Lernziele

Diese Kurseinheit soll Sie mit der Konstruktion von betrieblichen Informationssystemen vertraut machen. Ausgehend von der Struktur von Informationssystemen lernen Sie Möglichkeiten zur Festlegung der Aufgabenstruktur sowie zur Festlegung der Struktur der personellen und maschinellen Aufgabenträger kennen. Der Werkzeugunterstützung bei der Konstruktion betrieblicher Informationssysteme wird Rechnung getragen, indem unterschiedliche Modellierungssprachen behandelt werden. Ein besonderer Schwerpunkt wird auch auf die Beschreibung der Möglichkeiten zur Realisierung von Kopplungsarchitekturen gelegt.

Nach dem Studium von Abschnitt 3.1 kennen Sie den Konstruktionsbegriff für betriebliche Informationssysteme. Sie sollen bestimmte Möglichkeiten zur Aufgabenstrukturierung kennengelernt haben.

Nachdem Sie Abschnitt 3.2 durchgearbeitet haben, werden Sie in der Lage sein, die Aufgabenstruktur von Informationssystemen modellieren zu können.

Die Beschäftigung mit den Inhalten von Abschnitt 3.3 soll Sie zu einem Verständnis der Probleme bei der Modellierung der Aufbauorganisation eines Unternehmens befähigen.

Das Studium von Abschnitt 3.4 soll Sie mit der unter anderem für Webservices wesentlichen Extensible-Markup-Language vertraut machen bzw. bereits vorhandene Kenntnisse aktivieren. Sie werden in diesem Abschnitt ebenfalls in die Lage versetzt, ereignisgesteuerte Prozessketten zur Geschäftsprozessmodellierung einzusetzen. Sie wiederholen bzw. erwerben Kenntnisse bezüglich grundlegender Elemente der Unified-Modeling-Language (UML).

Nach dem Studium von Abschnitt 3.5 sollen Sie sicher in der Gestaltung von Kopplungsarchitekturen für betriebliche Anwendungssysteme sein.

Sichere Kenntnisse zur Konstruktion von betrieblichen Informationssystemen werden Sie nur erwerben, wenn Sie sich auch aktiv und selbständig mit den Übungsaufgaben auseinandersetzen. Die beigefügten Lösungen erlauben Ihnen eine Kontrolle des erreichten Wissensstandes.

3.1 Begriffsbestimmungen

Konstruktion

Wir gehen vom gewünschten Verhalten eines noch nicht existierenden Systems aus. Das Sachziel der Konstruktion besteht darin, eine Struktur für das zu konstruierende System zu finden, die das gewünschte Systemverhalten realisiert [18]. Typischerweise existieren mehrere unterschiedliche Systemstrukturen, die zum erwarteten Systemverhalten führen. Vorgaben in Form von qualitativen Anforderungen an die Konstruktion sowie Vorgaben bezüglich zu verwendender Komponenten werden als Formalziele einer Konstruktion bezeichnet. Durch die durchzusetzenden Formalziele einer Konstruktion wird die Menge der prinzipiell möglichen Strukturen eingeschränkt [18].

Informationssysteme unterstützen die Lösung betrieblicher Probleme. Bei der Konstruktion betrieblicher Informationssysteme wird typischerweise das gewünschte Systemverhalten durch funktionale Anforderungen an das zu konstruierende Informationssystem beschrieben. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.1.1 (Gesamtaufgabe einer Halbleiterfabrik) *Die Gesamtaufgabe einer Halbleiterfabrik besteht darin, integrierte Schaltkreise aufgrund vorgegebener Kundenspezifikationen zu bauen. Aus dieser Aufgabe lassen sich Anforderungen an ein zu konstruierendes Informationssystem für Halbleiterfabriken ableiten.*

Aufgaben

In Abschnitt 1.1.8 dieses Kurses wurde bereits erläutert, dass Informationssysteme der Lösung betrieblicher Aufgaben dienen. Diesen Aufgaben werden personelle und maschinelle Aufgabenträger zugeordnet. Die Aufgabenträger kommunizieren, um die Aufgaben kooperativ zu lösen. Als Ergebnis der Konstruktion eines Informationssystems erhalten wir die Feinstruktur des Informationssystems. Diese beschreibt, aus welchen Instanzen von Aufgaben und Aufgabenträgern das Informationssystem besteht sowie welche Beziehungen zwischen den identifizierten Instanzen existieren. Festlegungen bezüglich

- Aufgabenstruktur,
- Struktur der personellen Aufgabenträger,
- Struktur der maschinellen Aufgabenträger

müssen getroffen werden [18]. Wir erläutern nun der Reihe nach die einzelnen Festlegungen. Die Festlegung der **Aufgabenstruktur** besteht im Wesentlichen darin, ausgehend von der Gesamtaufgabe des Informationssystems diese sukzessive in Teilaufgaben zu zerlegen. Neben der Identifizierung der einzelnen Teilaufgaben ist es notwendig, Informationsbeziehungen festzulegen, auf deren Basis die erhaltenen Lösungen für die Teilaufgaben zur Lösung der Gesamtaufgabe zusammengesetzt werden können. Wir betrachten das nachfolgende Beispiel für die Aufgabendekomposition.

Beispiel 3.1.2 (Zerlegung einer Aufgabe in Teilaufgaben) *Die in Beispiel 3.1.1 beschriebene Gesamtaufgabe kann in die Teilaufgaben „Waferfertigung“ und „Montage/Test“ zerlegt werden. Nach der Verrichtung der Teilaufgabe „Waferfertigung“ liegen integrierte Schaltkreise auf Siliziumscheiben vor. Nach der Durchführung der Teilaufgabe „Montage/Test“ erhalten wir funktionsfähige, mit einem Plastikgehäuse versehene integrierte Schaltkreise.*

Eine Aufgabe wird wie bereits in Abschnitt 1.2 als vollautomatisiert bezeichnet, wenn sie von maschinellen Aufgabenträgern, d.h. Anwendungssystemen, ausgeführt wird. Sie heißt teilautomatisiert, wenn sie gemeinsam von personellen und maschinellen Aufgabenträgern gelöst wird. In diesem Zusammenhang wird von rechnergestützten Informationssystemen gesprochen. Schließlich wird eine Aufgabe nicht-automatisiert genannt, wenn sie ausschließlich von personellen Aufgabenträgern ausgeführt wird.

personelle Aufgabenträger

Die Struktur der personellen Aufgabenträger wird durch die Aufbauorganisation festgelegt. Die atomaren Einheiten der Aufbauorganisation sind Stellen. Eine Stelle wird mit genau einer Person besetzt. Unterschiedliche Stellen werden zu größeren organisatorischen Einheiten wie Gruppen oder Abteilungen zusammengefasst. Zwischen den unterschiedlichen Organisationseinheiten müssen Kommunikationsbeziehungen etabliert werden. Die Kommunikationsbeziehungen zwischen personellen Aufgabenträgern dienen der Umsetzung von Informationsbeziehungen zwischen Teilaufgaben und sind aus diesem Grund mit diesen abzugleichen.

maschinelle Aufgabenträger

Die Struktur der maschinellen Aufgabenträger wird durch die Beschreibung der Anwendungssysteme des Unternehmens und ihrer Kommunikationsbeziehungen untereinander festgelegt. Die angewandte Integrationsform bestimmt wesentlich die Art und Weise, in der die Kommunikation umgesetzt wird. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.1.3 (Realisierung von Kommunikation) *Im Falle einer Datenintegration kommunizieren zwei Anwendungssysteme über gemeinsam benutzte Datenbestände (vergleiche hierzu auch die datenorientierten Kopplungsarchitekturen aus Abschnitt 3.5 dieser Kurseinheit). Objektorientierte Anwendungssysteme kommunizieren untereinander durch Austausch von Nachrichten. Das Konzept der Datenintegration wird somit durch Objekt-Integration (vergleiche hierzu die Ausführungen in 2.3.1.6) erweitert.*

Für die Identifizierung von maschinellen Aufgabenträgern ist es notwendig, die Aufgaben soweit zu zerlegen, dass der Automatisierungsgrad der im Zuge der Dekomposition entstehenden Teilaufgaben bestimmt werden kann. Da im Allgemeinen keine vollständige Automatisierung der Informationsverarbeitung erreicht werden kann, ist es notwendig, die **Mensch-Maschine-Kommunikation** geeignet zu gestalten.

Entsprechend der Unterteilung in Aufgaben- und Aufgabenträgerebene beschäftigen wir uns im nächsten Abschnitt zunächst mit der Modellierung der

Aufgabenstruktur. In Abschnitt 3.3 gehen wir dann auf Möglichkeiten zur Modellierung der Aufbauorganisation ein. Anwendungssysteme als Repräsentanten maschineller Aufgabenträger werden detailliert in Kurseinheit 5 untersucht.

3.2 Modellierung der Aufgabenstruktur von Informationssystemen

Elemente einer Aufgabe

Eine betriebliche Aufgabe umfasst ein Aufgabenobjekt (AO), Aufgabenziele sowie Vor- und Nachereignisse . Am Aufgabenobjekt wird die Verrichtung der Aufgabe durchgeführt. Vorereignisse lösen die Verrichtung einer betrieblichen Aufgabe aus, während Nachereignisse das Ergebnis der Durchführung einer Aufgabe bilden. Durch Aufgabenobjekt, Aufgabenziel sowie die Vor- und Nachbedingungen wird die Außensicht auf eine betriebliche Aufgabe hergestellt.

Typ vs. Instanz

Im weiteren Verlauf dieser Kurseinheit wird bezüglich Aufgabenobjekten zwischen Typ und Instanz unterschieden. Der AO-Typ beschreibt die zu einer Aufgabe zugehörigen Attribute eines betrieblichen Systems. Ähnlich wie in der objektorientierten Welt werden AO-Instanzen durch konkrete Attributwerte beschrieben. Als Beispielaufgabe betrachten wir die Terminierung eines Loses.

Beispiel 3.2.1 (AO-Typ und -Instanz bei der Terminierung von Losen) *Der AO-Typ ist in diesem Fall durch die Attribute*

- *geplanter Fertigstellungstermin* d_j,
- *frühest möglicher Starttermin* r_j,
- *Priorität des Loses* w_j,
- *Produkt* P_j

gekennzeichnet. Eine AO-Instanz kann in diesem Fall wie folgt aussehen:

- $d_j = 50$,
- $r_j = 2$,
- $w_j = 10$,
- $P_j =$ P1223.

Lösungsverfahren

Die Innensicht einer betrieblichen Aufgabe wird durch Lösungsverfahren gebildet, die während der Durchführung der Aufgabe auf die AO-Instanzen angewandt werden. Wir sehen, dass Lösungsverfahren zunächst abstrakt auf der Ebene der AO-Typen formuliert werden können. Dazu betrachten wir das nachfolgende Beispiel.

Beispiel 3.2.2 (Lösungsverfahren) *Ein Terminierungsverfahren kann unter Verwendung der Größen r_j, d_j, w_j sowie unter Benutzung des zum Produkt P_j gehörigen Arbeitsplans formuliert werden. Ein konkreter Plan kann allerdings nur unter Verwendung der entsprechenden Losinstanzen ermittelt werden.*

Ein weiteres Merkmal einer Aufgabe sind ein Raum und eine Zeitspanne, in denen die Aufgabe verrichtet wird.

Die einzelnen Elemente einer betrieblichen Aufgabe sind in Abbildung 3.1 angegeben. Wir weisen auf die Analogie zur objektorientierten Modellierung hin. Die Lösungsverfahren entsprechen den Methoden, das Aufgabenobjekt auf Instanzebene einem Objekt und auf Typebene einer Klasse im Sinne der Objektorientierung.

