Beziehungen von Geschäftsprozessen, Anwendungssystemen, Anwendungssystemversionen, Standorten, Organisationseinheiten etc. bestehen. Jedes Szenario wurde in der Analysephase der EAMTS2005 mit jedem Werkzeug simuliert. Bei der Simulation wurden die Ergebnisse protokolliert, um diese abschließend in den Ergebnisbericht [se05a] einfließen zu lassen.

Zur späteren Bewertung wurden bei den Szenarios zur Analyse von Managementaufgaben, die miteinander vernetzt sind, die folgenden Fakten der Simulation zusammengefasst:

Erreichen der Zielvorgabe (engl. *Achievement of Objectives*): Inwieweit wurden die durch die Ergebnisse der Szenariodokumentation gestellten Ziele erreicht?

Handhabbarkeit des Werkzeugs (engl. *Tool Handling*): War die Handhabbarkeit des Werkzeugs derart, dass die Simulation den Fähigkeiten des Werkzeugs entsprach und die Anwender bei der Simulation adäquat unterstützt werden?

Konsistenz mit dem Vorgehen (engl. *Procedure Consistency*): Entsprach das vorgegebene Metamodell und die vorgegebene Methode des Werkzeugs dem Vorgehen bei der Simulation oder mussten Metamodell und/oder Methode *verbogen* werden, um die Ergebnisse zu erreichen?

Integration beim Vorgehen (engl. *Procedure Integration*): War die Integration mit den vernetzten Szenarios gegeben, so dass existierende Ergebnisse und/oder Eingaben wiederverwendet werden können?

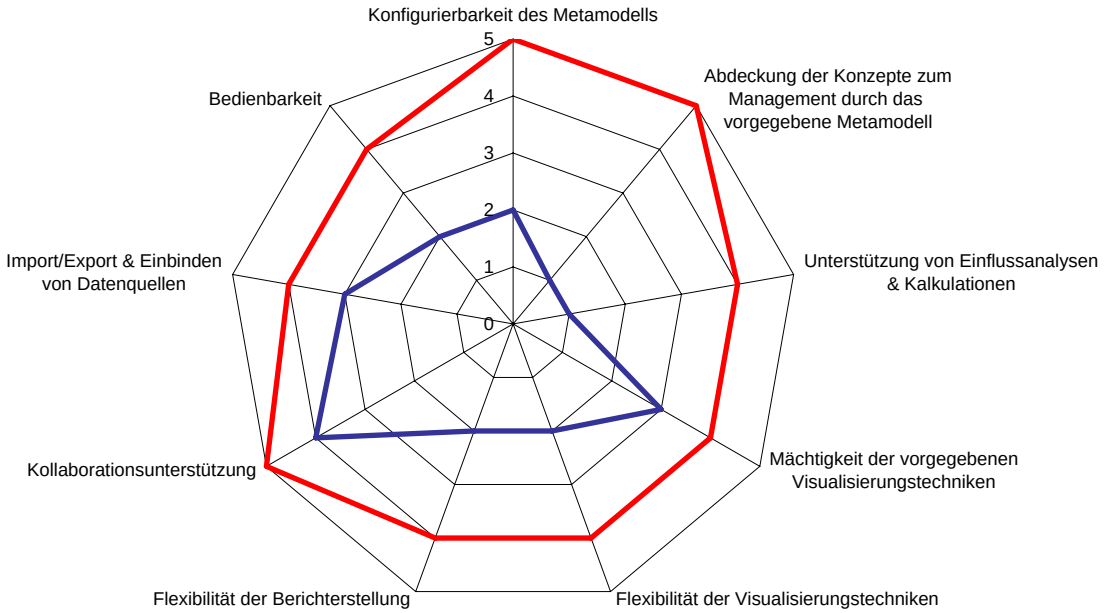


Abbildung 5.17.: EAMTS2005: Kiviat-Diagramm zur Bewertung spezifischer Funktionalitäten

5.2.2. Ergebnisse der Studie

Die Bewertung der Werkzeuge in der EAMTS2005 erfolgte anhand zweier Kiviat-Diagramme (siehe Abbildung 5.17 und Abbildung 5.18), die entlang der einzelnen Achsen verschiedene Aspekte zur Unterstützung des Managements von Unternehmensarchitekturen darstellen. Die Bewertung in den Achsen erfolgte mittels einer ordinalen Skalierung von eins (niedrige Fähigkeiten) bis fünf (hohe Fähigkeiten). Die roten Linien in den Kiviat-Diagrammen zeigen die maximale Bewertung und die blauen Linien zeigen die minimale Bewertung, die von den Werkzeugen erreicht wurden.

Das erste Kiviat-Diagramm (siehe Abbildung 5.17) visualisiert spezifische Fähigkeiten der Werkzeuge. Die Achsen sind wie folgt beschrieben:

Konfigurierbarkeit des Metamodells (engl. *Configurability of the Metamodel*) reflektiert die Möglichkeit, das (vorgegebene) Metamodell des Werkzeugs zu adaptieren.

Abdeckung der Konzepte zum Management durch das vorgegebene Metamodell (engl. *Coverage of Management Concepts by Predefined Metamodel*) reflektiert die Mächtigkeit des Metamodells, welches mit dem Werkzeug ausgeliefert wird.

Unterstützung von Einflussanalysen & Kalkulationen (engl. *Support for Impact Analysis & Calculation*) reflektiert die Fähigkeiten des Werkzeugs, Anfragen zur Bildung von Einflussanalysen und Kalkulationen anhand der Daten im Repository zu erstellen und dies zu visualisieren.

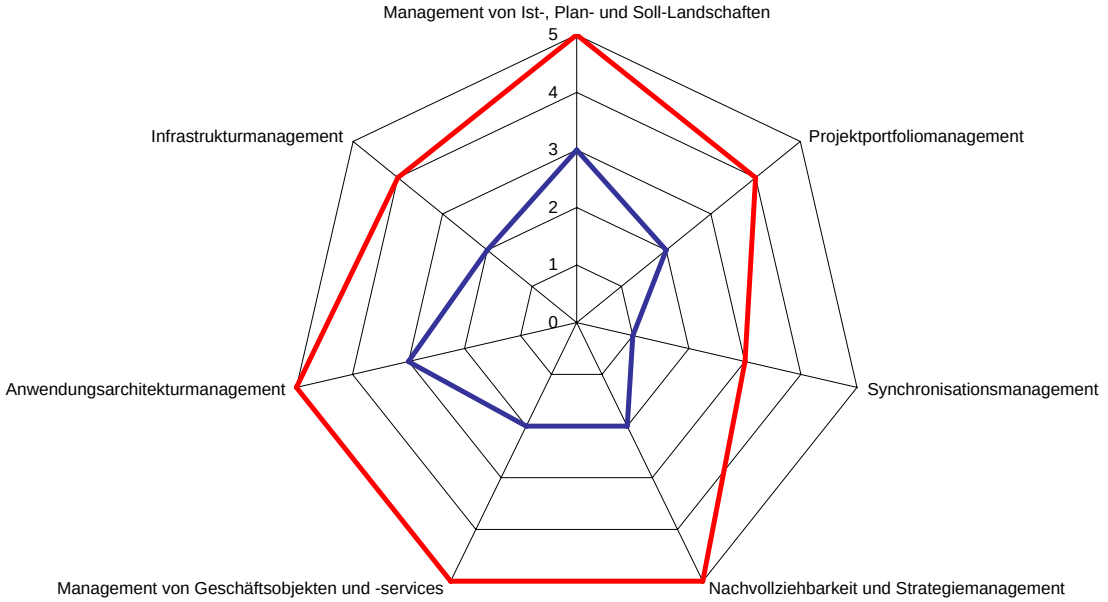


Abbildung 5.18.: EAMTS2005: Kiviat-Diagramm zur Bewertung von Managementaufgaben

Mächtigkeit der vorgegebenen Visualisierungstechniken (engl. *Richness of Predefined Visualization Techniques*) reflektiert die Fähigkeiten hinsichtlich der vom Werkzeug vorgegeben Visualisierungstechniken.

Flexibilität der Visualisierungstechniken (engl. *Flexibility of Visualization Techniques*) reflektiert zum einen die Fähigkeiten des Werkzeugs, neue Visualisierungsvorlagen zu erstellen, und zum anderen die Möglichkeiten Visualisierungen manuell anzupassen.

Flexibilität der Berichterstellung (engl. *Flexibility of Reporting*) reflektiert die Fähigkeiten des Werkzeugs, tabellarische Auswertungen auf Basis der Daten im Repository zu erstellen.

Kollaborationsunterstützung (engl. *Collaboration Support*) reflektiert die Fähigkeiten des Werkzeugs zur Unterstützung von parallelem Arbeiten durch mehrere Benutzer und welche Limitierungen hierbei existieren.

Import/Export & Einbinden von Datenquellen (engl. *Import/Export & External Data Sources*) reflektiert die Fähigkeiten des Werkzeugs, Daten in verschiedenen Formaten zu importieren und zu exportieren sowie externe Datenquellen einzubinden.

Bedienbarkeit (engl. *Usability*) bezieht sich auf die allgemeine subjektive Einschätzung der Benutzerfreundlichkeit.

Das zweite Kiviat-Diagramm (siehe Abbildung 5.18) visualisiert die Fähigkeiten der Werkzeuge für Managementaufgaben und besteht aus sieben Achsen. Die Achsen entsprechen den Szenarios zur Analyse von Managementaufgaben in Abschnitt 5.2.1.2.2.

Für jede der Achsen beider Diagramme wurden in dem *Executive Summary* der EAMTS2005 eine textuelle Beschreibung gegeben, die darstellt, welche Aspekte des Fragenkatalogs und der Szenarios in die Bewertung der Achse eingeflossen sind. Die Bewertung der einzelnen Achsen für jedes Werkzeug erfolgte nach der Erstellung der werkzeugabhängigen Abschnitte des Ergebnisberichts im Rahmen von Diskussionen und Workshops durch die Analyseteams.

5.2.3. Erkenntnisse der Studie

Die beiden Achsen *Konfigurierbarkeit des Metamodells* und *Abdeckung der Konzepte zum Management durch das vorgegebene Metamodell* sind von besonderer Bedeutung, da sie die Fähigkeiten des Metamodells und auch der Methode der Werkzeuge reflektieren. Während der Analyse der Werkzeuge ist aufgefallen, dass die Metamodelle sehr unterschiedlich sind. Dies bezieht sich sowohl auf die Größe (von ca. 30 Elemente bis zu ca. 400 Elemente) als auch auf Granularität, da sich die Abstraktionsgrade der Metamodelle unterscheiden. Eine Problematik von kleinen Metamodellen ist, dass diese ohne Adaption nicht den benötigten Funktionsumfang besitzen. Große Metamodelle sind hingegen mit hohem Aufwand an Einarbeitung verbunden und die Dokumentation des Metamodells, die oftmals unzureichend ist, bekommt ein größeres Gewicht.

Hinsichtlich der methodischen Unterstützung der Werkzeuge, die durch die Achse *Abdeckung der Konzepte zum Management durch das vorgegebene Metamodell* reflektiert wird, existieren deutliche Unterschiede zwischen den Werkzeugen. Einige Werkzeuge verfügen über keine Dokumentation der Methode, weder im Werkzeug noch in externen Dokumenten. Dies bedeutet für den Anwender, dass eine eigene Methode zu entwickeln ist.

Des Weiteren kann zwischen rein methodengetriebenen und prozessgetriebenen Werkzeugen unterschieden werden. Die methodengetriebenen Werkzeuge unterstützen dabei die Adressierung einzelner Interessen durch Modelle (Was?) und eine methodische Dokumentation der Schritte (Wie?). Es fehlt diesen jedoch an einer Beschreibung, wie die einzelnen Modelle zusammenhängen und wann welches Modell angewendet wird (Wann?). Die prozessgetriebenen Werkzeuge beinhalten dies und ergänzen die einzelnen (*kleinen*) Methoden und Modelle mit einem Gesamtbild.

Die Achse *Kollaborationsunterstützung* zeigt, dass dieser Aspekt von allen Werkzeugen verstanden und adäquat umgesetzt wurde. Alle Werkzeuge unterstützen das parallele Arbeiten mehrerer Benutzer und verfügen über geeignete Rollen- und Berechtigungskonzepte. Die Differenz zwischen vier und fünf auf der Bewertungsskala begründet sich insbesondere durch die Webfähigkeit der einzelnen Werkzeuge. Einige Werkzeuge verfügen nur über einen statischen HTML-Export, ohne eine Weboberfläche zur Verfügung zu stellen, die ebenso das Editieren von Elementen erlaubt.

Auffallend war die Achse *Unterstützung von Einflussanalysen und Kalkulationen*, da die Werkzeuge über keine Aggregatfunktionen, wie sie beispielsweise in SQL oder OQL vorhanden sind, verfügen. Diese Fähigkeiten wurden während der Simulation vermisst, da es beispielsweise nicht möglich war, die Anzahl von Geschäftsprozessunterstützungen zu *zählen*, wie es in SQL über den Befehl `COUNT` ermöglicht wird.

Eine weitere Erkenntnis aus den spezifischen Funktionalitäten ist, dass Verbesserungspotentiale für die Visualisierungen existieren. Die Konzepte zum Verschachteln und Ausrichten, wie sie die Gestaltungsregeln in Abschnitt 4.3 beschreiben, fehlen zahlreichen Werkzeugen. Eine semantisch korrekte Darstellung von Softwarekarten ist in diesen Werkzeugen nicht möglich. Bei der Flexibilität der Visualisierungen fehlt es vielen Werkzeugen an Möglichkeiten, zu filtern und Kennzahlen darzustellen, wie es in Abschnitt 4.3.1 beschrieben wurde.

Für eine maximale Bewertung bei der Achse *Import/Export & Einbinden von Datenquellen* fehlt bisher ein standardisiertes Austauschformat. XML als Basis für Import/-Export erleichtert zwar die (maschinelle) Lesbarkeit der Daten, jedoch gibt es kein Austauschformat wie beispielsweise UN/EDIFACT[8]. Dadurch wird das Einbinden von externen Datenquellen (bspw. Systemsmanagementsysteme, Projektportfoliomanagementsysteme) erschwert.

Im zweiten Kiviat-Diagramm sind mehrere Aspekte aufgefallen. Die Szenarios *Projektportfoliomanagement* und *Synchronisationsmanagement* benötigen beide das Konzept *Projekt*, welches aus Aktivitäten besteht sowie definierte Start- und Enddaten besitzt. Insbesondere das Konzept *Datum* bzw. *Zeit* wird von den meisten Werkzeugen nicht unterstützt, so dass Abfragen mit einem Zeitbezug nicht möglich sind.

Auffällig war im zweiten Kiviat-Diagramm die Achse *Management von Geschäftsobjekten und -services*, da die Simulation dieses Szenarios sehr unterschiedlich unterstützt und umgesetzt wurde. Die Modellierung von Geschäftsobjekten, auf die über Services zugegriffen werden kann, und den Informationsflüssen zwischen Anwendungssystemen wird von einigen Werkzeugen unzureichend unterstützt.

Abschließend ist festzuhalten, dass die von den Herstellern mitgelieferten Dokumentationen zu Metamodell und Methode nicht durchweg ausreichend sind. Einige Hersteller verfügen zwar über komplexe Metamodelle, aber ohne eine entsprechende Dokumentation kann dieses Metamodell nur unter Rückgriff auf das interne Know-how des Herstellers korrekt verwendet und auch adaptiert werden.

Der Aufbau eines Musterkatalogs, bestehend aus Mustern wie in Abschnitt 4.2.3 vorgestellt, kann hier Abhilfe schaffen. Die Hersteller können einzelne Muster referenzieren, die bereits eine Dokumentation besitzen. Die *Enterprise Architecture Management Viewpoint Survey*, die derzeit vom Lehrstuhl für Informatik 19 (sebis) der TU München durchgeführt wird, hat das Ziel, einen derartigen Musterkatalog aufzubauen.

Die Ergebnisse der EAMTS2005 und ihre Folgerungen wurden in zahlreichen Publikationen aufgegriffen (u. a. [ELW05], [ELW06], [Ke06], [De06], [Bu07]) und auf Konferenzen vorgestellt (u. a. EDOC'2006, Computerwoche Executive Program, EUROFORUM).

Im Nachgang der EAMTS2005 haben einige Werkzeughersteller reagiert. Das Werkzeug *planningIT* verfügt inzwischen über eine Anfragesprache, die Aggregatfunktionen unterstützt, und hat die Visualisierungstechniken erweitert, so dass inzwischen Kennzahlen mit Farben, Ampeln etc. visualisiert werden. Die ARIS Design Plattform, zu der das *ARIS*

[8] UN/EDIFACT - United Nations / Electronic Data Interchange for Administration, Commerce and Transport

Toolset zählt, wurde um ein eigenständiges Produkt *ARIS IT Architect* [Ma06b] erweitert, das u. a. Prozessunterstützungskarten und Zeitintervallkarten visualisieren kann sowie eine Unterstützung für das Verschachteln von Gestaltungsmitteln für Clusterkarten kennt (bei ARIS *Objekt-in-Objekt-Funktionalität* genannt).

Eine Neuauflage der EAMTS2005 wird derzeit erarbeitet. Die Publikation des Ergebnisberichts der *Enterprise Architecture Management Tool Survey 2008* ist für das erste Quartal 2008 geplant.

5.3. Design eines Werkzeugs für die Softwarekartographie

Um die Implementierbarkeit von Softwarekarten in einem Modellierungswerkzeug prototypisch zu zeigen, wird am Lehrstuhl für Informatik 19 (sebis) der TU München das *Software Cartography Tool* (kurz SoCaTool) entwickelt. Im Gegensatz zu den Werkzeugen, die in der EAMTS2005 (siehe Abschnitt 5.2) analysiert wurden, fokussiert das SoCaTool ausschließlich auf Anwendungsfälle beim Einsatz von Softwarekarten. Das Werkzeug besitzt somit keine Fähigkeiten zur Erstellung von tabellarischen Auswertungen etc.

Ziel dieses Abschnittes ist es, anhand des SoCaTools die Implementierbarkeit des Visualisierungsmodells aus Abschnitt 4.3 zu zeigen. Hierzu beschreibt Abschnitt 5.3.1 die Anforderungen und Softwarearchitektur des Werkzeugs, und Abschnitt 5.3.2 stellt die graphische Benutzungsschnittstelle vor.

5.3.1. Anforderungen und Softwarearchitektur SoCaTool

Die wesentlichen Anforderungen, die zu der gewählten Softwarearchitektur geführt haben, sind:

Abdeckung aller Softwarekartentypen: Alle Softwarekartentypen (siehe Abschnitt 3.4.1) müssen vom SoCaTool unterstützt werden.

Unterstützung des Schichtenprinzips: Das Schichtenprinzip von Softwarekarten (siehe Abschnitt 3.4.2) muss vom SoCaTool unterstützt werden, um verschiedene Merkmale einer Anwendungslandschaft auf verschiedenen Schichten zu visualisieren.

Flexibilität beim Informationsmodell: Das SoCaTool muss flexibel hinsichtlich des Informationsmodells zum Management von Anwendungslandschaften sein, um nicht ausschließlich auf *einem* Informationsmodell zu basieren. Die Vielzahl von Informationsmodellen bei Projektpartnern, in den verschiedenen Werkzeugen zum Management von Unternehmensarchitekturen etc. (siehe Abschnitte 4.2 und 5.2), verlangt das Vorsehen geeigneter Schnittstellen, um verschiedene Informationsmodelle und konforme Instantiierungen (semantische Modelle) zu importieren.

(Semi-) Automatische Generierung von Softwarekarten: Softwarekarten werden (semi-) automatisch vom SoCaTool generiert. Eine Anforderung ist, dass eine erste Version einer Softwarekarte automatisch aus dem semantischen Modell generiert werden muss. Die Softwarekarte muss hierbei korrekt hinsichtlich der Semantik sein und

somit alle Muss-Gestaltungsregeln erfüllen. Die *ästhetischen* Aspekte müssen nicht vollständig berücksichtigt werden. Die Ziel-Gestaltungsregeln des Visualisierungsmodells versuchen, diese ästhetischen Aspekte zu unterstützen, können jedoch nicht immer das Optimum für den Nutzer repräsentieren.

Editieren von Softwarekarten: Softwarekarten können graphisch editiert und die Eigenschaften einzelner Informationsobjekte können verändert werden. Das Editieren kann Änderungen des semantischen Modells bedingen.

Flexibel beim Exportformat: Als Exportformate für Softwarekarten sind webfähige Formate wie JPG, PNG oder SVG und ebenso PDF vorzusehen.

Anpassungsfähigkeit der Informationsdarstellung: Die Art, wie bestimmte Informationen auf einer Softwarekarte angezeigt werden, muss flexibel gestaltbar sein. Beispielsweise soll das Visualisieren der Konformität eines Anwendungssystems mit einer Musterarchitektur (siehe Abschnitt 4.2.3.2) sowohl mittels der Hintergrundfarbe als auch durch Ampeln (siehe Abschnitt 4.3.1) ermöglicht werden.

Die obigen Anforderungen wurden von Schweda [Sc06] in seiner Diplomarbeit in einer Anforderungsanalyse mit Anwendungsfällen aufgegriffen und dokumentiert.

Um insbesondere die Problematiken der *Flexibilität des Informationsmodells* und der *Anpassungsfähigkeit der Informationsdarstellung* zu adressieren, basiert das Werkzeug auf der in Abschnitt 4.4 vorgestellten Modelltransformation, welche aus semantischen Modellen symbolische Modelle erzeugt. Die symbolischen Modelle sind hierbei Instanzen des in Abschnitt 4.3 vorgestellten Visualisierungsmodells.

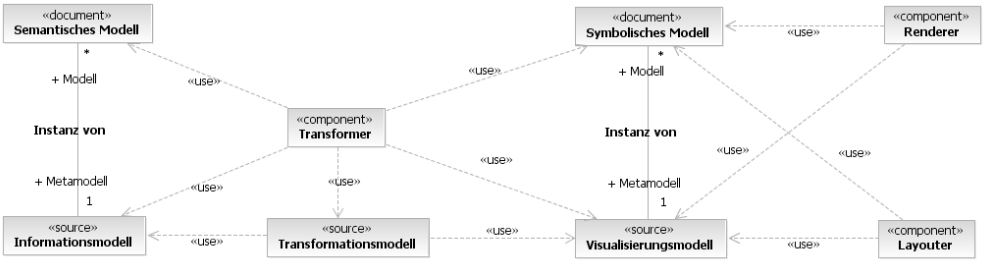


Abbildung 5.19.: Softwarearchitektur des SoCaTools

Auf Basis obiger Anforderungen und der Anforderungsanalyse von Schweda [Sc06] wurde eine Softwarearchitektur bestehend aus den folgenden Elementen (siehe Abbildung 5.19) entwickelt:

Informationsmodell: Das *Informationsmodell* ist das Metamodell eines semantischen Modells, wie in Abschnitt 4.1 eingeführt und in Abschnitt 4.2 diskutiert. Die Elemente des Informationsmodells werden im SoCaTool als Subklassen von `ENamedElement` des Ecore-Modells des *Eclipse Modeling Frameworks* (EMF) [Bu04] repräsentiert.

Semantisches Modell: Das *semantische Modell* ist die Menge von Informationsobjekten, wie in Abschnitt 4.1 eingeführt. Die Informationsobjekte enthalten die semantischen Informationen, die in einer Softwarekarte dargestellt werden. Im SoCaTool werden Informationsobjekte als `EObject`s des Ecore-Modells repräsentiert.

Visualisierungsmodell: Das *Visualisierungsmodell* ist das Metamodell eines symbolischen Modells, wie in Abschnitt 4.1 grundlegend eingeführt und in Abschnitt 4.3 detailliert beschrieben. Die Elemente des Visualisierungsmodells werden im SoCaTool als Subklassen von `ENamedElement` des Ecore-Modells repräsentiert.

Symbolisches Modell: Das *symbolische Modell* ist die Menge von Visualisierungselementen, wie in Abschnitt 4.1 eingeführt, ergänzt um die Metainformationen einer Softwarekarte, wie Titel, Legende, Autor etc. Die Visualisierungselemente sind die Bestandteile eines Diagramms einer Softwarekarte. Im SoCaTool werden Visualisierungselemente als `EObject`s des EMF repräsentiert.

Transformationsmodell: Das *Transformationsmodell* ist eine Menge von Regeln, welche Quellmodelle in Zielmodelle umsetzt. Diese Regeln beschreiben Transformationen von Informationsobjekten in Visualisierungselemente oder andersherum. Im SoCaTool sind diese Regeln in Java realisiert und nutzen die Java-API des EMF, um auf die Elemente des Informationsmodells, des semantischen Modells, des Visualisierungsmodells und des symbolischen Modells zuzugreifen.

Transformer: Der *Transformer* übersetzt auf Basis ausgewählter Regeln des Transformationsmodells ein Quellmodell in ein Zielmodell. Im SoCaTool ruft der Transformer die entsprechenden Java-Klassen mit den Regeln für das Transformationsmodell auf.

Layouter: Der *Layouter* verändert ein symbolisches Modell derart, dass die Gestaltungsregelinstanzen in Koordinaten der Gestaltungsmittelinstanzen übersetzt werden. Um diese Koordinaten zu berechnen, werden vom Layouter verschiedene Subkomponenten, die unterschiedliche Algorithmen implementieren, genutzt.

Renderer: Der *Renderer* stellt ein symbolisches Modell auf einem gewählten Ausgabemedium oder für ein Exportformat dar.

Das Transformationsmodell ist derzeit nativ in Java implementiert, da bisher kein Metamodell zur Spezifikation dieser Regeln existiert, welches die Anforderungen an eine Transformation erfüllt. Derzeit werden hierzu am Lehrstuhl für Informatik 19 (sebis) weitere Forschungsvorhaben initiiert, um die Möglichkeiten von *Query/View/Transformation* (QVT) [OM05a] zu evaluieren. Die Untersuchungen von ATL [AT06] und BOTL [BM03] haben ergeben, dass diese derzeit nicht den Anforderungen entsprechen. Beispielsweise besitzt BOTL ein eigenes Metamodell, welches unverträglich mit EMOF oder Ecore ist. ATL verfügt nicht über parametrisierbare Regeln, um in Abhängigkeit von Eigenschaften eines Informationsobjekts Regeln auszuführen.

Zur Berechnung der Koordinaten von flächenartigen Gestaltungsmittelinstanzen (`PlanarMapSymbol`, siehe Abschnitt 4.3.2) stehen dem *Layouter* derzeit zwei Komponenten zur Verfügung. Die erste Komponente löst ein Optimierungsproblem, welches zuvor vom Layouter aus dem symbolischen Modell abgeleitet wurde (siehe Abschnitt 4.4), mittels

eines genetischen Algorithmus. Die zweite Komponente verwendet so genannte Packing-Algorithmen, um mittels einer Heuristik die Positionen zu berechnen (siehe Lauschke [La07]). Zur Weiterentwicklung der Komponente, welche die Koordinaten ermittelt, indem ein Optimierungsproblem gelöst wird, sind weitere Forschungsvorhaben am Lehrstuhl für Informatik 19 (sebis) geplant, da diese Komponente derzeit ineffizient ist und teils ein ineffektives Verhalten besitzt.

Linienartige Gestaltungsmittelinstanzen (`LinearMapSymbol`, siehe Abschnitt 4.3.2) werden eigenständig in einem nachgeschalteten Schritt berechnet, der nach dem Layouting der flächenartigen Gestaltungsmittelinstanzen erfolgt. Grund hierfür ist, dass die Komplexität des Optimierungsproblems durch das Minimieren von Überkreuzungen von Linien zusätzlich steigen würde und der Packing-Algorithmus nicht auf linienartigen Gestaltungsmittelninstanzen arbeiten kann. Der für das Layouting der linienartigen Gestaltungsmittelinstanzen verwendete Algorithmus wird im Chip-Design verwendet und ermöglicht es, bei feststehender Positionierung der flächenartigen Gestaltungsmittel ein Layout der linienartigen Gestaltungsmittelinstanzen zu erhalten, welches minimale Linienkreuzungen besitzt.

Die Außensicht des SoCaTools entsteht durch die exportierten Schnittstellen, welche Informationsmodelle und konforme semantische Modelle importieren sowie Softwarekarten in unterschiedlichen Formaten exportieren. Da das SoCaTool in die *Eclipse Rich Client Plattform* eingebunden ist, werden diese Schnittstellen durch die Schnittstelle `Resource` des Eclipse Frameworks repräsentiert. Die exportierende Schnittstelle wird gleichfalls über `Resource` realisiert und exportiert Softwarekarten als XML-Serialisierung des symbolischen Modells, PDF, PNG und SVG. Für die Testumgebung wird das Informationsmodell als Ecore-konforme XML-Serialisierung eingelesen und das semantische Modell aus einer Datei im Format SpreadsheetML[9] importiert. Das Visualisierungsmodell, welches ebenso im Ecore-Format vorliegt, wird über eine (interne) Schnittstelle des SoCaTools eingelesen. Eine Beschreibung der Integration in die *Eclipse Richt Client Plattform* findet sich bei Schweda [Sc06] und in der Dokumentation des Werkzeugs.

Derzeit fehlt es dem SoCaTool an einem *InverseLayouter*, der Änderungen an der Visualisierung in das symbolische Modell propagiert und gegebenenfalls durch eine inverse Modelltransformation die Änderungen in das semantische Modell überführen kann. Um die verbundene Anforderung *Editieren von Softwarekarten* zu unterstützen, wird neben der Komponente *Layouter* eine Komponente *InverseLayouter* entstehen. Offen ist des Weiteren die bereits diskutierte Sprache zur Beschreibung der Transformationsregeln.

5.3.2. Graphische Benutzungsschnittstelle des SoCaTools

Die graphische Benutzungsschnittstelle des SoCaTools ist, wie auch die zuvor beschriebenen Komponenten, in die *Eclipse Rich Client Plattform* eingebettet. Die Benutzungsschnittelle besitzt hierbei ein ähnliches Aussehen wie die Entwicklungsumgebung *Eclipse* und das Modellierungswerkzeug *IBM Rational Software Architect*.

[9] SpreadsheetML ist ein von Microsoft Excel unterstütztes XML-Format.

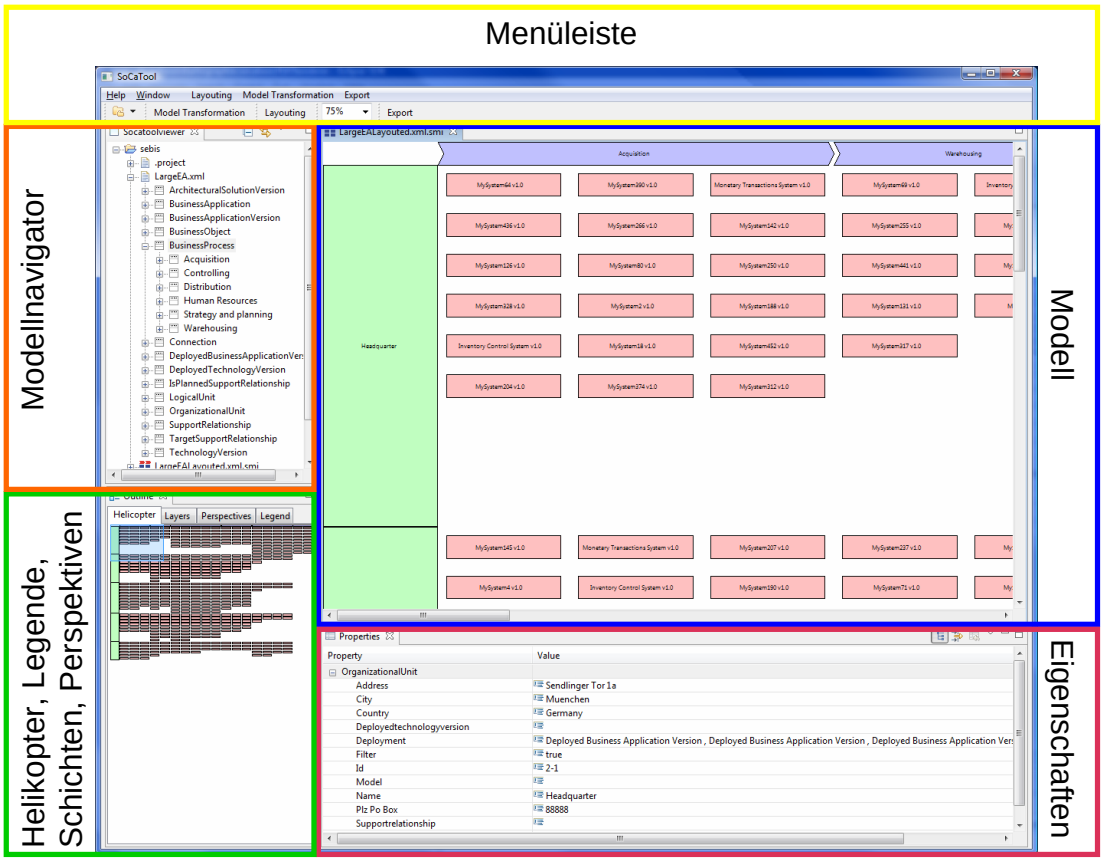


Abbildung 5.20.: Graphische Benutzungsschnittstelle des SoCaTools

Ein Screenshot der graphischen Benutzungsschnittstelle des SoCaTools ist Abbildung 5.20 zu entnehmen, wobei die einzelnen Elemente zusätzlich mit Namen gekennzeichnet wurden. Diese Elemente besitzen die folgende Bedeutungen und Funktionen:

Modell: Das Modell-Fenster stellt das Diagramm einer Softwarekarte dar. Der Benutzer kann einzelne Gestaltungsmittelinstanzen auswählen, so dass im Eigenschaften-Fenster die Attribute und Werte der ausgewählten Gestaltungsmittelinstanz angezeigt werden. Des Weiteren kann der Benutzer den Ausschnitt des Diagramms durch hinein- und herauszoomen verändern. In der derzeitigen Version des SoCaTools ist kein Editieren in diesem Fenster möglich.

Eigenschaften: Das Eigenschaften-Fenster zeigt die Attribute und Werte zu einer im Modell-Fenster oder Informationsmodell-Fenster ausgewählten Gestaltungsmittelinstanz an. Wird der Hintergrund gewählt, so erhält der Benutzer Auskunft über den Titel der Softwarekarte, den Ersteller etc. Das Verändern von Attributwerten einer Gestaltungsmittelinstanz wird derzeit nicht unterstützt.

Helikopter: Das Helikopter-Fenster ermöglicht das Navigieren in großen Softwarekarten, die im vollen Umfang nicht leserlich im Modell-Fenster angezeigt werden können.

Der Benutzer kann hierzu das Rechteck, welches dem im Modell-Fenster angezeigten Ausschnitt entspricht, verschieben.

Legende: Das Legenden-Fenster zeigt die Legende einer Softwarekarte.

Schichten: Das Schichten-Fenster zeigt alle verfügbaren Schichten der Softwarekarten und ihren Status (ein- oder ausgeblendet). Der Benutzer kann einzelne Schichten ein- und ausblenden. Zusätzlich kann die Reihenfolge der Schichten verändert werden, welches die z-Position einer Gestaltungsmittelinstanz verändert.

Perspektiven: Das Perspektiven-Fenster zeigt alle Perspektiven der Softwarekarte und welche Perspektive ausgewählt ist. Der Benutzer kann eine andere Perspektive wählen, wodurch die zugehörigen Schichten eingeblendet und alle nicht assoziierten Schichten ausgeblendet werden. Ist keine Perspektive gewählt, so wird die Perspektive *NULL* als aktiv markiert. Zusätzlich kann der Benutzer eine neue Perspektive erstellen, der automatisch die derzeit aktivierten Schichten zugewiesen werden.