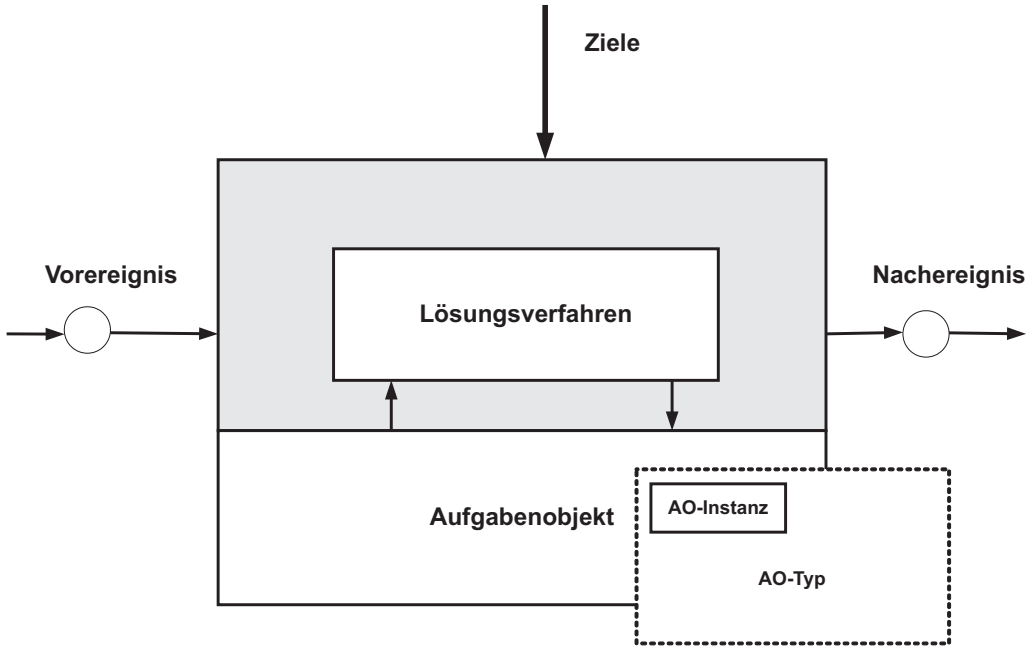

Abbildung 3.1: Elemente einer betrieblichen Aufgabe [17]

Übungsaufgabe 3.1 (Elemente betrieblicher Aufgaben) *Definieren Sie für die betriebliche Aufgabe der Erstellung eines Angebots für ein Produkt das Aufgabenobjekt, die Aufgabenziele, die Vor- und Nachbedingungen, den Aufgabenobjekt-Typ und die Aufgabenobjekt-Instanz. Beschreiben Sie ein mögliches Lösungsverfahren.*

Vier unterschiedliche Beziehungstypen sind zwischen den Aufgaben eines betrieblichen Informationssystems möglich [6, 17]: Beziehungstypen

1. Reihenfolgebeziehungen zwischen Aufgaben,

2. partielle Gleichheit von AO-Typen,
3. partielle Gleichheit von AO-Instanzen,
4. partielle Gleichheit von Lösungsverfahren.

Reihenfolgebeziehung

Von einer Reihenfolgebeziehung zwischen Aufgaben sprechen wir, wenn das Nachereignis einer Aufgabe A_1 gleichzeitig Vorereignis einer Aufgabe A_2 ist. Falls A_1 und A_2 unterschiedlichen Anwendungssystemen zugeordnet sind, müssen diese beiden Anwendungssysteme so gekoppelt werden, dass zwischen ihnen das entsprechende Ereignis übertragen werden kann. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.2.3 (Reihenfolgebeziehung zwischen Aufgaben) *Wir untersuchen die sukzessive Durchführung von Durchlaufterminierung und Kapazitätsabgleich [11]. Die Durchlaufterminierung dient der Bestimmung von Fertigungsterminen. Dabei werden typischerweise die zur Verfügung stehenden Kapazitäten nicht berücksichtigt. Das Ergebnis der Durchlaufterminierung sind Start- und Endtermine für die Arbeitsgänge der Lose. Die Start- und Endtermine können zur Berechnung der Kapazitätsbelastung verwendet werden. Der Kapazitätsabgleich wird zur Anpassung des Kapazitätsbedarfs an die zur Verfügung stehenden Kapazitäten durch Terminverschiebung verwendet. Wir betrachten die Durchlaufterminierung und den Kapazitätsabgleich als zwei Aufgaben, die sich in einer Reihenfolgebeziehung befinden. Nachereignis des Terminierungsaufgabenobjektes ist die Bereitstellung von Start- und Endterminen. Dieses Ereignis dient gleichzeitig als Vorereignis für das Kapazitätsabgleichaufgabenobjekt.*

partielle Gleichheit der AO-Typen

Falls die Lösungsverfahren für die unterschiedlichen Aufgaben A_1 und A_2 auf teilweise gleichen Attributmengen ihrer Aufgabenobjekte operieren, liegt eine partielle Gleichheit der AO-Typen vor. Falls A_1 und A_2 unterschiedlichen Anwendungssystemen zugeordnet sind, muss eine Kopplung dergestalt erfolgen, dass eine Abstimmung bezüglich der AO-Typen möglich ist. Wir betrachten das nachfolgende Beispiel zum besseren Verständnis der partiellen Gleichheit von AO-Typen.

Beispiel 3.2.4 (Partielle Gleichheit von AO-Typen) *Wir gehen von einem Anwendungssystem zur Feinplanung sowie einem zweiten Anwendungssystem zur Betriebsdatenerfassung aus. Die betrieblichen Aufgaben sind somit durch die Feinplanung und die Betriebsdatenerfassung gegeben. Die Feinplanung (vergleiche die Ausführungen in Abschnitt 6.1.3.3) beschäftigt sich mit der Zuteilung von Losen zu Maschinen und anschließend mit der Festlegung der Bearbeitungsreihenfolge der Lose auf den einzelnen Maschinen. Beide Anwendungssysteme müssen lesend auf Arbeitspläne zugreifen. Dabei ist zunächst nur die Struktur der Arbeitspläne entscheidend, die in bestimmten Tabellen einer relationalen Datenbank abgelegt sind. Eine Kopplung auf AO-Typebene bedeutet, dass beide Anwendungssysteme das gleiche Datenbankschema benutzen.*

partielle Identität von AO-Instanzen

Wenn die Lösungsverfahren nicht nur auf gleichen Attributmengen operieren, sondern zusätzlich auch noch die Attributwerte übereinstimmen, liegt eine partielle Identität von AO-Instanzen vor. Auch für diesen Beziehungstyp müssen im Falle von Aufgaben auf unterschiedlichen Anwendungssystemen diese so gekoppelt werden, dass die AO-Instanzen abgestimmt werden können. Wir zeigen das nachfolgende Beispiel für diesen Beziehungstyp.

Beispiel 3.2.5 (Partielle Identität von AO-Instanzen) *Wir betrachten ein Anwendungssystem zur Feinplanung und ein zweites Anwendungssystem zur Maschinenbelegung, im Folgenden als „Dispatchsystem“ bezeichnet. Sowohl Feinplanungs- als auch Dispatchsystem benötigen Informationen über diejenigen Lose, die vor einer bestimmten Maschinengruppe auf Bearbeitung warten. In diesem Fall liegt somit die partielle Gleichheit auf AO-Instanzebene vor.*

partielle Gleichheit von Lösungsverfahren

Die partielle Gleichheit von Lösungsverfahren ist als vierter Beziehungstyp zwischen Aufgaben möglich. Die partielle Gleichheit der Lösungsverfahren für die Aufgaben A_1 und A_2 führt zu gleichen Attributen für die unterschiedlichen AO-Typen. Gleiche Attribute bedeuten aber partiell identische AO-Typen. Wir zeigen dazu das nachfolgende Beispiel für diesen Beziehungstyp.

Beispiel 3.2.6 (Partielle Gleichheit von Lösungsverfahren) *Die gleichzeitige Nutzung eines Feinplanungsverfahrens in zwei unterschiedlichen Werken eines Unternehmens wird betrachtet. Das Feinplanungsverfahren wird auf einem eigenen Server betrieben. Die Manufacturing-Execution-Systems (MES) in den beiden Werken nutzen die Feinplanungsfunktionalität des Servers. Für die Darstellung der Funktionsweise von MES verweisen wir auf Abschnitt 6.1.3.3.*

Die vier eingeführten Beziehungstypen zwischen den Aufgaben eines Informationssystems werden in Abschnitt 3.5 zur Ausgestaltung von Kopplungsarchitekturen verwendet.

Übungsaufgabe 3.2 (Beziehungstypen zwischen Aufgaben) *Beschreiben Sie anhand eines selbst gewählten Beispiels die vier unterschiedlichen Beziehungstypen zwischen Aufgaben.*

Wir zählen an dieser Stelle nochmal zusammenfassend die wesentlichen Merkmale einer Aufgabe auf:

1. an Zielen orientierte Verrichtung,
2. ein Aufgabenobjekt zur Durchführung der Verrichtung,
3. Lösungsverfahren zur Verrichtung,
4. Raum und Zeitbezug der Verrichtung.

Aufgaben werden typischerweise nur in Form von Nachereignissen wahrgenommen. Aus diesem Grund bietet sich eine Aufgabenspezifikation in Form von Modellen an [18]. Dabei wird das in Abschnitt 1.1.3 beschriebene konstruktivistische Modellierungsverständnis zugrundegelegt. Eine Aufgabe wird dabei unter Berücksichtigung der Merkmale 1., 2., 3. spezifiziert. Die Aufgabe wird dann gemäß dieser Spezifikation unter Beachtung von Merkmal 4. durchgeführt.

Modellierungskonzepte

Die in Abschnitt 2.2.2 beschriebenen Modellierungskonzepte sind für die Beschreibung der Aufgabenstruktur von Informationssystemen nützlich. Durch die Modellierungskonzepte werden allerdings nicht alle Merkmale einer Aufgabe gleichermaßen abgedeckt, sondern lediglich Sichten auf ein oder mehrere Aufgabenmerkmale unterstützt. In Anlehnung an [18] wird zwischen den folgenden Sichten unterschieden:

- Funktionssicht,
- Datensicht,
- Interaktionssicht,
- Vorgangssicht.

Interaktionssicht

Funktions-, Daten- und Vorgangssicht sind bereits in Abschnitt 2.3.2 behandelt worden. Dabei entspricht die Vorgangssicht der Steuerungssicht in ARIS. Die hier neu hinzukommende Interaktionssicht wird wie folgt definiert.

Definition 3.2.1 (Interaktionssicht) *Die Interaktionssicht legt fest, wie einzelne Systembestandteile miteinander im zeitlichen Ablauf in Beziehung treten. Dabei wird insbesondere auch spezifiziert, welche Nachrichten zwischen den Teilfunktionen eines Informationssystems ausgetauscht werden.*

In Tabelle 3.1 stellen wir dar, welche Sicht durch welchen Modellierungsansatz unterstützt wird. Es ist klar, dass die Funktions- und die Datensicht einer statischen Sichtweise zuzuordnen sind, während die Interaktions- und Vorgangssicht einer dynamischen Sichtweise entsprechen.

Modellierung der Aufgabenstruktur

Wir beschreiben das Vorgehen bei der Modellierung der Aufgabenstruktur von Informationssystemen. Ein allgemein gültiges Vorgehensmodell ist dafür nicht vorhanden, sondern die vorhandene Problemstellung bestimmt im starken Maße das individuelle Vorgehen. Bei der Wahl des Vorgehens sind die nachfolgenden Punkte zu berücksichtigen [18]:

- Auswahl von geeigneten Modellierungsansätzen,
- Möglichkeit einer hierarchischen Dekomposition für die Modellierung,
- geeignete Abgrenzung des Modellierungsproblems,
- Berücksichtigung und Integration bereits verfügbarer Modellierungsergebnisse.