Modellnavigator: Das Modellnavigator-Fenster zeigt zu einem gewählten semantischen oder symbolischen Modell die zugehörigen Elemente in einer Baumdarstellung an. Der Benutzer kann in einem gewählten Modell hierarchisch navigieren und erhält bei Auswahl eines Elements die zugehörigen Eigenschaften im Eigenschaften-Fenster angezeigt.

Menüleiste: Die Menüleiste enthält Funktionen zum

- Öffnen eines Projektes,
- Editieren der Projekteinstellungen (Ort des Informationsmodells und des Visualisierungsmodells) und
- Speichern von semantischen und symbolischen Modellen.

Des Weiteren sind in der Menüleiste die Druckknöpfe zum Anstoßen der Modelltransformation, der unterschiedlichen Layouting-Algorithmen und zum Exportieren enthalten.

Nach dem Erstellen eines Projektes und dessen Konfiguration ist folgende Benutzerinteraktion auszuführen, um eine Softwarekarte zu generieren:

1. Auswählen des semantischen Modells in dem Modellnavigator.
2. Transformieren des semantischen Modells in ein (nicht gelayoutetes) symbolisches Modell durch Klicken des Tranformationsknopfes. In einem Dialogfenster wählt der Benutzer die gewünschte Transformation aus, um beispielsweise eine Prozessunterstützungskarte zu generieren.
3. Layouten des symbolischen Modells durch Klicken des Knopfes in der Menüleiste. In dem Fall der Prozessunterstützungskarte wird der Packing-Algorithmus (siehe Lauschke [La07]) genutzt. Wird eine Clusterkarte mit Darstellung von Kommunikationsbeziehungen gewählt, so wird in einem ersten Schritt der Packing-Algorithmus gewählt und in einem zweiten Schritt das Linien-Layouting erfolgen.

Diese Interaktion zeigt die Umsetzung der Modelltransformation und des Layoutings, um Softwarekarten zu generieren. Das Ecore-Modell ist sowohl das Metamodell des Informationsmodells als auch des Visualisierungsmodells, so dass für die Transformation ein einheitliches Metamodell vorhanden ist (siehe Abschnitt 4.4). Das *Eclipse Modeling Frameworks* implementiert das Ecore-Modell und ermöglicht das Schreiben von Transformationen in Java.

Das SoCaTool ist eine prototypische Implementierung eines Werkzeugs für die Softwarekartographie. Seitens der Hersteller von Werkzeugen zum Management der Unternehmensarchitekturen (siehe Abschnitt 5.2) besteht insbesondere Interesse an den Layouting-Algorithmen, um diese in ihren Werkzeugen umzusetzen. Am Lehrstuhl für Informatik 19 (sebis) wird das Werkzeug weiterentwickelt, um beispielsweise auch zur Berechnung von Metriken für Anwendungslandschaften eingesetzt zu werden.

5.4. Bewertung

Wie Softwarekarten angewendet werden, war Gegenstand dieses Kapitels. Abschnitt 5.1 hat mit den Konzepten der vertikalen und horizontalen Integration sowie dem Fassadenprinzip aufgezeigt, wie Anwendungslandschaften gestaltet und welche Ziele hierbei verfolgt werden können. Die vertikale und horizontale Integration fokussiert dabei auf einer Homogenisierung der Anwendungslandschaft und soll Skaleneffekte erzielen und die Effizienz der Anwendungslandschaft verbessern. Das Fassadenprinzip stellt eine Möglichkeit dar, die Komplexität der Kommunikationsbeziehungen in Anwendungslandschaften zu vereinfachen, indem die Kommunikation zwischen den Domänen der Anwendungslandschaft vereinheitlicht wird. Die abschließende Diskussion zur optimalen Anwendungslandschaft in Abschnitt 5.1 hat den Zusammenhang von Effektivität und Effizienz für Anwendungslandschaften aufgezeigt und trotz des Fazits, dass es keine *optimale* Anwendungslandschaft gibt, dargelegt, dass die Zielsetzung beim Management der Anwendungslandschaft die Flexibilität ist. Diese Flexibilität in der Anwendungslandschaft hat das Ziel, den steten Wandel von Anwendungslandschaften geeignet zu unterstützen.

Welche Werkzeuge zum Management von Anwendungslandschaften und Unternehmensarchitekturen eingesetzt werden können, wurde in Abschnitt 5.2 entlang der *Enterprise Architecture Management Tool Survey 2005* [se05a] gezeigt. Es wurde hervorgehoben, dass diese Werkzeuge unterschiedliche Stärken und Schwächen besitzen und die Ansätze zum Management von Anwendungslandschaften und Unternehmensarchitekturen deutliche Unterschiede aufweisen. Die Spanne der Werkzeuge reicht von *schlanken* Metamodellierungslösungen mit kleinen Metamodellen (ca. 30 Elemente), über Werkzeuge mit großen Metamodellen (ca. 300 Elemente) bis hin zu Werkzeugen mit methodischem Charakter, die teilweise auch einen komplexen Managementprozess implementieren. Bei der Unterstützung der Softwarekartographie, wie in dieser Arbeit vorgestellt, besteht bei allen Werkzeugen Verbesserungsbedarf. Positiv hervorzuheben ist, dass beispielsweise die Hersteller alfabet AG und IDS Scheer AG Kritikpunkte aus der Studie aufgegriffen und ihre Werkzeuge mit neuen Funktionalitäten ausgestattet haben.

Wie ein Werkzeug für die Softwarekartographie aussehen und wie die Erstellung und Pflege von Softwarekarten softwaretechnisch unterstützt werden kann, wurde am Beispiel des *Software Cartography Tools* (kurz SoCaTool) in Abschnitt 5.3 gezeigt. Die Implementierung des zuvor in Abschnitt 4.3 entwickelten Visualisierungsmodells und die Anwendung der Modelltransformation aus Abschnitt 4.4 im SoCaTool haben gezeigt, wie eine Modellierungssprache mit Softwarekarten in einem Werkzeug prototypisch umgesetzt wird. Die durch die Modelltransformation und die Wahl geeigneter Metamodelle gewahrte Flexibilität beim Informationsmodell erlaubt es dem SoCaTool, auf neue Anforderungen hinsichtlich der darzustellenden Informationen zu reagieren.

KAPITEL 6

Zusammenfassung und Ausblick

Inhaltsverzeichnis

In diesem Kapitel werden die Erkenntnisse der Arbeit in Abschnitt 6.1 bewertend zusammengefasst. Abschnitt 6.2 gibt einen Ausblick auf zukünftige Arbeiten, die auf den Ergebnissen dieser Arbeit aufbauen können.

6.1. Zusammenfassung

In dieser Arbeit wurde eine graphische Modellierungssprache für Anwendungslandschaften entwickelt und ihre Verwendung im Kontext des Managements von Anwendungslandschaften erläutert.

Es wurden sechs Prozesse für das Management von Anwendungslandschaften identifiziert und die Interaktion dieser Prozess untereinander sowie mit dem IT-Projektlebenszyklus dargestellt (siehe Kapitel 2). Die einzelnen Prozesse umfassen das *Anforderungsmanagement*, das *Strategie- & Zielemanagement*, das *Projektportfoliomanagement*, das *Synchronisationsmanagement*, das *IT-Architekturmanagement* und das *Management der Unternehmensarchitektur*.

Für das Management der Anwendungslandschaft bildet der Prozess *Management der Unternehmensarchitektur* den *Klebstoff* zwischen den einzelnen Prozessen, indem die

Modelle dieses Prozesses von den anderen Prozessen genutzt und im Gegenzug mit Informationen ergänzt werden. Auf diesen graphischen Modellen, die als Softwarekarten bezeichnet werden und Anwendungslandschaften graphisch darstellen, lag der Fokus der weiteren Arbeit.

Es wurde eine graphische Modellierungssprache mit Softwarekarten, bestehend aus abstrakter Syntax, konkreter Syntax, Serialisierungssyntax und Semantik, aufgebaut und unter Zuhilfenahme der thematischen Kartographie ein Begriffsapparat für das Zeichensystem der Softwarekartographie definiert (siehe Kapitel 3 und 4).

Die Unterscheidung von *semantischen* und *symbolischen Modellen* in der Softwarekartographie mit den Metamodellen *Informationsmodell* und *Visualisierungsmodell* ermöglicht eine korrekte Definition der graphischen Visualisierungen, so dass Defizite anderer Modellierungssprachen vermieden werden.

Existierende Informationsmodelle im Kontext des Managements von Anwendungslandschaften und Unternehmensarchitekturen wurden betrachtet, die jedoch den Anforderungen der Softwarekartographie und den Managementprozessen nicht genügen. Diese Modelle decken nicht alle Interessen der Stakeholder für das Management von Anwendungslandschaften ab. Um die Konstruktion von Informationsmodellen zu verbessern, wurden Informationsmodellmuster, als kleine, wiederverwendbare Einheiten für Informationsmodelle, vorgestellt, die bestimmte Interessen von Stakeholdern adressieren.

Das Visualisierungsmodell, als Metamodell eines symbolischen Modells, beschreibt die abstrakte Syntax von Softwarekarten und besteht u. a. aus den Gestaltungsmitteln und -regeln, die exemplarisch eingeführt wurden. Mittels der Klassen des Visualisierungsmodells können die Elemente einer Softwarekarte vollständig beschrieben werden.

Die Verbindung zwischen semantischem und symbolischem Modell erfolgt über eine Modelltransformation, die mittels eines Transformationsmodells semantische Modelle in symbolische Modelle (oder umgekehrt) überführt. Diese Kopplung von semantischen und symbolischen Modellen gewährleistet, dass die Softwarekartographie eine Modellierungssprache zur Architekturdokumentation zur Verfügung stellt, deren graphische Modelle (die Softwarekarten) eine definierte Semantik und Syntax besitzen.

Es wurde die exemplarische Anwendungen für Softwarekarten gezeigt und u. a. die Gestaltungsprinzipien *vertikale & horizontale Integration* sowie das *Fassadenprinzip* für Anwendungslandschaften vorgestellt, die Softwarekarten als graphische Modelle verwenden (siehe Kapitel 5). Das Gestaltungsprinzip vertikale & horizontale Integration beschäftigt sich mit der Heterogenität in Anwendungslandschaften, die in Organisationseinheiten mit unterschiedlichen Produktionsstandorten und Ländergesellschaften entstehen können. Das Fassadenprinzip adressiert die Kommunikationsbeziehungen zwischen fachlichen Domänen der Anwendungslandschaft und versucht eine Stabilität der domänenübergreifenden Kommunikationsbeziehungen durch den Einbau von Fassaden zu erreichen. Diese Gestaltungsprinzipien für Anwendungslandschaften beruhen auf den graphischen Modellen der Softwarekartographie und werden erst durch diese anwendbar.

In einer Studie zu Werkzeugen zum Management von Unternehmensarchitekturen wurde aufgezeigt, welche Schwächen existierende Softwarewerkzeuge bei der Visualisierung von

Anwendungslandschaften und Unternehmensarchitekturen besitzen (siehe Kapitel 5). Die Werkzeuge wurden mittels verschiedener Szenarios, die Tätigkeiten in den Managementprozessen (Kapitel 2) widerspiegeln, analysiert und evaluiert. Die Managementprozesse haben hierbei die Möglichkeit eröffnet, die Werkzeuge in unterschiedlichen Bereichen detailliert zu analysieren und die unterschiedlichen Fähigkeiten herauszuarbeiten.

Die softwaretechnische Implementierbarkeit von Softwarekarten wurde prototypisch durch das *Software Cartography Tool* (kurz SoCaTool) belegt. Das Werkzeug basiert auf der vorgestellten Modelltransformation, um aus semantischen Modellen automatisiert symbolische Modelle zu erstellen.

Die in dieser Arbeit entwickelte Modellierungssprache mit dem graphischen Modell der Softwarekarte wird von den Projektpartnern (u. a. BMW Group, Deutsche Post, HVB, Münchener Rück) des Forschungsprojektes Softwarekartographie angewendet, um das Management der Anwendungslandschaft zu unterstützen. Die Ideen und Konzepte von Softwarekarten sind gleichfalls in Werkzeuge, wie bspw. planningIT der alfabet AG und ARIS IT Architect der IDS Scheer AG, eingeflossen.

6.2. Ausblick

Die vorliegende Arbeit hat eine Modellierungssprache für Anwendungslandschaften entwickelt, die Anwendung des zentralen Konzepts Softwarekarte gezeigt und die Managementprozesse für Anwendungslandschaften beschrieben. Aufbauend auf diesen Ergebnissen sind weitere Forschungsarbeiten möglich.

Bei den Informationsmodellen für Unternehmensarchitekturen existiert bisher weder in der Forschung noch in der Praxis ein Konsens, welche Interessen zu adressieren und wie diese auf ein Informationsmodell abzubilden sind. Abschnitt 4.2 hat Informationsmodelle betrachtet, die unterschiedliche Interessen adressieren, und gezeigt, dass die Interessen zu verschiedenen Variationen in Informationsmodellen führen. Aufbauend auf diesen Ergebnissen und den in Abschnitt 4.2 vorgestellten Informationsmodellmustern wird derzeit am Lehrstuhl für Informatik 19 (sebis) der TU München die *Enterprise Architecture Management Viewpoints Survey* durchgeführt. Diese empirische Studie sammelt die bewährtesten Ansätze und führt diese in einem Katalog zusammen. Dieser Katalog verspricht die Bildung von Informationsmodellen zu erleichtern, indem für einzelne Interessen Muster (bestehend aus Methode, Viewpoint und Informationsmodellfragment) als Vorlagen dokumentiert sind.

Parallel zu dieser Arbeit werden am Lehrstuhl für Informatik 19 (sebis) der TU München Metriken zur Bewertung von Anwendungslandschaften entwickelt. Diese Metriken sollen helfen, quantifizierbare Aussagen über Eigenschaften der Anwendungslandschaft zu treffen. Dabei gewonnene Ergebnisse können in die Plan- und Soll-Landschaften einfließen. Ein mögliches Einsatzszenario für Metriken ist die Bewertung von Alternativen bei der Weiterentwicklung von Systemen der Anwendungslandschaft sowie die Entscheidungsunterstützung. Gleichfalls sollen die Metriken die Evolution der Anwendungslandschaft nachvollziehbar machen, indem Kennzahlen die Verbesserungen oder Verschlechterungen

quantifizieren. Die graphische Präsentation dieser Metriken und ihren Kennzahlen erfolgt mit Hilfe der in dieser Arbeit vorgestellten Softwarekarten und den Techniken zur Visualisierung von Kennzahlen aus Abschnitt 4.3.

Für die in Kapitel 2 beschriebene Plattform zum Management von Anwendungslandschaften und für Softwarekarten sind Informationen aus Geschäftsprozessmanagementwerkzeugen, Infrastrukturmanagementsystemen, Projektmanagementsystemen etc. relevant. Der Zusammenhang dieser verschiedenen Informationsquellen mit der Unternehmensarchitektur ist in Abbildung 6.1 dargestellt.

- Geschäftsprozesse werden beim Management der Unternehmensarchitektur verwendet, um den Zusammenhang von Geschäftsprozessen, Anwendungssystemen und Organisationseinheiten zu dokumentieren und zu analysieren. Die Informationen über Geschäftsprozesse werden in Werkzeugen des Geschäftsprozessmanagements verwaltet, welche jedoch häufig nicht mit den Werkzeugen zum Management der Unternehmensarchitektur integriert sind.
- Informationen über die Architektur von Anwendungssystemen werden vom Management der Unternehmensarchitektur verwendet. Die Informationen entstammen Softwaremodellierungswerkzeugen, die häufig ebenso nicht integriert sind.
- Informationen zu Infrastruktur-Services sind beispielsweise in *Configuration Management Databases* (CMDBs) vorhanden, werden jedoch (wenn überhaupt) nur manuell in Werkzeuge zum Management von Unternehmensarchitekturen übertragen.

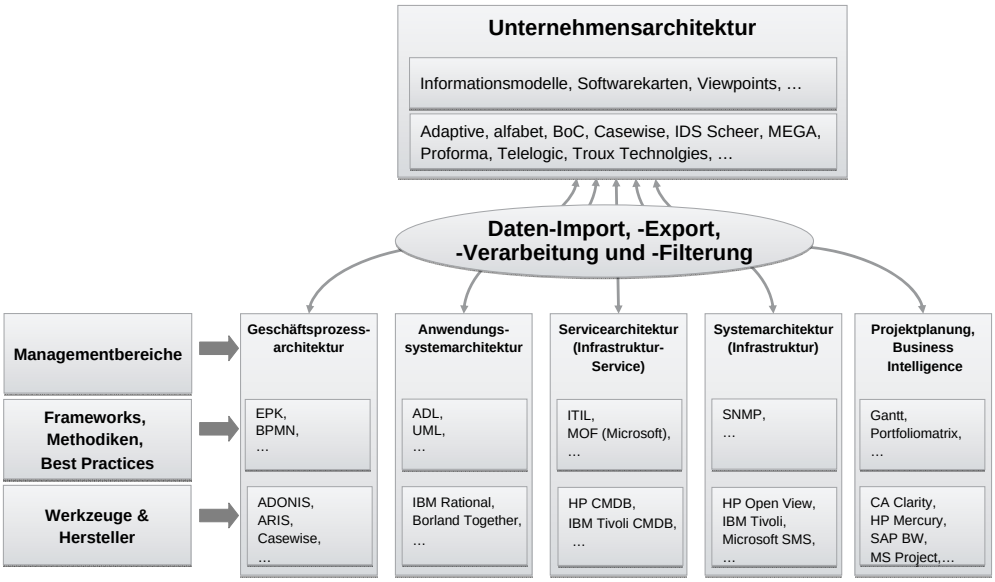


Abbildung 6.1.: Integration von Informationsquellen für die Unternehmensarchitektur (angelehnt an [se05a])

- Verfügbarkeiten von Infrastruktursystemen, wie Datenbanken, Hardwaresysteme etc., sind in Softwaresystemen zum Systemmanagement vorhanden. Fällt beispielsweise ein Hardwaresystem aus, so könnte diese Information bis zum betroffenen Geschäftsprozess verfolgt werden, wenn eine Integration mit Werkzeugen zum Management der Unternehmensarchitektur vorhanden ist.
- Die Planung und Synchronisation von IT-Projekten wurde bereits in den zugehörigen Managementprozessen beschrieben. Diese Informationen werden jedoch häufig durch manuelles Duplizieren der Informationen aus Projektportfoliomanagementwerkzeugen gewonnen. Auch hier fehlt es an einer Integration, um derartige Werkzeuge mit dem Management der Unternehmensarchitektur zu verbinden.

Das Interesse an einer Integration der Quellen, wie in der Abbildung dargestellt, nimmt zu, um Zusammenhänge unterschiedlicher Elemente transparent zu machen. Die Zahl der involvierten Stakeholder (zunehmend auch Fachabteilungen) beim Management der Anwendungslandschaft steigt und somit auch der Bedarf einer Integration, um die Zusammenhänge zu erkennen. Derzeit fehlt es an wohldefinierten Austauschformaten sowie an der Definition von *Datenhoheiten*.

Um die Informationen aus den verschiedenen Quellen auf Softwarekarten in einem geeigneten Abstraktionsgrad und in einer zusammengeführten Form zu visualisieren, müssen für die Informationsmodelle geeignete Transformationsmodelle gefunden werden, um ein semantisches Modell zu erzeugen. Dies semantische Modell kann durch die in Abschnitt 4.4 vorgestellte Modelltransformation in ein symbolisches Modell überführt werden, um eine Softwarekarte zu generieren.

Resümee

Das Management von Anwendungslandschaften steht am Anfang seiner Entwicklung und ist in der Praxis noch nicht vollständig institutionalisiert. Softwarekarten werden zwar auf einer breiten Basis eingesetzt, jedoch wird es noch einige Zeit dauern, bis das Verständnis für den Nutzen dieser Modelle derart gewachsen ist, dass Unternehmen ihre Anwendungslandschaft entlang eines Managementprozesses, welcher diese Modelle verwendet, gestalten.

Ob sich ein Architekturmanagement auf Unternehmensebene langfristig etablieren kann, hängt stark davon ab, wie sich die Schnittstellen und Zuständigkeiten der einzelnen Managementprozesse und -bereiche (siehe Abbildung 6.1) entwickeln. Informationssilos haben nur eine geringe Haltbarkeit, vor allem wenn diese Silos Daten aus anderen Bereichen benötigen, wie es beim Management der Anwendungslandschaft der Fall ist. Somit bleibt abzuwarten wie sich das Management der Anwendungslandschaft entwickeln wird und ob es einen Reifegrad wie das Produktmanagement in Fertigungsunternehmen erlangen kann.

ANHANG A

Merkmale von Anwendungslandschaften

Im Folgenden werden die verschiedenen relevanten Merkmale von Anwendungslandschaften gelistet und den einzelnen Kategorien (siehe Abschnitt 3.3.2) zugeordnet. Neben dem Namen des Merkmals, der Kategorie und dem Typ sowie ggf. dem Skalenniveau wird ebenso eine Beschreibung des Merkmals angegeben. Es wird zwischen Beziehungstypen und Werttypen unterschieden und bei den Werttypen zusätzlich ein Skalenniveau angegeben. Die unterschiedenen Skalenniveaus sind von Fahrmeier et al. [Fa99] übernommen und umfassen Nominalskala, Ordinalskala, Intervallskala und Verhältnisskala:

Nominalskala: Ein Merkmal ist dann nominalskaliert, wenn die einzelnen Ausprägungen der Merkmale lediglich unterscheidbar sind (es ist nicht notwendig, sie ordnen zu können). Nominalskalierte Merkmale besitzen nur abzählbar (endlich oder unendlich) viele verschiedene Merkmalsausprägungen. Soll eine Nominalskala durch Transformation in eine andere überführt werden (ohne dabei den Informationsgehalt zu verändern), so eignen sich dafür alle bijektiven Abbildungen.

Ordinalskala: Ein Merkmal wird dann als ordinal bezeichnet, wenn die einzelnen Merkmalsausprägungen nicht nur unterscheidbar, sondern auch zu ordnen sind, d.h. sich eine Rangordnung auf ihnen errichten lässt. Ordinale Merkmale können überabzählbar viele verschiedene Merkmalsausprägungen besitzen. Bei der Messung auf einer Ordinalskala werden *ranghöheren* Ausprägungen auch höhere Zahlenwerte zugeordnet. Erlaubte, d.h. den Informationsgehalt wahrende, Transformationen auf ordinalen Skalen sind alle isotonen (streng monoton steigenden) Abbildungen.

Intervallskala: Ein Merkmal wird dann als intervallskaliert bezeichnet, wenn die einzelnen Merkmalsausprägungen unterscheidbar sowie zu ordnen sind und der Abstand zweier Ausprägungen ermittelt werden kann. Informationswahrend sind hier sämtliche linearen Transformationen.

Verhältnisskala: Verhältnisskalierte Merkmale müssen neben den Anforderungen an ein intervallskaliertes Merkmal noch eine weitere Bedingung erfüllen: Es muss ein natürlicher Nullpunkt existieren. Informationserhaltend sind hier sämtliche homogenen Transformationen der Form $v' = av$ mit $a > 0$.

Die im Folgenden vorgestellten Merkmale stellen ebenso Anforderungen hinsichtlich ihrer Visualisierung an Softwarekarten.

Planerische/Strategische Merkmale

Name	Programme und Projekte auf der Anwendungslandschaft
Kategorie	Planerisch/Strategisch
Typ	Beziehungstyp und Werttyp (Nominalskala)
Beschreibung	Zur Programm-/Projektplanung und zum Aufzeigen von Abhängigkeiten zwischen Programmen/Projekten sollen Softwarekarten aktuelle und geplante Programme/Projekte in Verbindung mit den betroffenen Anwendungssystemen visualisieren. Durch die starke Verflechtung einer konzernweiten Anwendungslandschaft werden Abhängigkeiten durch eine überblicksartige Darstellung besser erkannt und die betroffenen Programme/Projekte benannt. Die Softwarekarten sollen mehrere Planungsphasen berücksichtigen. Dies Merkmal soll nominalskaliert auf Softwarekarten visualisiert werden.

Name	Lebenszyklus eines Anwendungssystems
Kategorie	Planerisch/Strategisch
Typ	Beziehungstyp und Werttyp (Ordinalskala oder Intervallskala)
Beschreibung	Der Lebenszyklus eines Anwendungssystems beschreibt die verschiedenen Phasen eines Anwendungssystems (in Entwicklung, im Test, in Produktion, in Ablösung, abgelöst) und soll auf Softwarekarten visualisiert werden. Durch das Knüpfen einer Version an das Anwendungssystem können verschiedene Lebenszyklen für ein Anwendungssystem existieren. Das Skalenniveau für die Darstellung als Werttyp ergibt sich aus den zu visualisierenden Informationen: Wird lediglich eine Vorgänger/Nachfolger-Beziehung von Anwendungssystemversionen visualisiert, so ist das Merkmal ordinalskaliert. Werden konkrete Zeitintervalle für die Status der Anwendungssystemversionen angegeben, so ist das Merkmal intervallskaliert und kann auf einer Intervallskala visualisiert werden.

Name	Zeitliche Veränderung der Anwendungslandschaft
Kategorie	Planerisch/Strategisch
Typ	Beziehungstyp und Werttyp (Ordinalskala oder Intervallskala)
Beschreibung	Durch die Langlebigkeit der Anwendungssysteme ist die zeitliche Veränderung der Anwendungslandschaft von besonderem Interesse. Die geplanten und getätigten Veränderungen der Anwendungslandschaft durch verschiedene IT-Projekte sollen auf Softwarekarten dargestellt werden. Durch das Überlagern zweier Softwarekarten mit gleichen Merkmalen unterschiedlichen Datums kann das Erreichen von strategischen Zielen oder Projektzielen belegt werden. Das Skalenniveau für die Darstellung als Werttyp ergibt sich aus den zu visualisierenden Informationen: Wird lediglich eine Vorgänger/Nachfolger-Beziehung von Anwendungssystemen visualisiert, so ist das Merkmal ordinalskaliert. Werden konkrete Zeitintervalle für die Status der Anwendungssystemversionen angegeben, so ist das Merkmal intervallskaliert und kann auf einer Intervallskala visualisiert werden.

Name	Soll-Landschaft
Kategorie	Planerisch/Strategisch
Typ	Beziehungstyp und Werttyp (Nominalskala)
Beschreibung	Strategische Ziele, beispielsweise das Reduzieren der Anzahl von Individualsoftware, sollen auf Softwarekarten visualisiert werden, dies ggf. ohne Bezug zu einem Projekt oder Programm. Dies Merkmal soll nominalskaliert auf Softwarekarten visualisiert werden.

Fachliche Merkmale

Name	Gruppierung von Anwendungssystemen zu logischen Einheiten
Kategorie	Fachlich
Typ	Beziehungstyp
Beschreibung	Die Gruppierung von Anwendungssystemen zu logischen Einheiten (z.B. Funktionsbereichen, Organisationseinheiten) soll auf Softwarekarten möglich sein.

Name	Nutzungsintensität je Anwendungssystem
Kategorie	Fachlich
Typ	Werttyp (Verhältnisskala)
Beschreibung	Die Nutzungsintensität von Anwendungssystemen, beispielsweise die Nutzungszeit pro Benutzer pro Tag oder die Anzahl von Geschäftsvorfällen pro Nutzer (siehe auch *Transaktionsraten* und *Datenvolumen je Anwendungssystem*), sollen auf Softwarekarten visualisiert werden. Die Softwarekarten können für Statistiken oder zur Erkennung von hochfrequentierten Anwendungssystemen genutzt werden. Eine Kombination mit wirtschaftlichen Merkmalen führt zur einer Kosten/Nutzen-Funktion.

Name	Anzahl von Nutzern je Anwendungssystem
Kategorie	Fachlich
Typ	Werttyp (Verhältnisskala)
Beschreibung	Die Anzahl von Nutzern (registrierte Nutzer etc.) ist neben anderen Faktoren, wie beispielsweise der Transaktionsrate, ein Mittel zur Bewertung der Bedeutung eines Anwendungssystems. Des Weiteren kann die Anzahl von Nutzern (ebenso wie die *Nutzungsintensität je Anwendungssystem*) für eine Kosten/Nutzen-Funktion verwendet werden.

Name	Nutzergruppen je Anwendungssystem
Kategorie	Fachlich
Typ	Nominalskala oder Intervallskala
Beschreibung	Die Nutzergruppen können Aufschluss über die Arten von Arbeitsplätzen in einem Unternehmen geben. Diese Zuordnung von Nutzergruppen zu Anwendungssystemen hilft bei der Bildung von logischen Einheiten und kann sinnvolle Gruppierungen durch verwandte Nutzungsszenarios aufzeigen (Ordinalskala). Zusätzlich soll die Anzahl von Nutzergruppen je Anwendungssystem als Messwert visualisiert werden können (Intervallskala).

Name	Funktionen je Anwendungssystem
Kategorie	Fachlich
Typ	Verhältnisskala
Beschreibung	Mittels eines Messwertes wie beispielsweise *Function Points* soll die Funktionalität der Anwendungssysteme quantifiziert und auf Softwarekarten abgebildet werden. Die zeitliche Veränderung dieses Messwertes (steigend/fallend/konstant) in Kombination mit der Veränderung der Wartungskosten kann als Indikator für ein mögliches Reengineering oder eine Performanzoptimierung genutzt werden.

Name	Unterstützte Geschäftsprozesse je Anwendungssystem
Kategorie	Fachlich
Typ	Beziehungstyp
Beschreibung	Die Zuordnung von Funktionen – im Sinne der unterstützenden Geschäftsprozesse – zu Anwendungssystemen soll auf Softwarekarten visualisiert werden. In Kombination mit logischen Einheiten sollen Anwendungssysteme mit gleichem funktionalen Charakter in unterschiedlichen Einheiten erkannt werden.

Name	Geschäftsobjekte je Anwendungssystem
Kategorie	Fachlich
Typ	Beziehungstyp
Beschreibung	Durch die große Anzahl von Anwendungssystemen einer Anwendungslandschaft existieren in unterschiedlichen Anwendungssystemen dieselben Geschäftsobjekte. Softwarekarten sollen diese redundante Datenhaltung visualisieren und ebenso die für einzelne Geschäftsobjekte *führenden* Anwendungssysteme hervorheben. Zusätzlich sollen Geschäftsobjekte, die über Schnittstellen ausgetauscht werden in Kombination mit dem Merkmal *Kommunikationsbeziehungen* dargestellt werden.

Name	Unterstützte Produkte je Anwendungssystem
Kategorie	Fachlich
Typ	Beziehungstyp
Beschreibung	Die Zuordnung von Produkten oder Produktgruppen, die durch die Wertschöpfung des Unternehmens entstehen, zu Anwendungssystemen soll auf Softwarekarten visualisiert werden können. In Kombination mit logischen Einheiten oder Geschäftsprozessen sollen Anwendungssysteme mit gleichem funktionalen Charakter in unterschiedlichen Einheiten oder Geschäftsprozessen erkannt werden.

Technische Merkmale

Name	Transaktionsrate je Anwendungssystem
Kategorie	Technisch
Typ	Werttyp (Verhältnisskala)
Beschreibung	Die Transaktionsrate im Sinne von Geschäftstransaktionen (z.B. Überweisungen pro Minute) soll die Nutzung eines Anwendungssystems im technischen Sinne (siehe *Nutzungsintensität*) abbilden. Wird zusätzlich eine zeitliche Komponente (24-Stundenskala, Monatsskala etc.) verwendet, so können beispielsweise Lastzeiten erkannt werden. Wird die Anzahl von Transaktionen nicht nur auf Anwendungssystemebene gemessen, sondern ebenso auf Ebene eines Hardwaresystems, so kann dieses Maß zur Verrechnung der Kosten eines Hardwaresystems genutzt werden. Insbesondere bei Host-Systemen, die eine große Anzahl von Anwendungen gleichzeitig betreiben, ist eine derartige Kostenrechnung relevant.

Name	Datenvolumen je Anwendungssystem
Kategorie	Technisch
Typ	Werttyp (Verhältnisskala)
Beschreibung	Das genutzte Datenvolumen eines Anwendungssystems zur Speicherung von Anwendungsdaten soll auf Softwarekarten visualisiert werden. Der Nutzen dieser Softwarekarten und dieses Merkmals ist vergleichbar mit *Transaktionsrate je Anwendungssystem*. Statistische Auswertungen mit zeitlichem Verlauf der Datenvolumen und die Zuordnung von Datenvolumen je Anwendungssystem in einem Datenbanksystem sollen dargestellt werden.

Name	Individual- vs. Standardsoftware
Kategorie	Technisch
Typ	Werttyp (Nominalskala)
Beschreibung	Die Unterscheidung von Individual- und Standardsoftware soll auf Softwarekarten visualisiert werden. Zusätzlich kann bei Individualsoftware zwischen Eigen- und Fremdentwicklung bzw. einer Kombination unterschieden werden, um eine feinere Detaillierung zu erreichen und zusätzlich Abhängigkeiten von externem Know-how bei Veränderungen an einem oder umgebenden Anwendungssystemen zu erkennen.

Name	Kommunikationsbeziehungen
Kategorie	Technische
Typ	Beziehungstyp
Beschreibung	Durch die starke Vernetzung einer Anwendungslandschaft sind für Softwarekarten die Kommunikationsbeziehungen von Anwendungssystemen relevant, sowohl die einzelnen Kommunikationswege als auch die Art der Kommunikation müssen betrachtet werden. Bei den Kommunikationswegen können die Kommunikationspartner (System A kommuniziert mit System B) sowie die Richtung des Kontrollflusses erfasst werden. Die Kommunikationsarten (Online vs. Batch vs. Manuell), die Art des Zugriffs (lesend vs. schreibend) und Implementierungstechnologien der Schnittstellen-Konnektoren können ebenso unterschieden werden (siehe *Konnektoren*) [MW04b].

Name	Konnektoren für Kommunikationsbeziehungen
Kategorie	Technisch
Typ	Beziehungstyp
Beschreibung	Um die Kommunikationsbeziehungen detaillierter zu analysieren, sind die Kommunikationsarten zu betrachten, die sich aus der Implementierung der fachlichen und technischen Konnektoren ergeben. Die technischen Konnektoren implementieren fachliche Konnektoren mittels einer bestimmten Technologie, so dass hieraus die Kommunikationsart (z.B. *Remote Method Invocation*, Batch-Datei, Fax etc.) resultiert. Verwendet der technische Konnektor eine Middleware (z.B. IBM MQSeries), so ist dies geeignet zu berücksichtigen [MW04b].

Name	Implementierungssprache je Anwendungssystem
Kategorie	Technisch
Typ	Beziehungstyp und Werttyp (Nominalskala oder Verhältnisskala)
Beschreibung	Die Anwendungssysteme einer Anwendungslandschaft sind in verschiedenen Programmiersprachen (C, C++, Java, Cobol, Pascal, ABAP etc.) realisiert, wobei ein einzelnes Anwendungssystem in mehreren Programmiersprachen implementiert sein kann. Die Anforderung, Anwendungssysteme hinsichtlich der Implementierungssprachen zu betrachten, resultiert aus der Mannigfaltigkeit von Programmiersprachen in existierenden Anwendungslandschaften. In Kombination mit dem Merkmal *Zeitliche Veränderung der Anwendungslandschaft* und *Individual- vs. Standardsoftware* ist erkennbar, wie sich die Anwendungslandschaft und das benötigte Know-how zur Wartung/Weiterentwicklung verändern. Der quantifizierbare Anteil dieses Merkmals, der in einer Verhältnisskala für das Merkmal resultiert, ergibt sich, wenn die Häufigkeit der Implementierungssprachen betrachtet wird. Die Häufigkeit einer Implementierungssprache in einer Anwendungslandschaft lässt einen Rückschluss auf das benötigte Know-how der Anwendungsentwickler bzw. Externen zu. Eine Erweiterung um *Lines of Code* je Implementierungssprache in einem Anwendungssystem erhöht die Aussagekraft.