Die Auswahl eines geeigneten Modellierungsansatzes ist eine wichtige Entscheidung, die für den Erfolg bei der Implementierung eines Informationssystems entscheidend sein kann. Falls ein Modellierungsansatz mehrere Sichten umfasst, bestehen zwischen diesen Sichten häufig zeitliche Vorrangbeziehungen. Wir betrachten dazu das nachfolgende Beispiel.

Vorrangbeziehungen

Beispiel 3.2.7 (Vorrangbeziehungen für Sichten) *Bei objektorientierten Modellierungsansätzen wird nach einer groben Beschreibung der Funktionalität des Anwendungssystems unter Verwendung von Anwendungsfällen (Use-Cases) zunächst die statische Sichtweise in Form von Objektmodellen eingenommen, während dann im zweiten Schritt eine dynamische Sichtweise in Form von Interaktionsdiagrammen zugrunde gelegt wird.*

Innerhalb eines Architekturkonzepts werden häufig mehrere Modellierungsansätze verwendet. Diese stehen unter Umständen auch in einem Abhängigkeitsverhältnis. Wir betrachten dazu das folgende Beispiel.

Beispiel 3.2.8 (Abhängigkeit von Modellierungsansätzen) *Bei serviceorientierten Architekturen wurde in Abschnitt 2.3.4 erläutert, dass jeweils ein Prozess-, Dienste- und Technikmodell erstellt werden muss. Diese Modelle sind in dieser Reihenfolge zu entwickeln.*

Durch eine hierarchische Dekomposition zur Sichtenbeschreibung kann die Modellkomplexität reduziert werden. Wir unterscheiden zwischen einem Top-Down-Vorgehen, bei dem Komponenten sukzessive in Teilkomponenten zerlegt werden, und einem Bottom-Up-Vorgehen, bei dem Komponenten schrittweise zu größeren Komponenten zusammengefasst werden.

Tabelle 3.1: Sichten vs. Modellierungsansätze

Modellierungsansatz	unterstützte Sichten
Datenmodellierung	Datensicht
Funktionale Dekomposition	Funktionssicht
Strukturierte Analyse	Funktionssicht
	teilweise auch Datensicht
objektorientierte Modellierung	Datensicht
	Funktionssicht
	Interaktionssicht
agentenbasierte Modellierung	Datensicht
	Funktionssicht
	Interaktionssicht
Geschäftsprozessmodellierung	Vorgangssicht

Bezüglich der Abgrenzung des Modellierungsproblems besteht die Möglichkeit, die Aufgabenstruktur des gesamten Informationssystems abzubilden und erst in einem zweiten Schritt Teilmodelle zu identifizieren. Umgekehrt ist es möglich, zunächst Teilmodelle zu erstellen und diese dann in einem zweiten Schritt zu einem Gesamtmodell zusammenzufügen.

Eine Berücksichtigung und Integration bereits verfügbarer Modellierungsergebnisse kann in Form von Referenzmodellen erfolgen. Außerdem besteht häufig die Notwendigkeit, beim Wechsel der Modellierungsmethode bereits vorhandene Modelle als Eingangsgrößen zur Erzeugung neuer Modelle zu verwenden. Wir zeigen dazu das nachfolgende Beispiel.

Beispiel 3.2.9 (Weiterverwendung von Modellen) *Bei der objektorientierten Reimplementierung eines Anwendungssystems kann das ER-Modell als Startpunkt zur Erzeugung eines Objektmodells verwendet werden, da Verfahren existieren, die ER-Modelle in Objektmodelle überführen [13].*

3.3 Modellierung der Aufbauorganisation

Aufbauorganisation

Wie bereits in Abschnitt 1.1.6 dargestellt, gibt die Aufbauorganisation das Stellensystem der Organisation vor. Die Bildung von Stellen und die Regelung der Beziehungen zwischen den Stellen werden durch Methoden zur Gestaltung der Aufbauorganisation unterstützt.

Stellenbildung

Wir beschäftigen uns zunächst mit der Bildung von Stellen. Dazu werden Teilaufgaben zu einem Aufgabenkomplex zusammengefasst. Der Aufgabenkomplex wird dann im zweiten Schritt genau einer Person als Aufgabenträger zugewiesen. Eine Stelle entsteht durch die Bildung des jeweiligen Aufgabenkomplexes. Die Stelle wird mit einer konkreten Person, dem Stelleninhaber, besetzt.

Aufgabenkomplexe

Die Zusammenfassung von Teilaufgaben zu Aufgabenkomplexen kann nach verrichtungsorientierten Kriterien erfolgen [18]. In diesem Fall erhalten wir die bereits in Abschnitt 1.1.6 eingeführten funktionalen Organisationsstrukturen. Die Anwendung objektorientierter Kriterien bei der Bildung von Aufgabenkomplexen bedeutet, dass gleichartige Aufgabenobjekte zusammengefasst werden. Wir erhalten objektorientierte Organisationsstrukturen. Bei hybriden Organisationsstrukturen werden zunächst nach dem Objektprinzip Teilaufgaben, die zu einer Objektart gehören, zu Querschnittsaufgaben zusammengefasst. Die davon noch nicht betroffenen Teilaufgaben dienen der Koordination der unterschiedlichen Objekte. Nach verrichtungsorientierten Kriterien werden aus ihnen betriebliche Funktionen gebildet.

Beziehungen zwischen Stellen werden unterschieden in weisungsgebundene und weisungsungebundene Kommunikationsbeziehungen. Bei der ersten Art von Kommunikationsbeziehungen werden Aktivitäten menschlicher Aufgabenträger koordiniert. Dadurch wird das hierarchische Leitungssystem einer Organisation abgebildet [18]. Kommunikationsbeziehungen zwischen unterschiedlichen Stellen

werden durch die Informationsbeziehungen zwischen den durch die Stellen repräsentierten Aufgabenkomplexen verursacht.

Die Aufbauorganisation des Informationssystems erhält man aus denjenigen Stellen und Beziehungen der Aufbauorganisation des Unternehmens, die mit informationsverarbeitenden Tätigkeiten betraut sind bzw. geeignete Kommunikationskanäle für die technische Realisierung der Beziehungen zwischen den Teilaufgaben bereitstellen. Das Erreichen kurzer Informationswege sowie das Vermeiden von Medienbrüchen stehen bei der Gestaltung der Beziehungen zwischen den Stellen des Informationssystems im Vordergrund [18].

Das Vorgehen bei der Gestaltung der Aufbauorganisation wird entscheidend davon bestimmt, dass die Aufgabenzusammenfassung von der Aufgabenanalyse abhängt. Die Aufgabenanalyse und -zusammenfassung kann sich dabei auf die gesamte Organisation oder auf ausgewählte Teilbereiche der Organisation erstrecken. Die laufende Anpassung der Aufbauorganisation an geänderte Anforderungen und Rahmenbedingungen stellt eine ständige Aufgabe für die Unternehmensleitung dar. In Abschnitt 2.3.1.2 wurde deshalb gezeigt, dass die Unternehmenspläne und Geschäftsprozesse der ersten und zweiten Ebene der SOM-Unternehmensarchitektur die Aufbauorganisation des Unternehmens beeinflussen.

Methoden und Vorgehensmodelle sind typische Hilfsmittel zur Konstruktion von Informationssystemen. Häufig existieren Werkzeuge, die den Einsatz von Methoden und Vorgehensmodellen geeignet unterstützen. Für die Gestaltung der Aufbauorganisation kommen dabei Techniken zum Einsatz, die bereits bei der Beschreibung der Organisationssicht auf den unterschiedlichen Ebenen von ARIS in Abschnitt 2.3.2.1 dargestellt wurden.

3.4 Hilfsmittel für die Konstruktion

In diesem Abschnitt der Kurseinheit werden Hilfsmittel für die Konstruktion von Informationssystemen bereitgestellt. Bei der Auswahl der Hilfsmittel für die Konstruktion gehen wir zunächst von den in Abschnitt 2.2.2 beschriebenen Modellierungskonzepten aus.

Wir setzen voraus, dass bereits Kenntnisse in Datenmodellierung mit dem ERM vorliegen, deshalb führen wir an dieser Stelle zusätzlich in die Extensible-Markup-Language (XML) ein, die in modernen Informationssystemen zur Datenmodellierung genutzt wird und unter anderem im Kontext von Webservices eine große Rolle spielt. Anschließend werden ereignisgesteuerte Prozessketten (EPKs) dargestellt, die zur Geschäftsprozessmodellierung notwendig sind und in betriebswirtschaftlicher Standardsoftware eingesetzt werden.

Wir setzen im Folgenden voraus, dass Kenntnisse einer objektorientierten Sprache vorliegen. Aus diesem Grund verzichten wir auf eine Darstellung von Methoden der Funktionsmodellierung.

Wir beschreiben in diesem Abschnitt außerdem wesentliche Elemente der

Unified-Modeling-Language (UML) in einem Umfang, der für die Anwendung von UML in diesem Fernstudienkurs erforderlich erscheint.

3.4.1 Extensible-Markup-Language

XML

Wir definieren zunächst den Begriff XML in Anlehnung an [5].

Definition 3.4.1 (XML) *XML ist eine Sprache zur Definition von Auszeichnungssprachen bzw. Datenaustauschformaten. Sie gibt die Syntax der Auszeichnungssprache vor und stellt eine Teilmenge der Standard-Generalized-Markup-Language (SGML) dar.*

Elemente eines XML-Dokumentes

Ein XML-Dokument besteht aus den drei Teilen:

1. XML-Prolog,
2. Dokumenttyp-Definition (Document-Type-Definition (DTD)),
3. Dokumentinstanz.

Die ersten beiden Bestandteile sind optional, während eine Dokumentinstanz zwingend vorgeschrieben ist. Die Dokumentinstanz enthält den strukturierten Text. Die DTD dient zur Festlegung der erlaubten Struktur der Dokumentinstanz. Dadaurch wird eine Grammatik für XML-Dokumente vorgegeben. Neben DTD ist hier auch XML-Schema möglich (vergleiche die Ausführungen in Abschnitt 3.4.1.4). Eigenschaften des XML-Dokumentes wie die XML-Versionsnummer oder das Vorhandensein von internen oder externen Dokumenttyp-Definitionen werden durch den XML-Prolog festgelegt.

Wir beschreiben zunächst den syntaktischen Aufbau von Dokumentinstanzen im Detail, bevor wir Dokumenttyp-Definitionen erläutern.

3.4.1.1 Dokumentinstanzen

Elemente

Elemente dienen der Strukturierung von Dokumentinstanzen. Jede Dokumentinstanz besitzt genau ein Wurzelelement. Ein Element besteht aus einer Anfangs- und Endemarke. Anfangs- und Endemarke schließen den Elementinhalt ein. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.4.1 (Nutzung von Elementen in XML) *Wir zeigen die Abbildung eines Produktes in XML:*

```
<Product>
        XXCCD13
</Product>.
```

Durch die Schachtelung von Elementen erhalten XML-Dokumente eine hierarchische Struktur. Wir veranschaulichen das im folgenden Beispiel.

Beispiel 3.4.2 (Schachtelung von Elementen) *Wir betrachten die Modellierung eines Arbeitsganges. Dadurch wird festlegt, wie lange die Bearbeitung sowie der Belade- und Entladevorgang dauert und auf welchen Maschinen eine Bearbeitung möglich ist. Es gilt:*

<ProcessStep>
<ProcessStepID> S102A </ProcessStepID>
<ProcessingTime> 20 </ProcessingTime>
<Resource> M23 </Resource>
<AlternativeResource> M34 </AlternativeResource>
<LoadTime> 5 </LoadTime>
<UnloadTime> 3 </UnloadTime>
</ProcessStep>.