Name	Softwarearchitektur je Anwendungssystem
Kategorie	Technisch
Typ	Beziehungstyp und Werttyp (Nominalskala)
Beschreibung	In Abhängigkeit von der Softwarearchitektur sollen die Anwendungssysteme auf Softwarekarten unterschieden werden. Dies Merkmal soll nominalskaliert auf Softwarekarten visualisiert werden.

Name	IT-Architekturkonformität je Anwendungssystem
Kategorie	Technisch
Typ	Beziehungstyp und Werttyp (Nominalskala)
Beschreibung	Die Konformität mit unternehmensspezifischen IT-Architekturen (bspw. Musterarchitekturen oder Softwarearchitekturbebauungsplänen) von Anwendungssystemen soll auf Softwarekarten dargestellt werden können. Derartige Vorgaben für IT-Architekturen in der Anwendungslandschaft besitzen Auswirkungen auf die Wartbarkeit und den Betrieb von Anwendungssysteme, da die Vorgaben unmittelbar mit dem Angebot der IT-Betriebsabteilung in Zusammenhang stehen und ebenso strategische und planerische Vorgaben widerspiegeln. Dies Merkmal soll nominalskaliert auf Softwarekarten visualisiert werden.

Name	Genutzte Middleware je Anwendungssystem
Kategorie	Technisch
Typ	Beziehungstyp und Werttyp (Nominalskala oder Verhältnisskala)
Beschreibung	Die Verbindung von Anwendungssystemen und den genutzten Middleware-Systemen ist für Softwarekarten relevant, mögliche Ausprägungen sind beispielsweise *Web Application Server*, *Transaction Server* oder *Messaging Server*. Auf einer detaillierteren Ebene können bestimmte Technologien oder Hersteller (z.B. IBM, BEA, Oracle oder Siemens) zugeordnet werden. Des Weiteren können Abhängigkeiten von bestimmten Produkten und Versionen in die Visualisierung einfließen, um bei Migrationen die betroffenen Systeme zu identifizieren. Der quantifizierbare Anteil dieses Merkmals ergibt sich, wenn die Häufigkeit der auftretenden Middleware-Systeme betrachtet wird (Verhältnisskala).

Name	Genutzte Datenbankmanagementsysteme je Anwendungssystem
Kategorie	Technisch
Typ	Beziehungstyp und Werttyp (Nominalskala oder Verhältnisskala)
Beschreibung	Anwendungssysteme nutzen verschiedene Datenbanktechnologien und Datenbankmanagementsysteme zur persistenten Speicherung von Daten. Zum Zweck der Homogenisierung der Anwendungslandschaft (Reduktion von Lizenzkosten, Know-how etc.) können unterschiedliche und gleiche Datenbankmanagementsysteme der Anwendungssysteme erkannt und analog zu *Genutzte Middleware je Anwendungssystem* ebenso Abhängigkeiten zu bestimmten Produkten und Versionen aufgezeigt werden. Der quantifizierbare Anteil dieses Merkmals ergibt sich, wenn die Häufigkeit der auftretenden Datenbankmanagementsysteme betrachtet wird (Verhältnisskala).

Name	Genutztes Betriebssystem je Anwendungssystem
Kategorie	Technisch
Typ	Beziehungstyp und Werttyp (Nominalskala oder Verhältnisskala)
Beschreibung	Zum Zweck der Homogenisierung der Anwendungslandschaft hinsichtlich der Betriebssysteme (Reduktion von Lizenzkosten, Know-how etc.) können unterschiedliche und gleiche Betriebssysteme der Anwendungssysteme erkannt und analog zu *Genutzte Middleware je Anwendungssystem* ebenso Abhängigkeiten zu bestimmten Betriebssystemen und Versionen aufgezeigt werden. Der quantifizierbare Anteil dieses Merkmals ergibt sich, wenn die Häufigkeit der auftretenden Betriebssysteme betrachtet wird (Verhältnisskala).

Name	Hardwaresystem je Anwendungssystem
Kategorie	Technisch
Typ	Beziehungstyp und Werttyp (Nominalskala oder Verhältnisskala)
Beschreibung	Zum Zweck der Homogenisierung der Anwendungslandschaft (Gemeinsame Ressourcennutzung, Know-how etc.) können unterschiedliche und gleiche Hardwaresysteme der Anwendungssysteme erkannt und analog zu *Genutzte Middleware je Anwendungssystem* ebenso Abhängigkeiten zu bestimmten Produkten aufgezeigt werden. Der quantifizierbare Anteil dieses Merkmals ergibt sich, wenn die Häufigkeit der auftretenden Hardwaresysteme betrachtet wird (Verhältnisskala).

Name	Skalierbarkeit und Fehlertoleranz je Anwendungssystem
Kategorie	Technisch
Typ	Nominalskala oder Ordinalskala
Beschreibung	Softwarekarten können die Eigenschaften der Anwendungssysteme hinsichtlich Skalierbarkeit und Fehlertoleranz darstellen. Das Skalenniveau ist abhängig von der gewählten Form der Quantisierung.

Wirtschaftliche Merkmale

Name	Kosten je Anwendungssystem
Kategorie	Wirtschaftlich
Typ	Werttyp (Verhältnisskala)
Beschreibung	Verschiedene Kostenarten (Wartungskosten, Bereitschaftskosten etc.) sollen auf Softwarekarten visualisiert werden. Neben den Kosten in einer bestimmten Periode sollen zeitliche Verläufe der einzelnen Kostenarten abgebildet werden.

Name	Kapitalwert der Anwendungssysteme
Kategorie	Wirtschaftlich
Typ	Werttyp (Verhältnisskala)
Beschreibung	Eine Kapitalwertanalyse der Anwendungssysteme mit zeitlichem Verlauf des Kapitalwertes kann beispielsweise einen Handlungsbedarf für ein Reengineering aufzeigen. Softwarekarten sollen verschiedene zeitliche Verläufe und Ist-Werte visualisieren.

Operative Merkmale

Name	Ausfall- und Betriebszeiten je Anwendungssystem
Kategorie	Operativ
Typ	Werttyp (Intervallskaliert)
Beschreibung	Die Anzahl von Systemausfällen, die Ausfallzeiten sowie die prozentuale Systemverfügbarkeit sollen auf Softwarekarten dargestellt werden. Eine Detaillierung wie bei einem *IT-Cockpit*, welches sämtliche Ausfallzeiten, Fehleranalysen und Performanzmessungen analysiert und archiviert, ist für Softwarekarten nur bedingt relevant, da derartige Anforderungen bereits durch geeignete Werkzeuge unterstützt werden.

Name	Betriebsort je Anwendungssystem
Kategorie	Operativ
Typ	Beziehungstyp und Werttyp (Verhältnisskala)
Beschreibung	Softwarekarten sollen die unterschiedlichen Betriebsorte der Systeme darstellen. Unternehmen, dessen Anwendungssysteme und Hardwaresysteme geographisch verteilt sind, wollen eine überblicksartige Darstellung der Anwendungslandschaft, wobei ein Detaillierungsgrad auf Standortebene als ausreichend angesehen wird. Der quantifizierbare Anteil ergibt sich, wenn die Anzahl von Anwendungssystemen pro Standort ermittelt wird (Verhältnisskala).

Name	Nutzungsorte je Anwendungssystem
Kategorie	Operativ
Typ	Beziehungstyp und Werttyp (Verhältnisskala)
Beschreibung	Analog zu den Betriebsorten sollen auch die Nutzungsorte (z.B. Standorte des Unternehmens) visualisiert werden. Eine Betrachtung der zeitlichen Entwicklung dieses Merkmals kann die Analyse ergänzen. Der quantifizierbare Anteil ergibt sich analog zum Merkmal *Betriebsort je Anwendungssystem* (Verhältnisskala).

Name	Erfüllung des SLAs je Anwendungssystem
Kategorie	Operativ
Typ	Werttyp (Intervallskaliert)
Beschreibung	Der Erfüllungsgrad eines Service-Level-Agreements (SLA) soll auf einer Softwarekarte dargestellt werden.

ANHANG B

Ausschnitte von Informationsmodellen

Im Folgenden sind Ausschnitte der in Abschnitt 4.2.2 diskutierten Informationsmodelle dargestellt:

- Abbildung B.1 zeigt einen Ausschnitt des *Common Information Models* (CIM).
- Abbildung B.2 zeigt einen Ausschnitt der *IT Portfolio Managment Facility* (ITPMF).
- Abbildung B.3 zeigt einen Ausschnitt des Informationsmodells von *ArchiMate*.

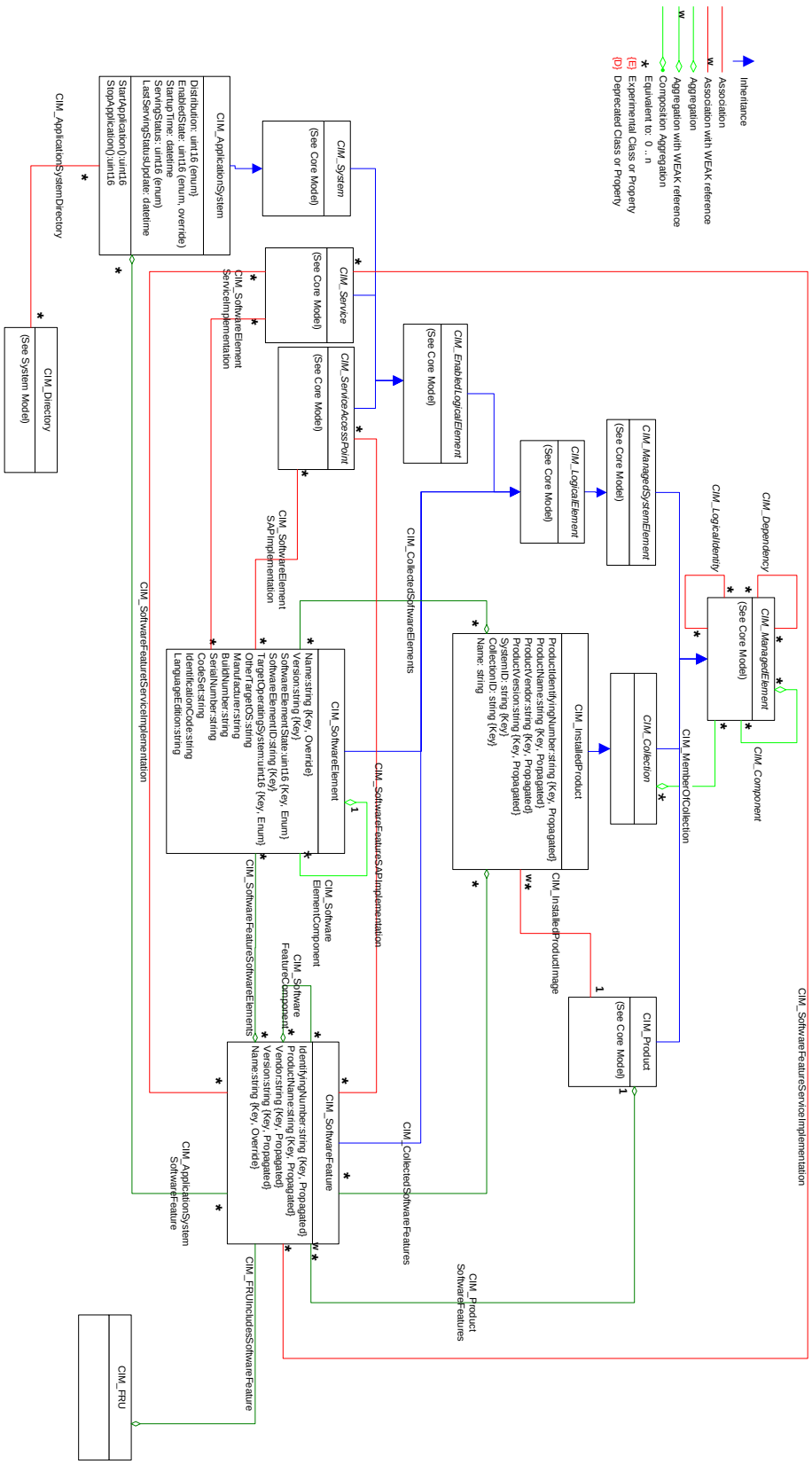


Abbildung B.1.: CIM: Ausschnitt der *Application Model* Spezifikation [DM06a]

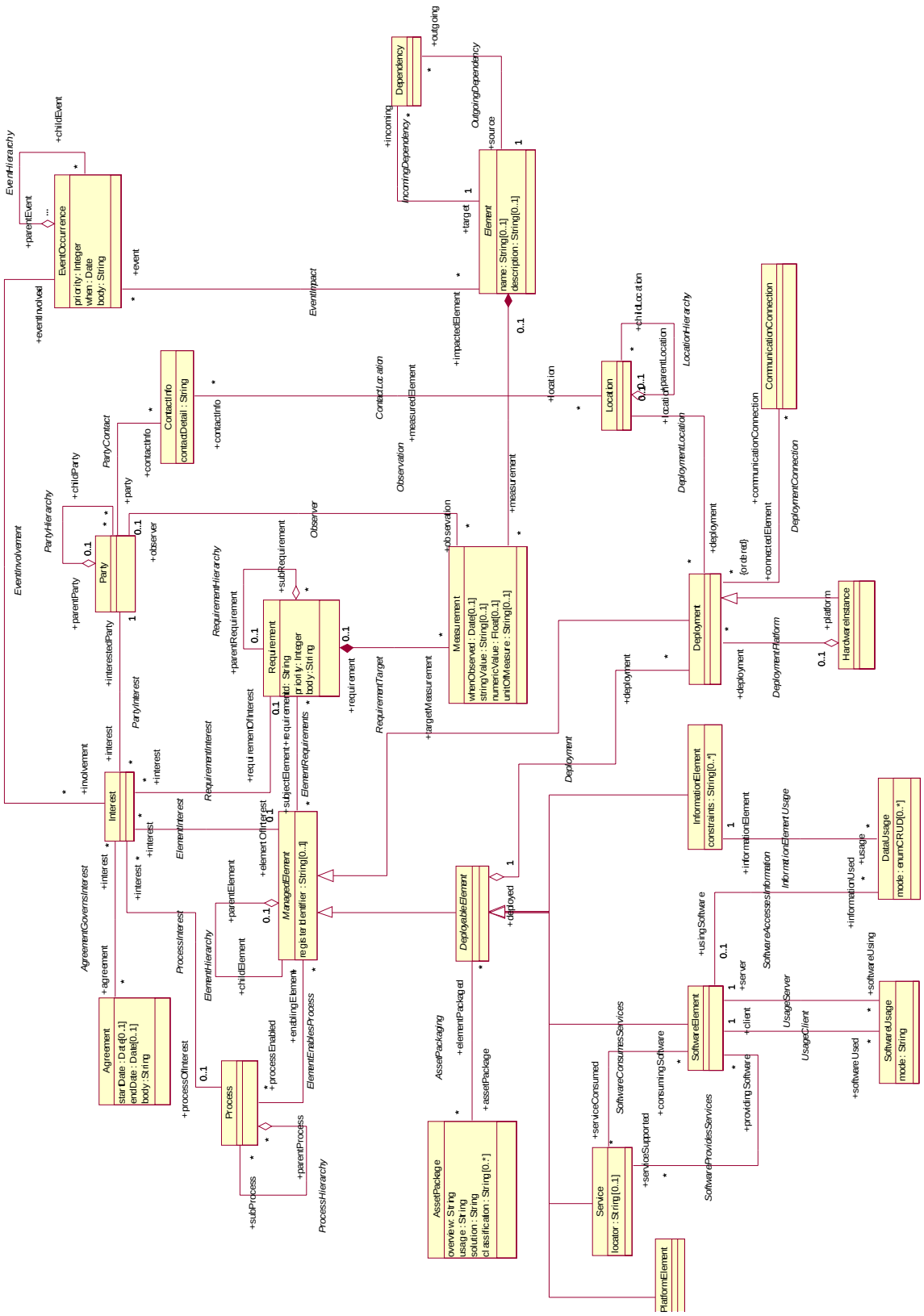


Abbildung B.2.: ITPMF: Kernelemente im Informationsmodell [OM06c]

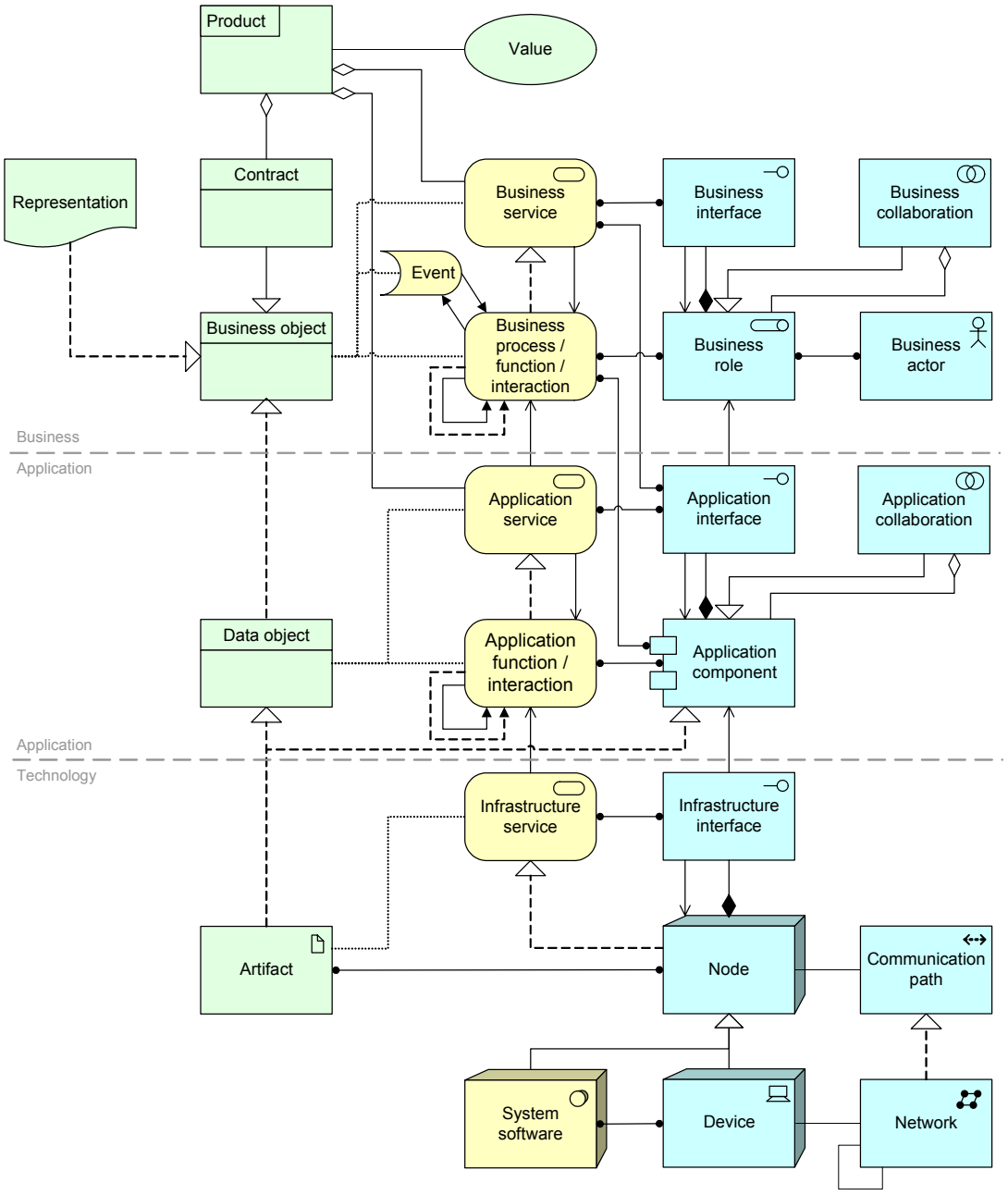


Abbildung B.3.: ArchiMate: Kernelemente im Informationsmodell [Jo06]

ANHANG C

Serialisierung von Softwarekarten

Die folgenden Abbildungen und Quelltexte zeigen Beispiele für Softwarekarten und ihre Serialisierungen.