Attribute

Durch Attribute können Elemente näher beschrieben werden. Die Attribute folgen dabei in der Anfangsmarke unmittelbar dem Elementnamen. Die Reihenfolge der Attribute spielt keine Rolle. Ein Element kann somit nicht mehrere Attribute mit gleichem Namen enthalten. Wir betrachten das folgende Beispiel für die Nutzung von Attributen.

Beispiel 3.4.3 (Elementbeschreibung durch Attribute) *Wir bilden ein Produkt in XML ab. Durch das Attribut Type = „DC“ (als Abkürzung für* **D***istribution* **C***enter) wird angedeutet, dass das Produkt XXCCD13 auf Lager produziert wird. Wir erhalten:*

<Product Type= “DC“>
XXCCD13
</Product>.

Zeichenketten

In den bisherigen Beispielen wurden lediglich Zeichenketten und andere Elemente als Inhalt von Elementen verwendet. Eine **Zeichenkette** ist ein beliebiger Freitext, der die Zeichen <, > und & nicht enthalten darf. Neben Zeichenketten sind die nachfolgenden Bestandteile von Elementen möglich:

Bestandteile von Elementen

- **Kommentare**: Diese werden in der Form <!- - Kommentartext - -> dargestellt. Innerhalb von Kommentaren sind zwei aufeinanderfolgende Bindestriche nicht erlaubt, da diese als Kommentarende interpretiert werden.
- **Entity-Referenzen**: Entity-Referenzen stellen Verweise auf an anderer Stelle deklarierte Entitäten dar. Entitäten dienen zur Definition der physischen Struktur eines XML-Dokumentes, während Elemente und Attribute

die logische Struktur des Dokumentes bestimmen. Entitäten werden an späterer Stelle genauer erläutert.

- **Steueranweisungen (Processing-Instructions)(PI)**: Steueranweisungen sorgen dafür, dass ein XML-Prozessor Anweisungen an eine Anwendung weiterleitet, in deren Kontext er läuft.
- **Zeichenketten-Abschnitte (Character-Data-Section)**: Derartige Abschnitte werden als reine Zeichenketten interpretiert, obwohl sie XML-Markups enthalten.

Entities

Entities unterteilen sich in allgemeine Entities („general Entities“) und Parameter-Entities. Allgemeine Entities existieren wiederum in den folgenden Ausprägungen:

- Zeichen-Entities,
- Entities mit gemischtem Inhalt,
- ungeparste Entities.

Zeichen-Entities dienen zur Darstellung einzelner Zeichen, die in XML eine besondere Bedeutung haben oder nicht auf der Tastatur vorkommen. Dazu wird das nachfolgende Beispiel betrachtet.

Beispiel 3.4.4 (Zeichen-Entities) *Die XML-Sonderzeichen <, > und & sind spezielle Zeichen-Entities.*

Die in Beispiel 3.4.4 genannten Zeichen sind Beispiele für namensdefinierte Entities. Namensdefinierte Entities zeichnen sich dadurch aus, dass ein Name als Referenz auf das entsprechende Zeichen verwendet wird. Sie sind enweder direkt in XML vordefiniert oder müssen als spezielles DTD-Modul dem Dokument zur Verfügung gestellt werden.

Zahlendefinierte Entities stellen die zweite Klasse von Zeichen-Entities dar. Sie werden zum Zugriff auf Zeichen des Unicode-Zeichensatzes über die Positionsnummer des jeweiligen Zeichens verwendet.

Entities mit gemischtem Inhalt dienen der Einbindung von Inhalten unbeschränkter Länge. Dabei kann es sich sowohl um reine Zeichenketten als auch um Texte mit XML-Markups handeln. Es gibt sowohl interne als auch externe Entities mit gemischtem Inhalt. Bei externen Entities befindet sich der einzubindende Inhalt in einer anderen Datei.

Bei Zeichen-Entities und Entities mit gemischtem Inhalt ersetzt der XML-Parser in einem ersten Schritt die Referenz durch den dazugehörigen Wert und setzt in einem zweiten Schritt die Analyse vor der Referenz fort. Dadurch wird auch der referenzierte Text überprüft.

Dieses Vorgehen wird bei ungeparsten Entities aufgegeben. Ungeparste

Entities werden vom Parser ignoriert und ermöglichen so das Einbinden von Texten, die in anderen Dateiformaten vorliegen.

Mit Ausnahme von Zeichen-Entities werden alle Entities in DTD-Modulen deklariert. Die Deklaration von Entities wird deshalb erst in Abschnitt 3.4.1.2 beschrieben werden. Die Syntax für Entity-Referenzen sieht für namensdefinierte Entities wie folgt aus:

& Name der Entity;.

Bei zahlendefinierten Entities wird anstelle des Namens die entsprechende Dezimalzahl mit einem vorangestellten Doppelkreuz verwendet. Bei hexadezimalen Zahlen folgt dem Doppelkreuz das zusätzliches Symbol x. Wir betrachten zum besseren Verständnis das nachfolgende Beispiel.

Beispiel 3.4.5 (Referenzen auf XML-Sonderzeichen) *Das Sonderzeichen & wird als namensdefinierte Entity mit* & *angesprochen. Die entsprechende zahlendefinierte Entity lautet* &*. Das Zeichen & besitzt die Unicode-Nummer 38. Zum Zugriff auf das Zeichen < wird die namensdefiniert Entity* < *verwendet. Die dazugehörige zahlendefinierte Entity ist* >*.*

Falls Textabschnitte viele XML-Sonderzeichen beinhalten, die nicht als XML-Markup interpretiert werden sollen, können diese als Zeichenketten-Abschnitte definiert werden. Zeichenketten-Abschnitte werden in <![CDATA[und]]> eingeschlossen. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.4.6 (Zeichenketten-Abschnitte) *Der Textabschnitt* <resource>machine1</resource> *mit XML-Markups wird in einen Zeichenkettenabschnitt umgewandelt, damit er von einem XML-Parser nicht interpretiert wird:* <![CDATA[<resource>machine1</resource>]]>.

wohlgeformte Dokumente

Dokumente, die den in diesem Abschnitt dargestellten Regeln genügen, werden als **wohlgeformt** bezeichnet. XML-Dokumente, für die ein DTD-Modul existiert und die den dort enthaltenen Strukturierungsregeln genügen, heißen **gültig**.

Übungsaufgabe 3.3 (Dokumentinstanz) *Erstellen Sie unter Verwendung der nachfolgenden verbalen Beschreibung eines Auftrags eine XML-Dokumentinstanz.*
Ein Auftrag gliedert sich in die drei Gruppen Bestellposition, Lieferdaten und Anmerkungen. Zur Bestellposition gehören die Positionsnummer (1), der Produktname (P0815), die Menge (20) und die Einheit (Stück). Die Lieferdaten bestehen aus Informationen zum Kunden und dem Lieferdatum (20.07.2008). Zu den Kundendaten gehören der Name des Kunden (König & Partner) und dessen Anschrift, bestehend aus der Straße (Fleyerstraße), der Nummer (100), der Postleitzahl (58097) und dem Ort (Hagen). Die Anmerkungen enthalten einen Text, der auf die Dringlichkeit des Auftrags hinweist.

3.4.1.2 Definition von Dokumenttypen

DTDs

DTDs werden in XML zur Abbildung von Strukturierungsvorschriften in Form von Grammatiken verwendet. Die folgenden Objekte können in DTDs deklariert werden:

- Elementtypen,
- Attributlisten,
- Kommentare,
- Entities,
- Steueranweisungen,
- Notationen.

Element-deklarationen

Elementdeklarationen in XML beginnen mit <!ELEMENT. Ein Elementtyp verfügt über einen Namen. Ihm ist eines der fünf nachfolgenden Inhaltsmodelle zugeordnet:

- EMPTY,
- ANY,
- (#PCDATA),
- Kindelemente,
- gemischter Inhalt.

Instanzen eines mit dem Inhaltsmodell EMPTY versehenen Elementtyps sind entweder leere Elemente oder bei ihnen folgen Anfangs- und Endemarke unmittelbar aufeinander. Wir geben das nachfolgende Beispiel zum besseren Verständnis an.

Beispiel 3.4.7 (Elementtyp mit Inhaltsmodell EMPTY) <Product/> *ist eine gültige Instanz zu der Elementtypdeklaration* <!ELEMENT Product EMPTY>. <Product/> *ist ein leeres Element.* <Product></Product> *ist ein Dokument, bei dem Anfangs- und Endmarke direkt aufeinanderfolgen.*

Leere Elemente sind sinnvoll, wenn sie Attribute besitzen, in denen die relevanten Informationen abgelegt sind. Außerdem können Instanzen mit dem Inhaltsmodell EMPTY zur Markierung von bestimmten Stellen in einem Dokument (im Beispiel 3.4.7 Absätze) verwendet werden.

Elemente des Inhaltstyps ANY können hingegen beliebige Inhalte besitzen. Dabei gilt, dass der Inhalt wohlgeformt sein muss. Für alle vorkommenden Kindelemente müssen Elementtypen deklariert sein. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.4.8 (Elementtyp mit Inhaltsmodell ANY) *Die nachfolgende Deklaration* <!ELEMENT Machinery ANY> *ist ein Elementtyp mit Inhaltsmodell* ANY. *Elemente vom Elementtyp „Machinery“ dürfen somit beliebige andere Elemente enthalten, die in der DTD deklariert sind.*

Elemente mit dem Inhaltsmodell (#PCDATA) zeichnen sich dadurch aus, dass die Inhalte ihrer Instanzen reine Zeichenketten sind. Daraus folgt insbesondere, dass keine Unterelemente erlaubt sind. Wir zeigen dazu das nachfolgende Beispiel.

Beispiel 3.4.9 (Elementtyp mit Inhaltsmodell (#PCDATA)) *Die Deklaration* <!ELEMENT ProductName (#PCDATA)> *legt fest, dass die Inhalte der Instanzen reine Zeichenketten darstellen.*

Verschachtelung von Elementen

Eine beliebig tiefe Verschachtelung von Elementen wird durch das Inhaltsmodell „Kindelemente“ erlaubt. Für die Kindelemente $E_k, k = 1, \ldots, n$ eines Elements E gibt es die nachfolgenden Anordnungsmöglichkeiten:

- Sequenz $(E_1, \ldots, E_n)$,
- Alternative $(E_1 | \ldots | E_n)$,
- Wiederholung:
  - $0..n$: $E*$
  - $1..n$: $E+$
  - $0..1$: $E?$.

Alle Anordnungsmöglichkeiten können miteinander kombiniert werden. Wir zeigen das im nachfolgenden Beispiel.

Beispiel 3.4.10 (Sequenz) *Die folgende Deklaration für den Elementtyp ProcessStep aus Beispiel 3.4.2*

<!ELEMENT ProcessStep (ProcessStepID, ProcessingTime,
Resource+, AlternativeResource∗, LoadTime, UnloadTime) >

entspricht dem Inhaltsmodell „Kindelemente“ mit Sequenz. Ähnlich wie in Beispiel 3.4.2 stellt

<ProcessStep>
<ProcessStepID> S102A </ProcessStepID>
<ProcessingTime> 20 </ProcessingTime>
<Resource> M23 </Resource>
<Resource> M25 </Resource>
<AlternativeResource> M34 </AlternativeResource>

```
        <LoadTime> 5 </LoadTime>
        <UnloadTime> 3 </UnloadTime>
</ProcessStep>
```

eine Instanz des Elementtyps ProcessStep dar.

Die Reihenfolge der Kindelemente ist durch die Deklaration vorgegeben und kann nicht geändert werden. Lediglich Kommentare und Steueranweisungen können davor, danach oder zwischen den Kindelementen auftreten.

Inhaltsmodell „gemischter Inhalt“

Das Inhaltsmodell „gemischter Inhalt“ gestattet es, Text und Auszeichnungen zu mischen. In einer Dokumentinstanz können Texte und Auszeichnungen in beliebiger Reihenfolge und beliebig häufig auftreten. Zwischen Texten und Auszeichnungen können auch Zeichenketten stehen.

Beispiel 3.4.11 (Elementtyp mit Inhaltsmodell „gemischter Inhalt“) *Instanzen, die der Elementtypdeklaration*

```
<!ELEMENT text (# PCDATA | Hervorhebung | Bild | Tabelle)* >
```

genügen, gehören zum Inhaltsmodell „gemischter Inhalt“. Die Instanzen dieses Elementtyps enthalten einen Text, der an einigen Stellen als Hervorhebung gekennzeichnet ist und Bilder und Tabellen enthält.

In Beispiel 3.4.11 beginnt die Elementtypdefinition mit $\#PCDATA$ und endet mit einem *. Das ist zwingend für alle Elementtypdeklarationen vom Elementyp „gemischter Inhalt“ vorgeschrieben. $\#PCDATA$ darf nicht in Sequenzen oder Alternativen im Inhaltsmodell „Kindelemente“ verwendet werden.

Das nachfolgende Beispiel zeigt eine Kombination der verschiedenen Elementtypen für einen Arbeitsplan. Ein Arbeitsplan stellt eine Abfolge von Arbeitsgängen dar [11]. Jedem einzelnen Arbeitsgang werden die Maschinen, auf denen der Arbeitsgang ausgeführt werden kann, sowie die Bearbeitungszeiten zugeordnet. Außerdem werden Belade- und Entladezeiten aufgeführt.

Beispiel 3.4.12 (DTD für einen Arbeitplan) *Wir zeigen den nachfolgenden Ausschnitt aus der DTD für einen Arbeitsplan:*

```
<!ELEMENT Route (Description, FlowInformation, AdditionalInfo)>
<!ELEMENT Description (ProductName, Technology, Creator+, Facility,
CreationDate)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT Technology (#PCDATA)>
<!ELEMENT Creator (FirstName, SureName)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT SureName (#PCDATA)>
<!ELEMENT Facility (#PCDATA)>
```

<!ELEMENT CreationDate (#PCDATA)>
<!ELEMENT FlowInformation (ProcessStep+, SimilarFlows*)>
<!ELEMENT ProcessStep(...)> ...
<!ELEMENT SimilarFlows (RouteNames)>
<!ELEMENT RouteNames (#PCDATA)>
<!ELEMENT AdditionalInfo(Text)>
<!ELEMENT Text (#PCDATA | Table | Figure)*>.

Die Deklaration von ProcessStep wurde bereits in Beispiel 3.4.10 vorgenommen. Wir sehen, dass die angebenene DTD die Inhaltsmodelle „(#PCDATA)“, „Kindelemente“ und „gemischterInhalt“ beinhaltet.

Das nachfolgende Beispiel zeigt einen Ausschnitt einer Instanz zur DTD aus Beispiel 3.4.12.

Beispiel 3.4.13 (Instanz für die DTD „Arbeitsplan“) *Der folgende Ausschnitt der XML-Instanz des Arbeitplans für Produkt „ABCCCD1000“ wird betrachtet:*

<Route>
<Description>
<ProductName> ABCCCD1000 </ProductName>
<Technology> CMOS1002 </Technology>
<Creator>
<FirstName> Karl </FirstName>
<SureName> Sonntag </SureName>
</Creator>
<Creator>
<FirstName> John W. </FirstName>
<SureName> Barts </SureName>
</Creator>
<Facility> X-FAB 1 Erfurt </Facility>
<CreationDate> 2007.12.02</CreationDate>
</Description>
<FlowInformation>
<ProcessStep>
<ProcessStepID> S102A </ProcessStepID>
<ProcessingTime> 20 </ProcessingTime>
<Resource> M23 </Resource>
<AlternativeResource> M34 </AlternativeResource>
<LoadTime> 5 </LoadTime>
<UnloadTime> 3 </UnloadTime>
</ProcessStep> ...
<ProcessStep>

```
            <ProcessStepID> S103A </ProcessStepID>
            <ProcessingTime> 25 </ProcessingTime>
            <Resource> M13 </Resource>
            <Resource> M16 </Resource>
            <Resource> M17 </Resource>
            <LoadTime> 5 </LoadTime>
            <UnloadTime> 3 </UnloadTime>
        </ProcessStep>
        <SimilarFlows>
            <RouteNames>ABCCCD1000L</RouteNames>
        </SimilarFlows>
    </FlowInformation>
    <AdditionalInfo>
        <Text> This product leads to many problems. </Text>
    </AdditionalInfo>...  .
```

Übungsaufgabe 3.4 (Dokumenttyp-Definition) *Erstellen Sie für die in Aufgabe 3.3 erstellte XML-Dokumentinstanz eine Dokumenttyp-Definition. Beachten Sie hierbei, dass mehrere Bestellpositionen möglich sind und beim Kunden neben der Anschrift auch eine Lieferanschrift angegeben werden kann. Bei den Anschriften können sowohl eine Adresse, bestehend aus Straße und Hausnummer, als auch ein Postfach angegeben werden.*

Arten von XML-Dokumenten

Zwei Arten von XML-Dokumenten existieren. **Datenorientierte** XML-Dokumente besitzen eine weitestgehend reguläre Struktur mit wenig Freitext. Die zweite Klassen von Dokumenten, **dokumentenorientierte** XML-Dokumente, beinhalten überwiegend Freitext [5]. In Beispiel 3.4.13 ist der Description- und FlowInformation-Teil datenorientiert, während der AdditionalInfo-Teil aufgrund des gewählten Inhaltsmodells „gemischter Inhalt“ dokumentenorientiert ist. Datenorientierte XML-Dokumente bieten den Vorteil, dass Anfragen entsprechend ihrer Struktur möglich sind. Bei dokumentenorientierten XML-Dokumenten sind derartige Anfragen aufgrund des verwendeten Inhaltsmodells „gemischter Inhalt“ schwieriger aufzufinden.

Attributtypen

Wie bereits beschrieben, können Elementtypen mit Attributen versehen werden. Dazu ist es notwendig, dass im zum Elementtyp gehörigen Dokumenttyp eine Attributliste deklariert wird. Das Schlüsselwort <!ATTLIST eröffnet eine Attributlistendeklaration. Daran schließen sich die jeweiligen Attributbezeichner mit Attributtyp an. Die nachfolgenden Attributtypen existieren:

- CDATA,
- ID,
- IDREF,
- IDREFS,

- ENTITY,
- ENTITIES,
- NMTOKEN,
- NMTOKENS,
- NOTATION.

Der Typ CDATA ermöglicht es, Attribute zu verwenden, die durch eine Zeichenkette gegeben sind. Der Attributtyp ID wird zur eindeutigen Identifikation von Elementen verwendet. Werte dieses Typs müssen dokumentweit eindeutig sein. Attribute vom Typ ID verlangen eine der beiden Angaben „#REQUIRED“ oder „#IMPLIED“.

Die Attributbedingung „#REQUIRED“ führt zu Attributen, die bei jedem Element dieses Elementtyps auftreten müssen. Durch die Bedingung „#IMPLIED“ gekennzeichnete Attribute sind optional. Als dritte mögliche Attributbedingung sind Defaultwerte möglich. Der Defaultwert wird immer dann angewendet, wenn kein Wert für das jeweilige Attribut gesetzt wird. Der Typ des Attributes und des Defaultwerts müssen zusammenpassen. Falls der Angabe des Defaultwerts die Bedingung „#FIXED“ vorangestellt wird, kann das Attribut ausschließlich den Defaultwert annehmen.

Neben dem Attributtyp ID wird der Typ IDREF angewendet. Attribute vom Typ IDREF verweisen auf ein Attribut vom Typ ID. Ein Verweis auf eine Menge von Attributen vom Typ ID erfolgt durch den Typ IDREFS.

Attribute vom Typ ENTITY werden zum Verweis auf eine ungeparste Entity mit gleichem Namen verwendet. Eine Menge derartiger Verweise wird durch Attribute vom Typ ENTITIES verwaltet.

Der Attributtyp NTOKEN wird für Namenstoken verwendet. Namenstoken bestehen aus Buchstaben, Ziffern und den Zeichen $z \in \{., -, \_, :\}$. Die verwendeten Buchstaben und Ziffern müssen Bestandteil des Unicode-Zeichensatzes sein. Der Attributtyp NTOKENS ist eine Liste von Attributen vom Typ NTOKEN. Namenstoken werden zur Realisierung von Aufzählungen verwendet. Eine Aufzählung beginnt und endet jeweils mit einer runden Klammer. Die einzelnen Elemente der Aufzählung werden durch senkrechte Striche getrennt.

Der Attributtyp NOTATION wird für Attribute verwendet, denen als Wert eine der vorher in der DTD deklarierten Notationen zugewiesen wurde. Attribute vom Typ NOTATION können zur Interpretation des Elements verwendet werden, dem das Attribut zugeordnet ist.

Wir zeigen nun ein Beispiel für eine Attributdeklaration.

Beispiel 3.4.14 (Attributdeklaration) *Wir betrachten Attribute, die dem Elementtyp ProcessStep zugeordnet sind.*

<!ATTLIST ProcessStep

ProcessStepID	ID	#REQUIRED
ProcessingTime	CDATA	#IMPLIED
SetupRequired	(y|n)	„n“

>

Wir bemerken, dass für das Attribut „SetupRequired“, das festlegt, ob eine Umrüstzeit notwendig ist oder nicht, der Defaultwert „n“ (nein) verwendet wird.

In Beispiel 3.4.14 wird dem Attribut „SetupRequired“ eine Aufzählung zugeordnet. Die beiden Aufzählungselemente „n“ und „y“ sind vom Typ NTOKEN.

Attribute beschreiben Elementinhalte. Sie stellen somit Metainformationen dar. Elemente hingegen beinhalten Datenobjekte und deren Komponenten. Wir veranschaulichen diesen Sachverhalt an einem Beispiel.

Beispiel 3.4.15 (Elemente vs. Attribute, Teil 1) *Die Umrüstzeit wird als Element modelliert, während die Zeiteinheit ein Attribut ist:*

```
<SetupTime TimeUnit = “Hours“>
      1.5
</SetupTime>.
```

Attribute stellen in gewisser Weise Zusatzinformationen dar. Aus diesem Grund sind sie nicht notwendigerweise sichtbar. Sichtbarkeit bedeutet in diesem Zusammenhang, dass Daten nach der Umwandlung in ein anderes Dokumentenformat immer noch sichtbar sind. In Beispiel 3.4.15 kann es unter Umständen erforderlich sein, dass im Zieldokument alle Umrüstzeiten in einer einzigen Zeiteinheit angegeben werden. Das kann durch einen einfachen Umrechnungsschritt bei der Umwandlung des Ausgangs- in das Zieldokument erreicht werden. In diesem Fall ist es sinnvoll, die Zeiteinheit wie in Beispiel 3.4.15 als Attribut zu modellieren, da die Zeiteinheiten zwar verarbeitet, aber nicht übertragen werden müssen, also nicht mehr sichtbar sind.

Umgekehrt kann es erwünscht sein, die Umrüstzeiten mit der im Ausgangsdokument verwendeten Zeiteinheit in das Zieldokument zu übertragen. In diesem Fall sind die Zeiteinheiten ebenfalls als Element zu modellieren. Beispiel 3.4.15 ist dann wie folgt zu modifizieren.