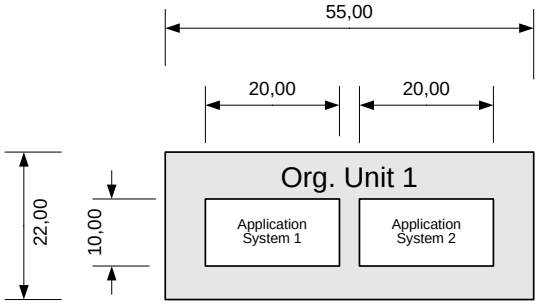


Abbildung C.1.: Clusterkarte zur Serialisierung in Abbildung C.1

```
1 <?xml version="1.0" encoding="ASCII"?>
2 <SoCaVisualizationModel:SoftwareMap xmi:version="2.0" xmlns:xmi="http
      ://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance" xmlns:ConstraintVisualizationRules="http://www.
      softwarekartographie.de/visModel1.1/SoCaVisualizationModel/
      ConstraintVisualizationRules/" xmlns:MapSymbols="http://www.
      softwarekartographie.de/visModel1.1/SoCaVisualizationModel/
      MapSymbols/" xmlns:SoCaVisualizationModel="http://www.
      softwarekartographie.de/visModel1.1/SoCaVisualizationModel/"
      xmlns:TargetVisualizationRules="http://www.softwarekartographie.
      de/visModel1.1/SoCaVisualizationModel/TargetVisualizationRules/"
      title="Cluster Map" creationDate="2007-04-24T17:26:37.808+0200"
      creator="A. Wittenburg">
```

```
3  <diagram>
4    <element xsi:type="MapSymbols:Rectangle" id="6" rulingLayer="25"
         width="55.0" height="22.0" textHorizontalAlignment="center"
         areaRule="27" nestingRule="22 23" borderStyle="solid">
5      <center id="3" x="22.5" y="11.0"/>
6      <borderColor id="5" red="0" green="0" blue="0"/>
7      <fillColor id="4" red="230" green="230" blue="230"/>
8      <text id="2" value="Org. Unit 1" fontSize="12">
9        <color id="0" red="0" green="0" blue="0"/>
10       <fontType id="1" name="Arial"/>
11     </text>
12   </element>
13   <element xsi:type="MapSymbols:Rectangle" identityRule="24" id
         ="13" rulingLayer="26" separationRule="21" width="20.0"
         height="10.0" textHorizontalAlignment="center"
         textVerticalAlignment="middle" nestedRule="22" borderStyle="
         solid">
14     <center id="10" x="16.0" y="12.0"/>
15     <borderColor id="12" red="0" green="0" blue="0"/>
16     <fillColor id="11" red="255" green="255" blue="255"/>
17     <text id="9" value="Application System 1" fontSize="6">
18       <color id="7" red="0" green="0" blue="0"/>
19       <fontType id="8" name="Arial"/>
20     </text>
21   </element>
22   <element xsi:type="MapSymbols:Rectangle" identityRule="24" id
         ="20" rulingLayer="26" separationRule="21" width="20.0"
         height="10.0" textHorizontalAlignment="center"
         textVerticalAlignment="middle" nestedRule="23" borderStyle="
         solid">
23     <center id="17" x="39.0" y="12.0"/>
24     <borderColor id="19" red="0" green="0" blue="0"/>
25     <fillColor id="18" red="255" green="255" blue="255"/>
26     <text id="16" value="Application System 2" fontSize="6">
27       <color id="14" red="0" green="0" blue="0"/>
28       <fontType id="15" name="Arial"/>
29     </text>
30   </element>
31   <element xsi:type="ConstraintVisualizationRules:Separation" id
         ="21" separated="13 20"/>
32   <element xsi:type="ConstraintVisualizationRules:Nesting" id="22"
         nestedSymbol="13" nestingSymbol="6"/>
33   <element xsi:type="ConstraintVisualizationRules:Nesting" id="23"
         nestedSymbol="20" nestingSymbol="6"/>
34   <element xsi:type="ConstraintVisualizationRules:
         IdentityOfProjection" id="24" ruledIdentical="13 20">
35     <constrainedAttribute>width</constrainedAttribute>
36     <constrainedAttribute>height</constrainedAttribute>
37     <constrainedAttribute>fillColor</constrainedAttribute>
38     <constrainedAttribute>borderColor</constrainedAttribute>
39   </element>
```

```
40      <element xsi:type="ConstraintVisualizationRules:VisibilityLayer"
            id="25" name="Base map" layeredSymbol="6" visible="true"/>
41      <element xsi:type="ConstraintVisualizationRules:VisibilityLayer"
            id="26" name="Application Systems" layeredSymbol="13 20"
            visible="true"/>
42      <element xsi:type="TargetVisualizationRules:AreaMinimization" id
            ="27" ruledArea="6"/>
43    </diagram>
44    <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#101" visualizationelement="6"/>
45    <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#201" visualizationelement="13"/>
46    <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#202" visualizationelement="20"/>
47  </SoCaVisualizationModel:SoftwareMap>
```

Quelltext C.1: Serialisierung der Clusterkarte aus Abbildung C.1

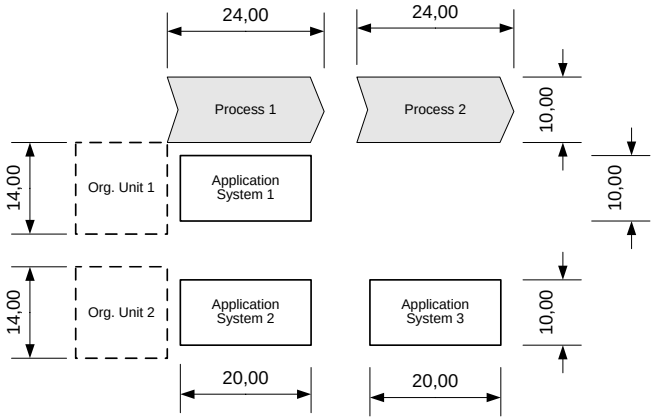


Abbildung C.2.: Kartesische Karte zur Serialisierung in Abbildung C.2

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <SoCaVisualizationModel:SoftwareMap xmi:version="2.0" xmlns:xmi="http
       ://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance" xmlns:ConstraintVisualizationRules="http://www.
       softwarekartographie.de/visModel1.1/SoCaVisualizationModel/
       ConstraintVisualizationRules/" xmlns:MapSymbols="http://www.
       softwarekartographie.de/visModel1.1/SoCaVisualizationModel/
       MapSymbols/" xmlns:SoCaVisualizationModel="http://www.
       softwarekartographie.de/visModel1.1/SoCaVisualizationModel/"
       title="Process Support Map" creationDate="2007-04-24T17
       :26:37.738+0200" creator="A. Wittenburg">
3    <diagram>
4      <element xsi:type="MapSymbols:Rectangle" identityRule="51" id="6"
           rulingLayer="60" width="14.0" height="14.0"
          textHorizontalAlignment="center" textVerticalAlignment="
          middle" ysequenceRule="50" yintersectedRule="54" borderStyle
          ="dashed">
```

```
5         <center id="3" x="7.0" y="17.0"/>
6         <borderColor id="5" red="0" green="0" blue="0"/>
7         <fillColor id="4" red="255" green="255" blue="255"/>
8         <text id="2" value="Org. Unit 1" fontSize="6">
9           <color id="0" red="0" green="0" blue="0"/>
10          <fontType id="1" name="Arial"/>
11        </text>
12      </element>
13      <element xsi:type="MapSymbols:Rectangle" identityRule="51" id
            ="13" rulingLayer="60" width="14.0" height="14.0"
            textHorizontalAlignment="center" textVerticalAlignment="
            middle" ysequenceRule="50" yintersectedRule="55 56"
            borderStyle="dashed">
14        <center id="10" x="7.0" y="36.0"/>
15        <borderColor id="12" red="0" green="0" blue="0"/>
16        <fillColor id="11" red="255" green="255" blue="255"/>
17        <text id="9" value="Org. Unit 2" fontSize="6">
18          <color id="7" red="0" green="0" blue="0"/>
19          <fontType id="8" name="Arial"/>
20        </text>
21      </element>
22      <element xsi:type="MapSymbols:Rectangle" identityRule="52" id
            ="20" rulingLayer="61" width="20.0" height="10.0"
            textHorizontalAlignment="center" textVerticalAlignment="
            middle" xintersectedRule="57" yintersectingRule="54"
            borderStyle="solid">
23        <center id="17" x="26.0" y="17.0"/>
24        <borderColor id="19" red="0" green="0" blue="0"/>
25        <fillColor id="18" red="255" green="255" blue="255"/>
26        <text id="16" value="Application System 1" fontSize="6">
27          <color id="14" red="0" green="0" blue="0"/>
28          <fontType id="15" name="Arial"/>
29        </text>
30      </element>
31      <element xsi:type="MapSymbols:Rectangle" identityRule="52" id
            ="27" rulingLayer="61" width="20.0" height="10.0"
            textHorizontalAlignment="center" textVerticalAlignment="
            middle" xintersectedRule="58" yintersectingRule="55"
            borderStyle="solid">
32        <center id="24" x="26.0" y="36.0"/>
33        <borderColor id="26" red="0" green="0" blue="0"/>
34        <fillColor id="25" red="255" green="255" blue="255"/>
35        <text id="23" value="Application System 2" fontSize="6">
36          <color id="21" red="0" green="0" blue="0"/>
37          <fontType id="22" name="Arial"/>
38        </text>
39      </element>
40      <element xsi:type="MapSymbols:Rectangle" identityRule="52" id
            ="34" rulingLayer="61" width="20.0" height="10.0"
            textHorizontalAlignment="center" textVerticalAlignment="
            middle" xintersectedRule="59" yintersectingRule="56"
            borderStyle="solid">
```

```
41        <center id="31" x="55.0" y="36.0"/>
42        <borderColor id="33" red="0" green="0" blue="0"/>
43        <fillColor id="32" red="255" green="255" blue="255"/>
44        <text id="30" value="Application System 3" fontSize="6">
45          <color id="28" red="0" green="0" blue="0"/>
46          <fontType id="29" name="Arial"/>
47        </text>
48      </element>
49      <element xsi:type="MapSymbols:Chevron" identityRule="53" id="41"
            rulingLayer="60" width="24.0" height="10.0"
            textHorizontalAlignment="center" textVerticalAlignment="
            middle" xintersectingRule="57 58" xsequenceRule="49"
            borderStyle="solid" inset="2.0">
50        <center id="38" x="26.0" y="5.0"/>
51        <borderColor id="40" red="0" green="0" blue="0"/>
52        <fillColor id="39" red="255" green="255" blue="255"/>
53        <text id="37" value="Process 1" fontSize="6">
54          <color id="35" red="0" green="0" blue="0"/>
55          <fontType id="36" name="Arial"/>
56        </text>
57      </element>
58      <element xsi:type="MapSymbols:Chevron" identityRule="53" id="48"
            rulingLayer="60" width="24.0" height="10.0"
            textHorizontalAlignment="center" textVerticalAlignment="
            middle" xintersectingRule="59" xsequenceRule="49" borderStyle
            ="solid" inset="2.0">
59        <center id="45" x="55.0" y="5.0"/>
60        <borderColor id="47" red="0" green="0" blue="0"/>
61        <fillColor id="46" red="255" green="255" blue="255"/>
62        <text id="44" value="Process 2" fontSize="6">
63          <color id="42" red="0" green="0" blue="0"/>
64          <fontType id="43" name="Arial"/>
65        </text>
66      </element>
67      <element xsi:type="ConstraintVisualizationRules:XSequence" id
            ="49" ruledXSequence="41 48"/>
68      <element xsi:type="ConstraintVisualizationRules:YSequence" id
            ="50" ruledYSequence="6 13"/>
69      <element xsi:type="ConstraintVisualizationRules:
            IdentityOfProjection" id="51" ruledIdentical="6 13">
70        <constrainedAttribute>width</constrainedAttribute>
71        <constrainedAttribute>height</constrainedAttribute>
72        <constrainedAttribute>fillColor</constrainedAttribute>
73        <constrainedAttribute>borderColor</constrainedAttribute>
74      </element>
75      <element xsi:type="ConstraintVisualizationRules:
            IdentityOfProjection" id="52" ruledIdentical="20 27 34">
76        <constrainedAttribute>width</constrainedAttribute>
77        <constrainedAttribute>height</constrainedAttribute>
78        <constrainedAttribute>fillColor</constrainedAttribute>
79        <constrainedAttribute>borderColor</constrainedAttribute>
80      </element>
```

```
81      <element xsi:type="ConstraintVisualizationRules:
            IdentityOfProjection" id="53" ruledIdentical="41 48">
82        <constrainedAttribute>width</constrainedAttribute>
83        <constrainedAttribute>height</constrainedAttribute>
84        <constrainedAttribute>inset</constrainedAttribute>
85        <constrainedAttribute>fillColor</constrainedAttribute>
86        <constrainedAttribute>borderColor</constrainedAttribute>
87      </element>
88      <element xsi:type="ConstraintVisualizationRules:
            FullYSpecificIntersection" id="54" yintersectedSymbol="6"
            yintersectingSymbol="20"/>
89      <element xsi:type="ConstraintVisualizationRules:
            FullYSpecificIntersection" id="55" yintersectedSymbol="13"
            yintersectingSymbol="27"/>
90      <element xsi:type="ConstraintVisualizationRules:
            FullYSpecificIntersection" id="56" yintersectedSymbol="13"
            yintersectingSymbol="34"/>
91      <element xsi:type="ConstraintVisualizationRules:
            FullXSpecificIntersection" id="57" xintersectedSymbol="41"
            xintersectingSymbol="20"/>
92      <element xsi:type="ConstraintVisualizationRules:
            FullXSpecificIntersection" id="58" xintersectedSymbol="41"
            xintersectingSymbol="27"/>
93      <element xsi:type="ConstraintVisualizationRules:
            FullXSpecificIntersection" id="59" xintersectedSymbol="48"
            xintersectingSymbol="34"/>
94      <element xsi:type="ConstraintVisualizationRules:VisibilityLayer"
            id="60" name="Base map" layeredSymbol="41 6 48 13" visible="
            true"/>
95      <element xsi:type="ConstraintVisualizationRules:VisibilityLayer"
            id="61" name="Application Systems" layeredSymbol="20 27 34"
            visible="true"/>
96    </diagram>
97    <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#101" visualizationelement="6"/>
98    <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#102" visualizationelement="13"/>
99    <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#201" visualizationelement="20"/>
100   <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#202" visualizationelement="27"/>
101   <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#203" visualizationelement="34"/>
102   <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#301" visualizationelement="41"/>
103   <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#302" visualizationelement="48"/>
104 </SoCaVisualizationModel:SoftwareMap>
```

Quelltext C.2: Serialisierung der kartesischen Karte aus Abbildung C.2

```
1 <?xml version="1.0" encoding="ASCII"?>
```

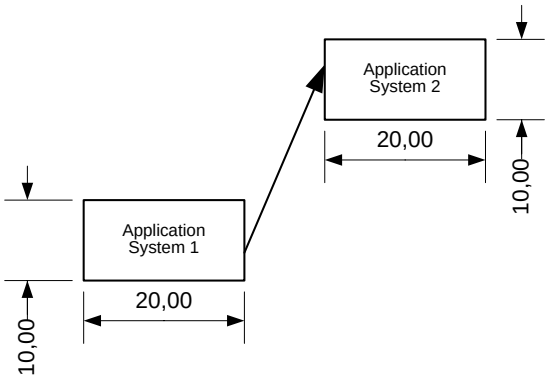


Abbildung C.3.: Graphlayoutkarte zur Serialisierung in Abbildung C.3