Beispiel 3.4.16 (Elemente vs. Attribute, Teil 2) *Sowohl die Umrüstzeit als auch die verwendete Zeiteinheit werden als Element modelliert:*

```
<SetupTime>
      <TimeUnit>Hours</TimeUnit>
      <Value> 1.5 </Value>
</SetupTime>.
```

Gegen die Verwendung von Attributen spricht unter anderem die Tatsache, dass Attribute nicht mehrfach auftreten dürfen. Außerdem sind Attribute unstrukturiert. Für die Verwendung von Attributen in der Modellierung spricht die Tatsache, dass Attribute eine kompaktere Schreibweise als Elemente erlauben. Attribute ermöglichen eine bessere Abbildbarkeit von alternativen Bedingungen. Wenn wie in Beispiel 3.4.14 das Attribut „Setup“ mit den Aufzählungswerten „y“ und „n“ belegt werden kann, ist das leichter nachzuvollziehen als eine Abbildung durch ein optionales leeres Element. In vielen Fällen ist aber die Entscheidung, ob Daten als Elemente oder Attribute modelliert werden, eine Sache des Geschmacks des Modellierers.

Übungsaufgabe 3.5 (Attribute in DTDs) *Ändern Sie die DTD aus Aufgabe 3.4 so ab, dass Attribute verwendet werden können. Dafür eignet sich bei der Bestellposition die Mengenangabe. Die Einheit kann hier als Attribut modelliert werden. Weiterhin müssen die Positionsnummern eindeutig sein.*

Durch Entities wird die physische Struktur eines XML-Dokumentes bestimmt, während Elemente und Attribute die logische Struktur festlegen. Entities dienen der Textersetzung in XML-Dokumenten. Unter Textersetzung versteht man in diesem Kontext, dass die Referenz auf die Entität beim Parsen des Dokumentes durch den Inhalt der Entität ersetzt wird. Wie bereits dargestellt, unterscheiden wir zwischen allgemeinen und Parameter-Entities.

Allgemeine Entities können in Dokumentinstanzen referenziert, aber nur in DTDs deklariert werden. Im Gegensatz dazu können Parameter-Entities nur in DTDs referenziert werden. Parameter-Entities dienen dazu, Attribut- und Elementtypdeklarationen an einer zentralen Stelle zu ändern. Außerdem können externe DTDs durch Parameter-Entities in DTDs eingebunden werden. Parameter-Entities werden analog zu allgemeinen Entities referenziert, lediglich das führende & wird durch ein „%“ ersetzt.

Entity-Deklarationen werden durch das Schlüsselwort <!ENTITY eingeleitet. Nach diesem Schlüsselwort folgt der Namen der jeweiligen Entity. Bei Parameter-Entities geht dem Namen noch ein Prozentzeichen voraus. Auf den Namen folgt dann direkt der Wert der Entity. In diesem Fall spricht man von einer internen Entity. Alternativ kann anstelle des Wertes ein externer Identifikator verwendet werden. Dann spricht man von externen Entities.

Externe Entities können entweder durch einen URI oder durch Angabe eines Public-Identifiers (PublicID) identifiziert werden. Im ersten Fall wird das Schlüsselwort SYSTEM verwendet, während im zweiten Fall das Schlüsselwort PUBLIC vorangestellt wird.

Interne allgemeine Entities werden häufig als Platzhalter für mehrfach vorkommende Textfragmente verwendet. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.4.17 (Verwendung von internen allgemeinen Entities) *Die nachfolgenden Deklarationen können in einem Anschreiben eines Vertriebsmit-*

arbeiters an seine Kunden verwendet werden:

<!ENTITY Kunde „Herr Maier“>
<!ENTITY Firma „Automotive GmbH Esslingen“>
<!ENTITY Adresse „Max-Weber-Platz 1, 89076 Esslingen“>
<!ENTITY Produkt „CMOS10254“>
<!ENTITY Menge „10000 Stück“>
<!ENTITY Liefertermin „14.06.2008“>.

Diese Deklarationen können in der nachfolgenden Dokumentinstanz angewandt werden:

<Auftragsbestätigung>
<Anrede> Sehr geehrter &Kunde;,</Anrede>
<Text> hiermit bestätigen wir Ihnen Ihre Bestellung von &Menge; von Produkt &Produkt;.
Die Lieferung erfolgt an &Firma;, &Adresse;. Der Liefertermin ist der &Liefertermin;. Wir bedanken uns für das in uns gesetzte Vertrauen. </Text>.

Die Entity-Deklarationen in Beispiel 3.4.17 erhöhen die Wartbarkeit der XML-Dokumente, da notwendige Änderungen nur noch an einer zentralen Stelle vorgenommen werden müssen.

Externe Entities mit gemischtem Inhalt werden hingegen dazu verwendet, große Dokumentinstanzen auf mehrere Dateien aufzuteilen. Außerdem gestatten externe Entities die Einbindung von Textfragmenten in mehrere Dokumentinstanzen. Wir setzen diese Technik im nachfolgenden Beispiel ein.

Beispiel 3.4.18 (Externe Entities mit gemischtem Inhalt) *Es wird eine externe Entity zum Einbinden von Adressdaten für das Anschreiben aus Beispiel 3.4.17 verwendet:*

<! ENTITY Anschreiben-Kopf SYSTEM “adressdaten.xml“>.

Durch diese Deklaration wird auf eine Datei verwiesen, die lokal unter dem Namen „adressdaten.xml“ vorliegt. Der Inhalt der Datei muss dabei wohlgeformten XML-Text enthalten.

ungeparste Entities

Ungeparste Entities sind spezielle allgemeine Entities. Sie erlauben es, Dateien, die keinen XML-Text enthalten, in XML-Dokumente einzubinden. Ungeparste Entities werden zusammen mit Notationen eingesetzt, die beschreiben, wie mit dem Format der Datei umzugehen ist. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.4.19 (Ungeparste Entity) *Die nachfolgende Deklaration ermöglicht es, .bmp-Bilder einzubinden:*

<! ENTITY bild
SYSTEM “http://www.meine-bilder.com/bild.bmp“
NDATA bmp>.

Referenzen auf ungeparste Entities dürfen nur als Attributwerte von Elementen verwendet werden. Die in Beispiel 3.4.19 angegebene Deklaration von .bmp-Bildern verlangt deshalb zunächst die Deklaration des in Beispiel 3.4.20 angegebenen Entitytyps.

Beispiel 3.4.20 (Verwendung ungeparster Entities) *Der Elementtyp*

<!ELEMENT Figure EMPTY>
<!ATTLIST Figure
source ENTITY #REQUIRED>

bereitet die Verwendung von ungeparsten Entities als Attribute vor. Eine Instanz dieses Entity-Typs ist durch

<Figure source = “bild“>

gegeben.

Interne Parameter-Entities erleichtern genauso wie interne allgemeine Parameter-Entities die Wartbarkeit von XML-Dokumenten und verringern den Anteil an Schreibarbeit. Wir betrachten dazu das nachfolgende Beispiel.

interne Parameter-Entities

Beispiel 3.4.21 (Interne Parameter-Entity) *In Anlehnung an [5] werden Kontaktdaten als interne Parameter-Entities modelliert:*

<!ENTITY % KontaktDatenDefinition „Straße, HNr, PLZ, Ort, Land, TelefonNr, FaxNr“>.

Die einzelnen Bestandteile dieser Deklaration (Straße, Hausnummer, PLZ, Ort, Land, Telefon- und Faxnummer) müssen separat deklariert werden. Diese Deklaration kann wie folgt angewandt werden:

<!Element Anschrift (Vorname, Name, %Kontaktdatendefinition;)>.

Änderungen können in Beispiel 3.4.21 nachträglich leicht vorgenommen werden. Beispielsweise kann die Rufnummer eines mobilen Telefons zusätzlich zu den Kontaktdaten hinzugefügt werden. Die dadurch vorgenommene Änderung der Kontaktdaten steht dann überall dort zur Verfügung, wo die Kontaktdaten angewandt werden.

Wenn man nicht mit mehreren Dokumentinstanzen, sondern sogar mit mehreren DTDs arbeitet, kann man mit externen Parameter-Entities arbeiten. Die zugehörige Parameter-Entity ist dann eine DTD-Datei, die dann beispielsweise über einen URI zur Verfügung gestellt wird.

Das Schlüsselwort NDATA wird bei externen allgemeinen Entities zum Verweis auf eine Notationsdeklaration verwendet. Notationen dienen dazu, externe Dateien, die nicht XML-konform sind, in ein XML-Dokument einzubinden. Für jedes zu verwendende Dateiformat muss eine Notation deklariert werden. Die Notation enthält einen Verweis auf ein Programm, das für Daten des jeweiligen Typs verwendet werden kann. Notationen müssen entsprechend der nachfolgenden Syntax deklariert werden:

Notation

$$<!\text{NOTATION Name ExterneID} > .$$

Wir betrachten als Beispiel die Deklaration einer Notation, die festlegt, wie .bmp-Dateien zu behandeln sind. Dazu wird hier das entsprechende Bildbetrachtungsprogramm angegeben.

Beispiel 3.4.22 (Deklaration einer Notation) *Die Beispiele 3.4.19 und 3.4.20 werden hier erweitert. Die nachfolgende Deklaration legt fest, wie .bmp-Dateien zu behandeln sind:*

<!NOTATION bmp
SYSTEM "C:\WINDOWS\system32\mspaint.exe">.

Steueranweisungen (Processing-Instructions (PI)) dienen dazu, dass ein XML-Prozessor Informationen an diejenige Anwendung weiterleitet, in deren Kontext er läuft. Steueranweisungen müssen der folgenden Syntax genügen:

$$<?\ \text{Name Daten}\ ?> .$$

Der Bestandteil „Name“ gibt den Namen der Anwendung an, in deren Kontext der XML-Prozessor verwendet wird. Der Datenbestandteil dient dann der eigentlichen Übertragung von Daten an die Anwendung. Dabei ist es aus XML-Sicht uninteressant, ob die Anwendung überhaupt existiert oder ob diese die übergebenen Daten interpretieren kann. Durch die Verwendung von Steueranweisungen werden XML-Dokumente von einer bestimmten Systemumgebung abhängig.

3.4.1.3 Dokumenteigenschaften

XML-Prolog

Durch den XML-Prolog werden wesentliche Eigenschaften eines XML-Dokumentes festgelegt. Dazu gehören

- die XML-Versionsnummer,
- die Art der Zeichenkodierung,

- vorhandene DTDs.

Das folgende Beispiel zeigt ein gültiges XML-Dokument.

Beispiel 3.4.23 (Gültiges XML-Dokument) *Wir betrachten dazu das nachfolgende Dokument.*

```
<?xml version="1.0" encoding = „iso-8859-1"
          standalone = „yes"?>
<!DOCTYPE Route [
      <!ELEMENT Route (Description, FlowInformation, AdditionalInfo)>
      <!ELEMENT Description (ProductName, Technology, Creator+,
        Facility, CreationDate)>
      <!ELEMENT ProductName (#PCDATA)>
      <!ELEMENT Technology (#PCDATA)>
      <!ELEMENT Creator (FirstName, SureName)>
      <!ELEMENT FirstName (#PCDATA)>
      <!ELEMENT SureName (#PCDATA)>
      <!ELEMENT Facility (#PCDATA)>
      <!ELEMENT CreationDate (#PCDATA)>
      <!ELEMENT FlowInformation (ProcessStep+,SimilarFlows*)>
      <!ELEMENT ProcessStep (...)>
      ...
      <!ELEMENT SimilarFlows (RouteNames)>
      <!ELEMENT RouteNames (#PCDATA)>
      <!ELEMENT AdditionalInfo (Text)>
      <!ELEMENT Text (#PCDATA | Table | Figure)*>
      <!ELEMENT Table (...)>
      <!ELEMENT Figure (...)>
]>
<Route>
      <Description>
            <ProductName> ABCCCD100L </ProductName>
            <Technology> CMOS1002 </Technology>
            <Creator>
                  <FirstName> Karl </FirstName>
                  <SureName> Sonntag </SureName>
            </Creator>
            <Facility> X-FAB 1 Erfurt </Facility>
            <CreationDate> 2007.12.02</CreationDate>
      </Description>
      <FlowInformation>
            <ProcessStep>
                  <ProcessStepID> S102A </ProcessStepID>
```

```
                <ProcessingTime> 20 </ProcessingTime>
                <Resource> M23 </Resource>
                <AlternativeResource> M34 </AlternativeResource>
                <LoadTime> 5 </LoadTime>
                <UnloadTime> 3 </UnloadTime>
            </ProcessStep>
                ...
            <ProcessStep>
                <ProcessStepID> S103A </ProcessStepID>
                <ProcessingTime> 25 </ProcessingTime>
                <Resource> M13 </Resource>
                <Resource> M16 </Resource>
                <LoadTime> 5 </LoadTime>
                <UnloadTime> 3 </UnloadTime>
            </ProcessStep>
            <SimilarFlows>
                <RouteNames>ABCCCD1000L</RouteNames>
            </SimilarFlows>
        </FlowInformation>
        <AdditionalInfo>
            <Text> This product leads to many problems. </Text>
        </AdditionalInfo>
</Route>.
```

Das XML-Dokument dient zur Abbildung von Arbeitsplänen und basiert auf den Beispielen 3.4.12 und 3.4.13. Details der Deklarationen für die Elementtypen "ProcessStep", "Table" und "Figure" werden nicht angegeben.

Der in Beispiel 3.4.23 durch <?xml und ?> begrenzte Teil des XML-Dokumentes wird als XML-Deklaration bezeichnet. Jedes XML-Dokument muss zwingend eine XML-Deklaration enthalten.

In dieser Deklaration muss stets das Attribut „version" auftauchen, die Attribute „encoding" und „standalone" sind hingegen optional. Der Name des verwendeten Zeichensatzes wird dem Attribut „encoding" zugewiesen. Das Attribut „standalone" kann die Werte „yes" und „no" besitzen. Falls „standalone" den Wert „yes" annimmt, werden keine externen Entities referenziert, und es findet keine externe Teilmenge der DTD Anwendung. Im umgekehrten Fall ist das möglich. Der Defaultwert für das Attribut „standalone" ist „no".

In Beispiel 3.4.23 ist „Route" der Name des Dokumenttyps. Der Name eines Dokumenttyps legt fest, welches Element der XML-Instanz Wurzelelement ist, d.h., alle anderen Elemente enthält. In Beispiel 3.4.23 ist die DTD für „Route" enthalten. Die Dokumentinstanz muss dieser DTD genügen. Die dem Dokument zugrundeliegende DTD kann auch partiell oder vollständig extern deklariert werden.

Kommentare und Steueranweisungen (PI) können sowohl im Prolog als auch nach der Dokumenttypdeklaration verwendet werden.

3.4.1.4 Weitere Technologien im XML-Umfeld

Nachdem wir beschrieben haben, wie die Syntax einer Auszeichnungssprache durch die Definition eines Dokumenttyps festgelegt wird, sollen an dieser Stelle Werkzeuge vorgestellt werden, die Basisfunktionalität für die Behandlung von XML-Dokumenten zur Verfügung stellen. Die Entwicklung derartiger Werkzeuge wird durch die für alle XML-Auszeichnungssprachen geltenden Regeln wesentlich erleichtert [5].

XPath

Die XML-Path-Language (XPath) wird zur Adressierung in XML-Dokumenten verwendet. XML-Dokumente werden dabei als Bäume aufgefasst. Innerhalb der Bäume kann durch die Verwendung von Pfadausdrücken navigiert werden. XPath-Ausdrücke sind nur innerhalb eines XML-Dokumentes gültig. Diese Einschränkung wird durch die XPointer-Language aufgehoben. Bei Verwendung von XPointer ist es möglich, auch auf Stellen in externen Dokumenten zu verweisen. XPath basiert auf Techniken zur Lokalisierung von Dateien im Internet und den Navigationsmöglichkeiten von XPath in den XML-Dokumenten.

XPointer

XLink

Die XML-Linking-Language (XLink) basiert auf XPath und XPointer. XLink erlaubt die Verknüpfung von XML-Dokumenten. Durch Verwendung von XPath und XPointer kann auf Stellen in XML-Dokumenten verwiesen werden, ohne dass die entsprechende Stelle explizit gekennzeichnet sein muss.

XML-Namensräume

XML-Namensräume (XML-Namespaces) werden zur Vermeidung und Auflösung von Namenskonflikten in XML-Dokumenten eingesetzt. Ein Namensraum ermöglicht die Definition von Entity- und Attributbezeichnern in einem bestimmten Kontext. Durch das Voranstellen des jeweiligen Namensraumbezeichners werden die Entity- bzw. Attributbezeichner eindeutig.

XML-APIs

Auf die in XML-Dokumenten vorgehaltenen Daten kann im Wesentlichen durch zwei Programmierschnittstellen zugegriffen werden:

- Simple-API-for-XML (SAX),
- Document-Object-Model (DOM).

SAX stellt eine ereignisorientierte Schnittstelle dar. Das XML-Dokument wird sequentiell durchlaufen. In Abhängigkeit von den gefundenen Werten werden unterschiedliche Methoden aufgerufen.

DOM fasst XML-Dokumente als Bäume auf. Auf die Bäume kann beliebig über die Schnittstelle zugegriffen werden. Das XML-Dokument wird gelesen und anschließend in eine Hauptspeicherdatenstruktur umgewandelt. Dadurch wird eine Validierung und sogar eine Umstrukturierung der XML-Dokumente ermöglicht. Die Verarbeitung ist im Vergleich zu SAX aufwendiger. Außerdem wird

mehr Speicherplatz benötigt, da die XML-Dokumente im Hauptspeicher abgelegt werden.

XML-Schema

Neben DTD kann auch XML-Schema zur Beschreibung von XML-Auszeichnungssprachen herangezogen werden. XML-Schema erlaubt durch die Verwendung von Datentypen und die Spezifikation von Kardinalitäten eine detailliertere Beschreibung der zu modellierenden Sachverhalte. XML-Schema ist selber eine XML-Anwendung. Dadurch können die Werkzeuge zur Verarbeitung von XML-Dokumenten herangezogen werden. Der Zugriff auf die Daten eines XML-Dokumentes von einem Anwendungsprogramm aus ist in Abbildung 3.2 dargestellt.

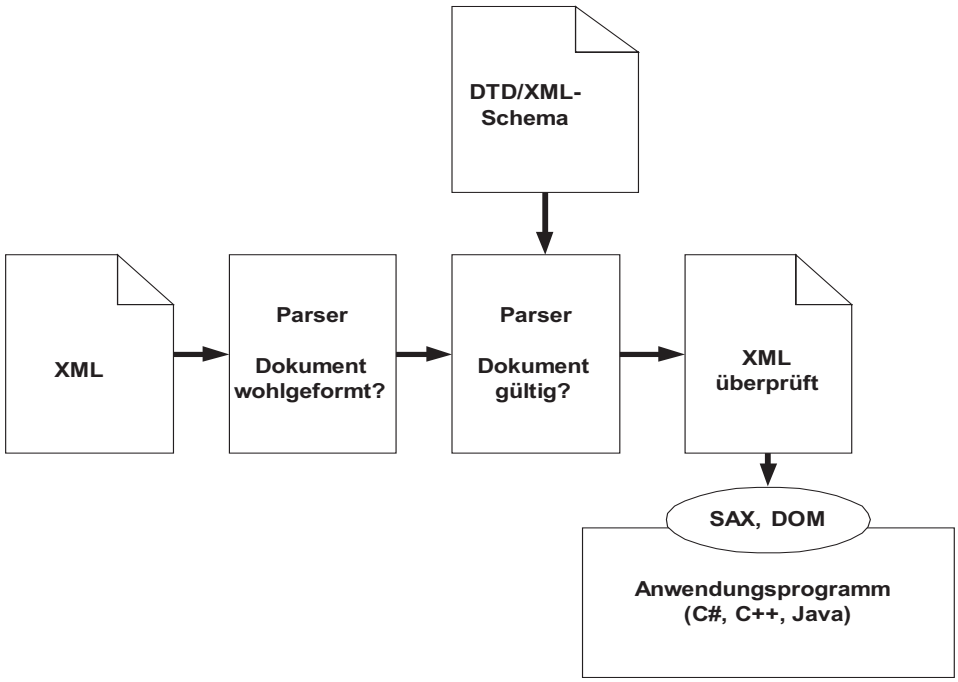

Abbildung 3.2: Ablauf beim Zugriff auf ein XML-Dokument von einem Anwendungsprogramm aus

Neben der Vorgabe der Struktur durch DTDs und der dadurch erreichten Festlegung einer Syntax ist es möglich, durch eine geeignete Wahl von Entity- und Attributbezeichnern Semantik bzw. Metadaten in XML-Dokumente einzubringen. Das wird im nachfolgenden Beispiel illustriert.

Beispiel 3.4.24 (Verwendung von Metadaten in XML-Dokumenten)
Die reelle Zahl 62.0 in einem XML-Dokument sagt wenig über deren Bedeutung aus. Wenn zusätzlich bekannt ist, dass es sich um eine Zeitdauerangabe handelt, wird die Bedeutung klarer erkennbar. Beispielsweise kann es sich um eine Bearbeitungs- oder Umrüstdauer handeln. Die Bedeutung wird noch klarer und

ermöglicht eine unmittelbare Weiterverarbeitung, wenn man durch ein weiteres Attribut modelliert, dass es sich um eine Zeitdauerangabe in Minuten handelt.

Metadaten

Neben einer geeigneten Modellierung wird die Abbildung von Metadaten durch zusätzliche Technologien unterstützt. Das Resoure-Description-Framework (RDF) ist ein syntaxunabhängiges Modell zur Modellierung von Metadaten. Da aber auch eine XML-Syntax vorhanden ist, ist eine Verarbeitung von RDF-Dokumenten mit XML-Werkzeugen möglich. Die durch RDF getätigten syntaktischen Vorgaben von Metadaten werden durch RDF-Schema verfeinert. RDF-Schema und die darauf basierende OWL Web-Ontology-Language ermöglichen die Entwicklung von Ontologien (vergleiche Abschnitt 2.2.3 für den Ontologiebegriff sowie die Arbeit [14] für die Anwendung von OWL in der Produktionsdomäne). Ontologien ermöglichen in diesem Kontext speziell das Inbeziehungsetzen verschiedener Begriffe, so dass eine automatische Weiterverarbeitung von Metadaten sowie herkömmlicher Daten möglich ist.

XML-Query

XML-Query-Language ist eine Abfragesprache, die zur Extraktion von Informationen aus XML-Dokumenten dient. XML-Query unterstützt einerseits Anfragen an stark strukturierte Dokumente. Andererseits sind auch textbasierte Anfragemöglichkeiten an semistrukturierte Dokumente möglich. Ein Dokument heißt semistrukturiert, wenn nur gelegentlich Markierungen vorkommen, um z.B. Überschriften zu kennzeichnen.

XML-Dokumente enthalten keine Layoutinformationen. Die Extensible-Stylesheet-Language (XSL) ermöglicht eine Umwandlung von XML-Dokumenten in Präsentationsformate wie HTML, PDF oder Postscript. Eine partielle Behandlung von XML-Dokumenten ist dabei auch möglich.