```
2 <SoCaVisualizationModel:SoftwareMap xmi:version="2.0" xmlns:xmi="http
      ://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance" xmlns:ConstraintVisualizationRules="http://www.
      softwarekartographie.de/visModel1.1/SoCaVisualizationModel/
      ConstraintVisualizationRules/" xmlns:MapSymbols="http://www.
      softwarekartographie.de/visModel1.1/SoCaVisualizationModel/
      MapSymbols/" xmlns:SoCaVisualizationModel="http://www.
      softwarekartographie.de/visModel1.1/SoCaVisualizationModel/"
      title="Graph Layout Map" creationDate="2007-04-24T17
      :26:37.828+0200" creator="A. Wittenburg">
3   <diagram>
4     <element xsi:type="MapSymbols:Rectangle" identityRule="21" id="6"
            attachmentRule="18" separationRule="20" width="20.0" height
          ="10.0" textHorizontalAlignment="center"
          textVerticalAlignment="middle" borderStyle="solid">
5       <dockingPoints id="7" x="20.0" y="6.0"/>
6       <center id="3" x="10.0" y="5.0"/>
7       <borderColor id="5" red="0" green="0" blue="0"/>
8       <fillColor id="4" red="255" green="255" blue="255"/>
9       <text id="2" value="Application System 1" fontSize="6">
10        <color id="0" red="0" green="0" blue="0"/>
11        <fontType id="1" name="Arial"/>
12      </text>
13    </element>
14    <element xsi:type="MapSymbols:Rectangle" identityRule="21" id
          ="14" attachmentRule="19" separationRule="20" width="20.0"
          height="10.0" textHorizontalAlignment="center"
          textVerticalAlignment="middle" borderStyle="solid">
15      <dockingPoints id="15" x="30.0" y="12.0"/>
16      <center id="11" x="20.0" y="6.0"/>
17      <borderColor id="13" red="0" green="0" blue="0"/>
18      <fillColor id="12" red="255" green="255" blue="255"/>
19      <text id="10" value="Application System 2" fontSize="6">
20        <color id="8" red="0" green="0" blue="0"/>
21        <fontType id="9" name="Arial"/>
```

```
22        </text>
23      </element>
24      <element xsi:type="MapSymbols:Polyline" id="17" attachmentRule
            ="18 19" lineStyle="solid">
25        <lineColor id="16" red="0" green="0" blue="0"/>
26        <arrowHeads>none</arrowHeads>
27        <arrowHeads>closedHeadedArrow</arrowHeads>
28      </element>
29      <element xsi:type="ConstraintVisualizationRules:Attachment" id
            ="18" attached="6 17" dockingPoints="7"/>
30      <element xsi:type="ConstraintVisualizationRules:Attachment" id
            ="19" attached="14 17" dockingPoints="15"/>
31      <element xsi:type="ConstraintVisualizationRules:Separation" id
            ="20" separated="6 14"/>
32      <element xsi:type="ConstraintVisualizationRules:
            IdentityOfProjection" id="21" ruledIdentical="6 14">
33        <constrainedAttribute>width</constrainedAttribute>
34        <constrainedAttribute>height</constrainedAttribute>
35        <constrainedAttribute>fillColor</constrainedAttribute>
36        <constrainedAttribute>borderColor</constrainedAttribute>
37      </element>
38    </diagram>
39    <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#201" visualizationelement="6"/>
40    <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#202" visualizationelement="14"/>
41    <semanticbridge uri="svn://www.softwarekartographie.de/
          exemplaryClustermap.xmi#401" visualizationelement="17"/>
42  </SoCaVisualizationModel:SoftwareMap>
```

Quelltext C.3: Serialisierung der Graphlayoutkarte aus Abbildung C.3

Literaturverzeichnis

[AIS77] Alexander, C.; Ishikawa, S.; Silverstein, M.: *A Pattern Language*. Oxford University Press, New York, 1977.

[al06] alfabet: *Introduction to planningIT*. White Paper, alfabet AG, 2006.

[An03] Andrews, T. et al.: *Business Process Execution Language for Web Services Version 1.1*. BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems, 2003. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf` (abgerufen am 2007-06-04).

[AT06] ATLAS group: *ATL: Atlas Transformation Language*. ATLAS group at LINA & INRIA, Nantes, 2006. `http://www.eclipse.org/gmt/atl/doc/ATL_User_Manual[v0.7].pdf` (abgerufen am 2007-03-13).

[Ba98] Balzert, H.: *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum, Heidelberg, Berlin, 1998. ISBN 3-8274-0065-1.

[BBB03] Bernhard, M.; Blomer, R.; Bonn, J. (Hrsg.): *Strategisches IT-Management Band 1*. Symposion, Düsseldorf, 2003. ISBN 3-936608-34-2.

[BCK03] Bass, L.; Clements, P.; Kazman, R.: *Software Architecture in Practice*. 2. Auflage, Addison Wesley, Boston, 2003. ISBN 0-321-15495-9.

[BCR06] Broy, M.; Cengarle, M.; Rumpe, B.: *Semantics of UML: Towards a System Model for UML – The Structural Data Model*. Technischer Bericht TUM-I0612, Technische Universität München, Institut für Informatik, 2006. `http://www4.informatik.tu-muenchen.de/publ/papers/RBC_SystemModel.Part1.pdf` (abgerufen am 2007-06-28).

[Be04] Beyer, N.: *Kennzahlen zur Beschreibung von Anwendungslandschaften und ihre Visualisierung auf Softwarekarten.* Bachelor-Arbeit, Technische Universität München, Fakultät für Informatik, 2004.

[BES04] Buchta, D.; Eul, M.; Schult-Croonenberg, H.: *Strategisches IT-Managamenet.* Gabler, Wiesbaden, 2004. ISBN 3-409-12527-2.

[BLF05] Berners-Lee, T.; Fielding, R.; Masinter, L.: *Uniform Resource Identifier (URI): Generic Syntax.* RFC 3986, Internet Engineering Task Force, 2005. `http://www.rfc-editor.org/rfc/rfc3986.txt` (abgerufen am 2006-04-13).

[Br99] Brockhaus Enzyklopädie: *Bd. 14. Mag.-Mod.* 18. Auflage, F.A. Brockhaus. 1999. ISBN 3-7653-1114-6.

[BM03] Braun, P.; Marschall, F.: *BOTL – The Bidirectional Object Oriented Transformation Language.* Technischer Bericht TUM-I0307, Technische Universität München, Institut für Informatik, 2003. `http://wwwbroy.in.tum.de/publ/papers/TUM-I0307.pdf` (abgerufen am 2007-06-28).

[Br05a] Brendebach, K.: *Integrierte Modelle und Sichten für das IT-Management.* Diplomarbeit, Technische Universität München, Fakultät für Informatik, 2005.

[Br05b] Brewer, C: *ColorBrewer: Instructions.* 2005. `http://www.personal.psu.edu/cab38/ColorBrewer/ColorBrewer_instructions.html` (abgerufen am 2007-04-24).

[BS96] Becker, J.; Schütte, R.: *Handelsinformationssysteme.* Verlag Moderne Industrie, Landsberg/Lech, 1996. ISBN 3-478-39820-7.

[Bu96] Buschmann, F. et al.: *Pattern-oriented Software Architecture: A System of Patterns.* Springer, John Wiles & Sons, 1996. ISBN 0-471-95869-7.

[Bu04] Budinsky, F. et al.: *Eclipse Modeling Framework.* Addison Wesley, Boston et al., 2004. ISBN 0-13-142542-0.

[Bu05] Buckl, S.: *Modell-basierte Transformationen von Informationsmodellen zum Management von Anwendungslandschaften.* Diplomarbeit, Technische Universität München, Fakultät für Informatik, 2005.

[Bu07] Buckl, S. et al.: *A Pattern based Approach for constructing Enterprise Architecture Management Information Models.* In (Oberweis, A. et al. Hrsg.): *8. Internationale Tagung Wirtschaftsinformatik*, Band 2, S. 145–162, Universitätsverlag Karlsruhe, Karlsruhe, 2007.

[BW05] Braun, C.; Winter, R.: *A Comprehensive Enterprise Architecture Metamodel and Its Implementation Using a Metamodeling Platform.* In (Desel, J.; Frank, U. Hrsg.): *Enterprise Modelling and Information Systems Architectures.* LNI P-75, S. 64–79, Köllen Druck+Verlag, Klagenfurt, 2005.

[Ch76] Chen, P.: *The Entity-Relationship Model – Toward a Unified View of Data.* In: *ACM Transactions on Database Systems (TODS)*, 1 (1), S. 9–36. 1976.

[Cl80] von Clausewitz, C.: *Vom Kriege*. 19. Auflage. Dümmler, Bonn, 1980. ISBN 3-427-82019-X.

[Cl02] Clements, P. et al.: *Documenting Software Architectures: Views and Beyond*. Addison Wesley, Boston, 2002. ISBN 0-2017-0372-6.

[De06] Dern, G.: *Management von IT-Architekturen*. 2. Auflage. Vieweg, Wiesbaden, 2006. ISBN 978-3-528-15816-3.

[DM05] DMTF: *Common Information Model (CIM) Infrastructure Specification – Version 2.3 Final*. Distributed Management Task Force, 2005. `http://www.dmtf.org/standards/published_documents/DSP0004V2.3_final.pdf` (abgerufen am 2007-03-27).

[DM06a] DMTF: *Common Information Model (CIM) – Application Model Specification Version 2.13*. Distributed Management Task Force, 2006. `http://www.dmtf.org/standards/cim/cim_schema_v214/CIM_Application.pdf` (abgerufen am 2007-03-28).

[DM06b] DMTF: *Common Information Model (CIM) – System Specification Version 2.14*. Distributed Management Task Force, 2006. `http://www.dmtf.org/standards/cim/cim_schema_v214/CIM_System.pdf` (abgerufen am 2007-03-27).

[Do04] DoD Architecture Framework Working Group: *DoD Architecture Framework Version 1.0 - Volume I: Definitions and Guidelines*. Department of Defense, United States of America, 2004. `http://www.dod.mil/nii/doc/DoDAF_v1_Volume_I.pdf` (abgerufen am 2006-08-10).

[Do05] ter Doerst, H. et al.: *Tool Support*. In (Marc., L. Hrsg.): *Enterprise Architecture at Work*. Springer, Berlin, Heidelberg, New York, 2005. ISBN-13 978-3-540-24371-7.

[ELW05] Ernst, A.; Lankes, J.; Wittenburg, A.:. *Werkzeuge für das Architekturmanagement großer IT-Landschaften*. In: *is report*, 2005 (12), OXYGON Verlag, 2005.

[ELW06] Ernst, A.; Lankes, J.; Wittenburg, A.:. *Tool Support for Enterprise Architecture Management - Strengths and Weaknesses*. In: *The Tenth IEEE International EDOC Conference (EDOC 2006)*, S. 13–22, IEEE, Hong Kong, China. 2006.

[En06a] Encyclopædia Britannica: *Architecture*. Encyclopædia Britannica Online, 2006. `http://www.search.eb.com/eb/article-9110410` (abgerufen am 2006-10-25).

[En06b] Encyclopædia Britannica: *Taylor, Frederick W.* Encyclopædia Britannica Online, 2006. `http://www.search.eb.com/eb/article-9071464` (abgerufen am 2006-03-16).

[Er06] Ernst, A. et al.: *Using Model Transformation for Generating Visualizations from Repository ContentsSoftwarekarten.* Technischer Bericht TB0601, Technische Universität München, Chair for Informatics 19 (sebis), 2006. `http://wwwmatthes.in.tum.de/file/Publikationen/2006/Er06b/Er06b.pdf` (abgerufen 2007-03-02).

[Fa99] Fahrmeier, L. et al.: *Statistik - Der Weg zur Datenanalyse.* 2. Auflage, Springer, Berlin, Heidelberg, New York, 1999. ISBN 3-540-65053-9.

[FMW05] Fischer, F.; Matthes, F.; Wittenburg, A.: *Improving IT Management at the BMW Group by Integrating Existing IT Management Processes.* In: *The Ninth IEEE International EDOC Conference (EDOC 2005)*, S. 219–228, IEEE, Enschede, Niederlande, 2005.

[Fr95] Frank, U.: *MEMO: Eine werkzeuggestützte Methode zum integrierten Entwurf von Geschäftsprozessen und Informationssystemen.* In (König, W. Hrsg.): *Wirtschaftsinformatik '95. Wettbewerbsfähigkeit, Innovation, Wirtschaftlichkeit*, S. 67–82, Physica-Verlag, Heidelberg, 1995.

[Fr02] Frank, U.: *Multi-Perspective Enterprise Modeling (MEMO) - Conceptual Framework and Modeling Languages.* In: *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, S. 1258–1267. Band 3, IEEE, Honolulu, 2002.

[Ga95] Gamma, E. et al.: *Design Patterns – Elements of Reusable Object-Oriented Software.* Addison Wesley Longman, Reading, Massachusetts, 1995. ISBN 0-201-63361-2.

[Ga04] Gabler: *Gabler Wirtschaftslexikon K-R.* 16. Auflage, Gabler Verlag, 2004. ISBN 3-409-12993-6.

[GMW97] Garlan, D.; Monroe, R.; Wile, D.: *ACME: An Architecture Description Interchange Language.* In: *CASCON'97*, IBM Press, Toronto, Ontario, Canada, 1997.

[Gu04] Gutwenger, C. et al.: *GoVisual – A Diagramming Software for UML Class Diagrams.* In (Jünger, M.; Mutzel, P. Hrsg.): *Graph Drawing Software*, S. 257–278. Springer, Berlin, Heidelberg, New York, 2004. ISBN 3-540-00881-0.

[Ha04] Halbhuber, T.: *Entwicklung eines Informationsmodells für das IT-Management.* Diplomarbeit, Technische Universität München, Fakultät für Maschinenwesen, 2004.

[He07] Helbig, J.: *Wachsender Fokus auf IT-Effektivität.* Vortragsunterlagen zur Vorlesung Strategisches IT-Management, Technische Universität München, 2007-01-19.

[HGM02] Hake, G.; Grünreich, D.; Meng, L.: 8. Auflage, *Kartographie.* Walter de Gruyter, Berlin, New York, 2002. ISBN 3-11-016404-3.

[Hi99] Hilliard, R.: *Using the UML for Architectural Description.* In (France, R.; Rumpe, B. Hrsg.): *"UML" '99 - The Unified Modeling Language: Beyond the Standard, Second International Conference.* LNCS 1723, S. 32–48, Springer-Verlag Heidelberg, Fort Collins, CO, USA, 1999.

[Hi05a] Hitz, M. et al.:. *UML@Work.* dpunkt.verlag, Heidelberg, 2005. ISBN 3-89864-261-5.

[Hi05b] Hilliard, R.: *Re: Questions IEEE 1471.* E-Mail in der IEEE-1471-USERS-LIST, 2005-09-12.

[HMT02] Heberling, M.; Maier, C.; Tensi, T.:. *Visual Modelling and Managing the Software Architecture Landscape in a large Enterprise by an Extension of the UML.* In: *OOPSLA 2002*, Seattle, USA, 2002.

[HV04] HVB Systems: *Softwarearchitekturbebauungsplan.* Vortragsunterlagen, 2004-12-07.

[IB04] IBM: *IBM Patterns for e-Business.* IBM Corporation, 2004. `http://www-106.ibm.com/developerworks/patterns/` (abgerufen am 2005-03-03).

[IE98] IEEE: *IEEE Std 830-1998: Recommended Practice for Software Requirements Specifications.* The Institute of Electrical and Electronics Engineers, Inc., 1998. ISBN 0-7381-0332-2.

[IE00] IEEE: *IEEE Std 1471-2000: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.* The Institute of Electrical and Electronics Engineers, Inc., 2000.

[IG04] Iyer, B.; Gottlieb, R.: *The Four-Domain Architecture: An approach to support enterprise architecture design.* In: *IBM Systems Journal*, 43 (3), S. 587–596, 2004.

[IT05] IT Governance Institute: *CobiT 4.0 - Control Objectives for Information and related Technology.* Information Systems Audit and Control Association, Rolling Meadows, USA, 2005. ISBN 1-933284-37-4.

[Ja05] James, G.: *Magic Quadrant for Enterprise Architecture Tools, 4Q04.* Gartner, Inc., 2005.

[JLL06] Jonkers, H.; van Leeuwen, D.; Lankhorst, M.: *ArchiMate Visio Stencils.* Telematic Insituut, 2006. `https://doc.telin.nl/dscgi/ds.py/Get/File-32177/ArchiMate_Visio_stencils.zip` (abgerufen am 2006-02-21).

[Jo04] Jonkers, H. (Hrsg.): *Concepts for Architectural Description, v4.0.* Technischer Bericht, Telematica Instituut, 2004. `https://doc.telin.nl/dscgi/ds.py/Get/File-29421` (abgerufen am 2006-03-28).

[Jo05] Jonkers, H. et al.:. *A Language for Enterprise Modelling.* In (Marc., L. Hrsg.): *Enterprise Architecture at Work*, Springer, Berlin, Heidelberg, New York, 2005. ISBN-13 978-3-540-24371-7.

[Jo06] Jonkers, H. (Hrsg.): *Architecture Language Reference Manual, v4.1.* Technischer Bericht, Telematica Instituut, 2006. `https://doc.telin.nl/dscgi/ds.py/Get/File-31626` (abgerufen am 2006-03-28).

[JV87] Jessen, E.; Valk, R.: *Rechnernetze - Grundlagen der Modellbildung.* Springer, Berlin, Heidelberg et al., 1987. ISBN 3-540-16383-2.

[Kü03] Kütz, M.: *Kennzahlen in der IT.* dpunkt.verlag, Heidelberg, 2003. ISBN 3-89864-225-9.

[Kü06] Kühn, H.: *Methodenintegration im Business Engineering.* Dissertation, Universität Wien, Fakultät für Wirtschaftswissenschaften und Informatik, 2006.

[Ka00] Kaschek, R.: *Schwachstellen einer Analyse des Modellbegriffs.* In: *EMISA FORUM*, 2000 (1), 2000.

[Ke02] Keller, W.: *Enterprise Application Integration.* dpunkt.verlag, Heidelberg, 2002. ISBN 3-89864-186-4.

[Ke06] Keller, W.: *IT-Unternehmensarchitektur.* dpunkt.verlag, Heidelberg, 2006. ISBN 3-89864-419-7.

[Ki03] Kirchner, L.: *Eine Sprache für die Modellierung von IT-Landschaften: Anforderungen, Potentiale, zentrale Konzepte.* In (Sinz, E.; Plaha, M.; Neckel, P. Hrsg.): *Modellierung betrieblicher Informationssysteme – MobIS 2003*, LNI P-38, S. 69–86, Köllen Druck+Verlag, Bamberg, 2003.

[KN91] Kaplan, R. S.; Norton, D. P.: *The Balanced Scorecard -Measures That Drive Performance.* In: *Harvard Business Review*, 70, S. 71–79. 1991.

[KNS92] Keller, G.; Nüttgens, M.; Scheer, A.: *Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“.* Technischer Bericht Heft 89, Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes, 1992. `http://www.iwi.uni-sb.de/Download/iwihefte/heft89.pdf` (abgerufen am 2007-04-01).

[Kr95] Kruchten, P.: *Architectural Blueprints - The “4+1” View Model of Software Architecture.* In: *IEEE Software*, 12 (6), S. 42–50, 1995.

[Kr05] Krcmar, H.: *Informationsmanagement.* 4. Auflage, Springer, Berlin et al., 2005. ISBN 3-540-23015-7.

[KSH93] Kronlöff, K.; Sheehan, A.; Hallmann, M.: *The Concept of Method Integration.* In (Kronlöff, K. Hrsg.): *Method Integration*, John Wiles & Sons, West Sussex, 1993. ISBN 0-471-93555-7.

[La04] Lankhorst, M.: *ArchiMate Language Primer, version 1.0.* Technischer Bericht, Telematica Instituut, 2004. `https://doc.telin.nl/dscgi/ds.py/Get/File-43839/ArchiMate_Language_Primer.pdf` (abgerufen am 2006-08-09).

[La05a] Lankhorst, M.: *Introduction to Enterprise Architecture.* In (Lankhorst, M. Hrsg.): *Enterprise Architecture at Work*, Springer, Berlin, Heidelberg, New York, 2005. ISBN 3-540-23271-2.

[La05b] Lauschke, S.: *Softwarekartographie: Analyse und Darstellung der IT-Landschaft eines mittelständischen Unternehmens.* Bachelor-Arbeit, Technische Universität München, Fakultät für Informatik, 2005.

[La06] Lang, R.: *Topographische und Thematische Karte des Landes Baden-Württemberg.* Statistisches Landesamt Baden-Württemberg, 2006. `http://www.mapview.de/beispiele/bw/index.html` (abgerufen am 2007-01-05).

[La07] Lauschke, S.: *Automatische Generierung von Softwarekarten: Entwicklung eines Ansatzes zum Layout deklarativ beschriebener Visualisierungen.* Master's Thesis. Technische Universität München, Fakultät für Informatik, 2007.

[Le05] Lentrodt, A.: *planningIT eXchange – Keynote.* Vortragsunterlagen, BMW Group, planningIT eXchange, 2005-11-15.

[LM04] Luftmann, J.; McLean, E.: *Key Issues for IT Executives.* In: *MIS Quarterly Executive*, 3 (2), S. 90–104. 2004.

[LMW05] Lankes, J.; Matthes, F.; Wittenburg, A.: *Architekturbeschreibung von Anwendungslandschaften: Softwarekartographie und IEEE Std 1471-2000.* In (Liggesmeyer, P.; Pohl, K.; Goedicke, M. Hrsg.): *Software Engineering 2005*, LNI P-64, S. 43–54. Köllen Druck+Verlag, Essen, Germany, 2005.

[LMW06] Lankes, J.; Matthes, F.; Wittenburg, A.: *Exkurs Softwarekartographie.* In (Keller, W. Hrsg.): *IT-Unternehmensarchitektur*, dpunkt.verlag, Heidelberg, 2006. ISBN 3-89864-419-7.

[LRR03] Lichtenegger, R.; Rohloff, M.; Rosauer, B.: *Beschreibung von Unternehmensarchitekturen: Sichten und Abhängigkeiten am Beispiel der IT-Infrastrukturarchitektur.* In (Dittrich, K. et al. Hrsg.): *INFORMATIK 2003 – Innovative Informatikanwendungen*, LNI P-35, S. 426–434. Köllen Druck+Verlag, Frankfurt am Main, 2003.

[LSV00] Laartz, J.; Sonderegger, E.; Vinckier, J.: *The Paris guide to IT architecture.* In: *McKinsey Quarterly*, 2000 (3), S. 118–127. 2000.

[Lu05] Luftmann, J.: *Key Issues for IT Executives 2004.* In: *MIS Quarterly Executive*, 4 (2) S. 269–285. 2005.

[LW04] Langenberg, K.; Wegmann, A.: *Enterprise Architecture: What Aspects is Current Research Targeting?* Technischer Bericht IC/2004/77, Ecole Polytechnique Fédérale de Lausanne, Laboratory of Systemic Modeling, 2004. `http://ic2.epfl.ch/publications/documents/IC_TECH_REPORT_200477.pdf` (abgerufen 2006-10-30).

[Ma06a] Mark, D.; Rau, D.: *Splitting Demand from Supply in IT.* In: *The McKinsey Quarterly*, 2006. `http://www.mckinseyquarterly.com/article_page.aspx?ar=1849` (abgerufen am 2007-05-06).

[Ma06b] Maurer, D.; Büch, P.: *Vom Geschäftsprozessdesign zur Enterprise Architecture.* Expert Paper, IDS Scheer AG, 2006. http://www2.ids-scheer.com/sixcms/media.php/2646/ARIS_Expert_Paper_-_Enterprise_Architecture_Buech-Maurer_2006-05_de.pdf (abgerufen 2007-06-28).

[ME02] META Group: *Enterprise Architecture Desk Reference.* META Group, Inc., 2002.

[MW04a] Matthes, F.; Wittenburg, A.: *Softwarekarten zur Visualisierung von Anwendungslandschaften und ihren Aspekten – Eine Bestandsaufnahme.* Technischer Bericht TB0401, Technische Universität München, Lehrstuhl für Informatik 19 (sebis), 2004. http://wwwmatthes.in.tum.de/file/Publikationen/2004/MaWi04a/040326-MaWi-Statusbericht-Softwarekartographie.pdf (abgerufen 2007-02-04).

[MW04b] Matthes, F.; Wittenburg, A.:. *Softwarekartographie: Visualisierung von Anwendungslandschaften und ihrer Schnittstellen.* In (Dadam, P.; Reichert, M. Hrsg.): *Informatik 2004 – Informatik verbindet, 34. Jahrestagung der GI.* LNI P-51, S. 71–75, Köllen Druck+Verlag, Ulm, 2004.

[MW05] Macharzina, K.; Wolf, J.: *Unternehmensführung.* 5. Auflage, Gabler Verlag, Wiesbaden, 2005. ISBN 3-409-63150-X.

[Ni05] Niemann, K.: *Von der Unternehmensarchitektur zur IT-Governance.* Vieweg, Wiesbaden, 2005. ISBN 3-528-05856-9.

[Of06] Office of Management and Budget: *FEA Consolidated Reference Model Document Version 2.0.* U.S. Office of Management and Budget, 2006. http://www.whitehouse.gov/omb/egov/documents/FEA_CRM_v20_Final_June_2006.pdf (abgerufen am 2006-08-10).

[OGB03] von Oetinger, B.; von Ghyczy, T.; Bassford, C. (Hrsg.): *Clausewitz Strategie denken.* Deutscher Taschenbuch Verlag, München, 2003. ISBN 3-423-34033-9.

[OG00a] Office of Government Commerce (OGC): *ITIL - Service Delivery.* IT Infrastructure Library (ITIL), The Stationery Office, Norwich, UK, 2000. ISBN 0-11-330017-4.

[OG00b] Office of Government Commerce (OGC): *ITIL - Service Support.* IT Infrastructure Library (ITIL), The Stationery Office, Norwich, UK, 2000. ISBN 0-11-330015-8.

[OM02] OMG: *Meta Object Facility (MOF) Core Specification, Version 1.4, formal/02-04-03.* Object Management Group, 2002.

[OM03] OMG: *MDA Guide, version 1.0.1.* Object Management Group, 2003.

[OM05a] OMG: *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, ptc/05-11-01.* Object Management Group, 2005.

[OM05b] OMG: *MOF 2.0/XMI Mapping Specification, v2.1, formal/05-09-01.* Object Management Group, 2005.

[OM05c] OMG: *Unified Modeling Language: Infrastructure, version 2.0, formal/05-07-05.* Object Management Group, 2005.

[OM05d] OMG: *Unified Modeling Language: Superstructure, version 2.0, formal/05-07-04.* Object Management Group, 2005.

[OM06a] OMG: *Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification, dtc/06-02-01.* Object Management Group, 2006.

[OM06b] OMG: *Common Object Request Broker Architecture, ptc/06-05-01.* Object Management Group, 2006.

[OM06c] OMG: *IT Portfolio Management Facility Specification, dtc/06-05-01.* Object Management Group, 2006.

[OM06d] OMG: *Meta Object Facility (MOF) Core Specification, Version 2.0, formal/06-01-01.* Object Management Group, 2006.

[OM06e] OMG: *Object Constraint Language, version 2.0, formal/06-05-01.* Object Management Group, 2006.

[Pe04] Peyret, H.: *Getting Value From Enterprise Architecture Tools.* Forrester Research, Inc., 2004.

[Po85] Porter, M.: *Competitive Advantage.* The Free Press, New York, 1985. ISBN 0-02-925090-0.

[Pu01] Purchase, H. et al.: *Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study.* In (Eades, P.; Pattison, T. Hrsg.): *Australian Symposium on Information Visualisation.* Sydney, Australien, 2001.

[Ri05] Riihinen, J.: *Enterprise Architecture Tool Survey TU Munich.* E-Mail-Korrespondenz von Herrn Jaakko Riihinen (Director, Chief Enterprise Architect, Nokia, 2005-03-21.

[Ro95] Robinson, A. et al.:. *Elements of Cartography.* 6. Auflage, John Wiley & Sons, Inc., Hoboken, USA, 1995. ISBN 0-471-55579-7.

[Ro05a] Rohloff, M.: *Enterprise Architecture – Framework and Methodology for the Design of Architecture in the Large.* In (Bartmann, D. et al. Hrsg.): *Proceedings of the Thirteenth European Conference on Information Systems*, Regensburg, 2005.

[Ro05b] Ross, J.: *Forget Strategy: Focus IT on your Operating Model.* Technischer Bericht Volume 5, Number 3C, Center for Information Systems Research (CISR), MIT Sloan, 2005.

[Ro06] Ross, J.: *Maturity Matters: How Firms Generate Value from Enterprise Architecture.* Technischer Bericht Volume 4, Number 2b, Center for Information Systems Research (CISR), MIT Sloan, 2004 (rev. 2006).

[RWR06] Ross, J.; Weill, P.; Robertson, D.: *Enterprise Architecture as Strategy – Creating a Foundation for Business Execution.* Harvard Business School Press, Boston, Massachusetts, 2006. ISBN 978-1-591-39839-4.

[Sc95] Schmidt, D.: *Programming Language Semantics.* In (Tucker, A. Hrsg.): *Computer Science Handbook*, CRC Press LLC, Boca Raton, Florida, 2004. ISBN 1-58488-360-X.

[Sc96] Scheer, A.: *ARIS-House of Business Engineering.* Technischer Bericht Heft 133, Universität des Saarlandes, Instituts für Wirtschaftsinformatik (IWi), 1996. `http://www.iwi.uni-sb.de/Download/iwihefte/heft133.pdf` (abgerufen 2007-03-12).

[Sc99] Schuette, R.: *Zum Realitätsbezug von Informationsmodellen.* In: *EMISA FORUM*, 1999 (2), 1999.

[Sc00] Schütte, R.: *Realitätsbezug von Informationsmodellen - Eine Erwiderung auf Kritik. EMISA FORUM*, 2000 (2), 2000.

[Sc01] Scheer, A.: *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen.* 4. Auflage, Springer, Berlin, 2001. ISBN 3-540-41601-3.

[Sc02] Scheer, A.: *ARIS - Vom Geschäftsprozess zum Anwendungssystem.* 4. Auflage, Springer, Berlin, 2002. ISBN 3-540-65823-8.

[Sc06] Schweda, C.: *Architektur eines Visualisierungswerkzeugs für Anwendungslandschaften - Anforderungsanalyse und Realisierung von Kernkomponenten.* Diplomarbeit, Technische Universität München, Fakultät für Informatik, 2006.

[se05a] sebis: *Enterprise Architecture Management Tool Survey 2005.* Technische Universität München, Lehrstuhl für Informatik 19 (sebis), 2005.

[Se05b] Sekatzek, P.: *Visualisierung von IT-Bebauungsplänen in Form von Softwarekarten - Konzeption und prototypische Umsetzung.* Diplomarbeit, Technische Universität München, Fakultät für Informatik, 2005.

[Si05] Siemens CIO: *Methods and Tools IT Navigator Object Model, Version 2.14*, Siemens AG, 2005.

[Sl05] Slocum, T. et al.: *Thematic Cartography and Geographic Visualization.* 2. Auflage, Pearson Prentice Hall, Upper Saddle River, NJ 07458, 2005. ISBN 0-13-035123-7.

[Sm03] Smith, T.: *Vitruvius on Architecture.* The Monacelli Press, New York, 2003. ISBN 1-58093-127-8.

[St73] Stachowiak, H.: *Allgemeine Modelltheorie.* Springer, Wien, New York, 1973. ISBN 3-211-81106-0.

[St03] Stangler, A.: *Unternehmensweite Musterarchitekturen und Technologiestandards.* Diplomarbeit, Technische Universität München, Fakultät für Informatik, 2003.

[St05a] Starke, G.: *Effektive Software-Architekturen.* Hanser, München, Wien, 2005. ISBN 3-446-22846-2.

[St05b] Struck, K.: *IT Governance at DB with planningIT - Experiences in Change Management.* Vortragsunterlagen, Deutsche Bahn, planningIT eXchange, 2005-11-15.

[SU01] SUN Microsystems: *J2EE Patterns.* Sun Microsystems, Inc., 2001. `http://java.sun.com/developer/technicalArticles/J2EE/patterns/` (abgerufen am 2004-01-30).

[SZ92] Sowa, J.; Zachman, J.: *Extending and formalizing the framework for information systems architecture.* In: *IBM Systems Journal*, 31 (3), S. 590–616, 1992.

[Sz02] Szyperski, C.: *Component Software.* 2. Auflage, Addison Wesley, London, 2002. ISBN 0-201-74572-0.

[Te99] Teubner, R.: *Organisations- und Informationssystemgestaltung.* Deutscher Universitäts-Verlag, Wiesbaden, 1999. ISBN 3-8244-6951-0.

[Te05a] Telamatica Instituut:. *ArchiMate Quick Reference.* Telematica Instituut, 2005. `https://doc.telin.nl/dscgi/ds.py/Get/File-52048` (abgerufen am 2007-04-03).

[Te05b] Tensi, T.: *Anwendungslandschaft: Nachverfolgung von IT-Modellen.* In (Breu, R.; Matzner, T.; Nickl, F. Hrsg.): *Software-Engineering*, Oldenbourg, München, 2005. ISBN 3-486-57574-0.

[To05] van der Torre, L. et al.:. *Landscape Maps for Enterprise Architectures.* Technischer Bericht SEN-E0514, Centrum voor Wiskundeen Informatica, 2005. `http://ftp.cwi.nl/CWIreports/SEN/SEN-E0514.pdf` (abgerufen am 2007-06-28).

[To06] van der Torre, L. et al.: *Landscape Maps for Enterprise Architectures.* In (Dubois, E.; Pohl, K. Hrsg.): *CAiSE 2006*, LNCS 4001, S. 351–366, Springer, Luxemburg, 2006.

[TOG02] The Open Group: *TOGAF "Enterprise Edition" Version 8.1.* The Open Group, 2002. `http://www.opengroup.org/architecture/togaf8-doc/arch/` (abgerufen am 2006-08-10).

[Vö03] Vöhringer, J.; Wieschalla, M.; Kohlmaier, A.: *VIT Benutzerhandbuch, Version 1.0.* sd&m, 2003.

[We06] Weber, U.: *Enterprise Architecture Managment.* Vortragsunterlagen, Detecon Consulting, Vortrag Pressegespräch planningIT, 2006-02-15.

[WEK04] Wiese, R.; Eiglsperger, M.; Kaufmann, M.: *yFiles – Visualization and Automatic Layout of Graphs.* In (Jünger, M.; Mutzel, P. Hrsg.): *Graph Drawing Software*, S. 173–191, Springer, Berlin, Heidelberg, New York, 2004. ISBN 3-540-00881-0.

[WHS06] Wehrmann, A.; Heinrich, B.; Seifert, F.: *Quantitatives IT-Portfoliomanagement – Risiken von IT-Investitionen wertorientiert steuern.* In: *Wirtschaftsinformatik*, 48 (4), S. 234–245. 2006.

[Wi07] Wittenburg, A. et al.: *Building an integrated IT governance platform at the BMW Group.* In: *International Journal of Business Process Integration and Management*, 2 (3), 2007 (in Veröffentlichung).

[WK94] WKWI: *Profil der Wirtschaftsinformatik.* In: *Wirtschaftsinformatik*, 36 (1), S. 80–81. 1994.

[WR04] Weill, P.; Ross, J.: *IT Governance.* Harvard Business School Press, Boston, Massachusetts, 2004. ISBN 1-59139-253-5.

[Za76] Zachman, J.: *Conceptual graphs for a data base interface.* In: *IBM Journal of Research and Development*, 20 (4), S 336–357, 1976.

[Za87] Zachman, J.: *A framework for information systems architecture.* In: *IBM Systems Journal*, 26 (3), S. 276–292, 1987.

[Za96] Zachman, J.: *The Framework for Enterprise Architecture: Background, Description and Utility.* Zachman Institute for Framework Advancement (ZIFA), 1996.

[Za06] Zachman, J.: *Enterprise Architecture: The Issue of the Century.* Zachman Institute for Framework Advancement (ZIFA), 2006.