Das World-Wide-Web-Consortium (W3C) hat 2006 die vierte Empfehlung zur Standardisierung von XML gegeben. Industriegremien wie die Object-Management-Group beteiligen sich am Standardisierungsprozess.

XML hat insbesondere für die Webservice-Technologie eine grundlegende Bedeutung. Darauf wird in Abschnitt 4.3 dieses Kurses eingegangen.

3.4.2 Ereignisgesteuerte Prozessketten zur Geschäftsprozessmodellierung

EPKs

Zur Beschreibung von Geschäftsprozessen werden ereignisgesteuerte Prozessketten (EPKs) verwendet. Wir definieren den Begriff in Anlehnung an [8] wie folgt.

Definition 3.4.2 (Ereignisgesteuerte Prozesskette) *EPKs stellen eine Methode zur Modellierung der zeitlichen und sachlogischen Abhängigkeiten zwischen Aktivitäten und Ereignissen eines Geschäftsprozesses dar.*

EPKs werden in ARIS (vergleiche Abschnitt 2.3.2) für die Modellierung der Steuerungssicht vorgeschlagen. Sie werden in Diagrammform angegeben.

Diagrammelemente

Die wesentlichen Diagrammelemente sind Funktionen, Ereignisse, Verknüpfungen, Prozesswegweiser sowie Organisations- und Entitätstypen. Funktionen dienen der Bezeichnung von fachlichen Aufgaben, die von einem oder mehreren Objekten ausgeführt werden. Als Notation für Funktionen wird ein Rechteck mit abgerundeten Ecken verwendet. Ereignisse sind das Ergebnis von Funktionen. Auf sie kann in anderen Funktionen geeignet reagiert werden. Eine Funktion wird stets von einem oder mehreren Ereignissen, sogenannten Startereignissen, ausgelöst und hat ein oder mehrere Ereignisse, sogenannte Endereignisse, als Ergebnis. Ereignisse dienen somit der Spezifikation von Zuständen, die vor der Durchführung von Funktionen erfüllt sein müssen. Als Ergebnis der Durchführung von Funktionen werden die Zustände von Objekten verändert. Ereignisse werden in EPKs graphisch als Sechsecke notiert. Da Ereignisse Vorbedingungen und Ergebnis von Funktionen sind, wechseln sich Ereignisse und Funktionen in EPKs alternierend ab. Ereignisse und Funktionen werden durch gerichtete Kanten zu einem Prozess verbunden. Der logische und zeitliche Ablauf des Prozesses, d.h. die Ausführung von durch Funktionen repräsentierten Aktivitäten, geht aus dieser Darstellung hervor.

Logische Verknüpfungsoperatoren, alternativ als Konnektoren bezeichnet, ermöglichen es, in EPK-Diagrammen alternative (OR-Operator), exklusivalternative (XOR-Operator) sowie parallele (AND-Operator) Aktionen zu modellieren. Zur Darstellung der logischen Verknüpfungsoperatoren werden Kreise verwendet, die als Symbol den jeweiligen logischen Operator enthalten. Wir unterscheiden Ereignis- und Funktionsverknüpfungen. Der Steuerfluss von mindestens zwei Ereignissen mit einer vor- bzw. nachgelagerten Funktion wird bei einer Ereignisverknüpfung durch einen logischen Verknüpfungsoperator aufgesplittet bzw. zusammengefasst. Die Resultate von mindestens zwei unterschiedlichen Funktionen werden bei der Funktionsverknüpfung als Ereignis modelliert. Der zugehörige Steuerfluss wird mit dem entsprechenden logischen Verknüpfungsoperator versehen. Ein einzelnes Ereignis kann mit mehreren nachfolgenden Funktionen durch einen AND-Operator verbunden sein. OR- oder XOR-Operatoren dürfen an dieser Stelle nur verwendet werden, wenn das auslösende Ereignis tatsächlich entscheiden kann, welche Funktion ausgeführt wird. Bei einem sich öffnenden Konnektor sind die zugehörigen Prozessalternativen an späterer Stelle wieder zusammenzuführen.

Die Bezeichnung Prozesskette ist irreführend. Aufgrund der in EPKs auftretenden Parallelisierung ist die Bezeichnung ereignisgesteuertes Prozessnetz angemessener. Eine EPK kann als ein gerichteter Graph mit den folgenden Eigenschaften [12] interpretiert werden:

EPK als Graph

- Ereignisse, Funktionen sowie die logische Operatoren XOR, AND und OR sind die möglichen Knotentypen des Graphen.
- Alle Kanten verbinden jeweils zwei Knoten von unterschiedlichem Typ.
- Die logischen Operatoren verbinden Ereignisse mit Funktionen und umge-

kehrt. Ausschließlich logische Operatoren verzweigen. Eingehende Kanten eines Knotens vom Typ Operator sind ausgehende Kanten von Knoten vom Typ Ereignis oder vom Typ Funktion. Ausgehende Kanten eines Knotens vom Typ Operator führen entweder ausschließlich zu Knoten vom Typ Ereignis oder vom Typ Funktion.

- Knoten des Graphen, die ausschließlich ausgehende oder eingehende Kanten besitzen, sind vom Typ Ereignis. Mindestens ein Start- und ein Zielereignis sind vorhanden.

Prozesswegweiser werden zur Darstellung von Abhängigkeiten zwischen EPKs verwendet. Sie verweisen auf EPKs, die entweder vor oder nach der sie enthaltenen EPK ausgeführt werden. Prozesswegweiser werden durch abgerundete Rechtecke mit einem dahinterliegenden Sechseck symbolisiert. Dadurch soll angedeutet werden, dass Prozesswegweiser auf aus Funktionen und Ereignissen bestehende Diagramme verweisen.

Entitätstypen werden zur Modellierung von erzeugten, verbrauchten oder umgewandelten Objekten verwendet. Entitätstypen werden als einfache Rechtecke notiert. Sie werden gleichzeitig zur Modellierung der Datensicht von ARIS (vergleiche hierfür Abschnitt 2.3.2) verwendet.

Organisationseinheiten werden in EPKs durch Organisationseinheitstypen abgebildet. Wir betrachten dazu das folgende Beispiel.

Beispiel 3.4.25 (Organisationseinheitstypen) *Abteilungen, Gruppen, Projektteams und Stellen im Unternehmen sind Organisationseinheitstypen.*

Für ihre Notation werden Ellipsen verwendet, die auf der linken Seite einen waagerechten Strich enthalten. Organisationseinheitstypen sind spezielle Entitätstypen, da ihre Instanzen wie z.B. eine konkrete Abteilung im Rahmen des Geschäftsprozesses der Unternehmensgestaltung erzeugt, verändert und gegebenenfalls gelöscht werden [2]. Aufgrund der besonderen Rolle der Organisationssicht in ARIS ist aber eine Unterscheidung von den allgemeinen Entitätstypen sinnvoll. Organisationseinheitstypen werden in ARIS in der Organisationssicht verwaltet. Sowohl Organisationseinheitstypen als auch Entitätstypen werden in EPKs zur genaueren Beschreibung von Funktionen verwendet. Die einzelnen Elemente eines EPK-Diagramms sind in Abbildung 3.3 veranschaulicht.

In Abbildung 3.4 stellen wir als Beispiel eine EPK dar, die typische Geschäftsabläufe der Auftragsabwicklung modelliert. Die Abbildung wird im nachfolgenden Beispiel erläutert.

Beispiel 3.4.26 (EPK für die Auftragsabwicklung) *Abbildung 3.4 enthält ein Prozessmodell, aus dem dann das zugehörige Datenmodell sowie das Organisationsmodell abgeleitet werden können. Das Prozessmodell dient der Beschreibung der Umsetzung von Planaufträgen in Fertigungsaufträge. Ein Planauftrag entsteht zur Deckung eines Bedarfs. Planaufträge stellen somit Aufforderungen*

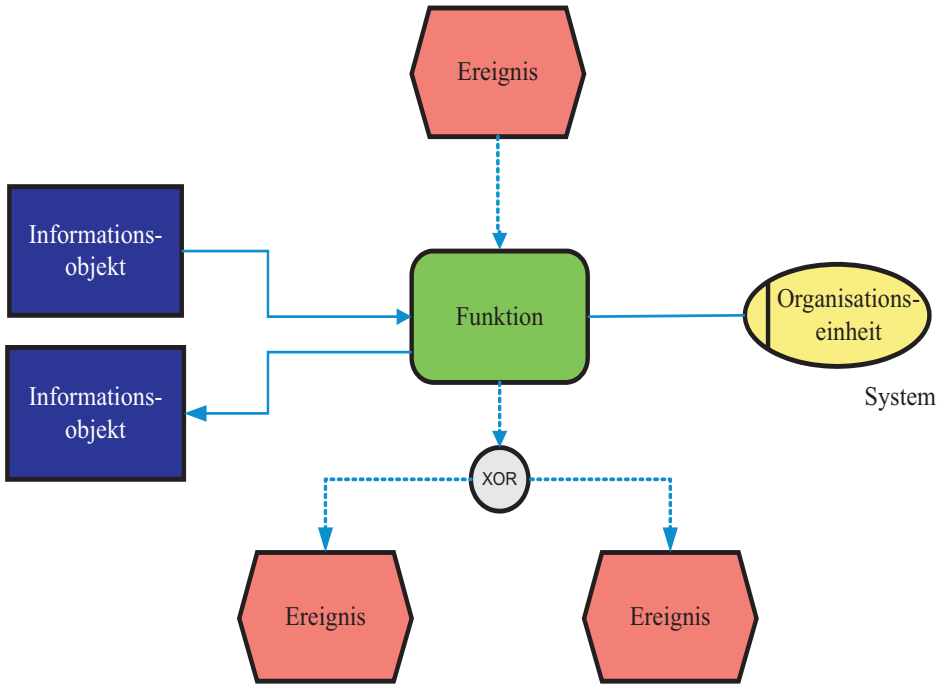

Abbildung 3.3: Darstellung der EPK-Elemente

zur Beschaffung oder Produktion eines Materials dar. Falls die Materialien des Planauftrags in Eigenfertigung hergestellt werden, ist der Planauftrag in entsprechende Fertigungsaufträge umzusetzen. Die für die Fertigung notwendigen Komponenten sind als einzelne Positionen im Planauftrag enthalten. Sie werden bei der Umsetzung des Planauftrags direkt in den Fertigungsauftrag übernommen. Anschließend kann der Fertigungsauftrag eröffnet werden. Nach der Eröffnung ist der Fertigungsauftrag angelegt, terminiert und notwendige Materialbestellungen wurden veranlasst. Die Eröffnung des Fertigungsauftrags findet in einem Werk statt. Jeder Fertigungsauftrag ist einer Disponentengruppe zugeordnet. Diese beiden Organisationseinheiten sind im Organisationsmodell zusammengefasst. Typischerweise liegt eine Folge von Fertigungsaufträgen vor. Jeder einzelne Fertigungsauftrag besteht aus einzelnen Fertigungspositionen. Einer Fertigungsposition sind bestimmte Materialien zugeordnet. Dieser Sachverhalt wird im Datenmodell abgebildet.

Referenzmodelle

EPKs sind unter anderem in verschiedenen Referenzmodellen in SAP-Systemen enthalten [2, 16].

Übungsaufgabe 3.6 (Ereignisgesteuerte Prozessketten) *Erstellen Sie eine ereignisgesteuerte Prozesskette für die Entscheidung einer Kundenauftragsannahme bei einer Kundenbestellung per Telefon. Beachten Sie dabei, dass bestimmte Funktionen parallel ausgeführt werden müssen.*

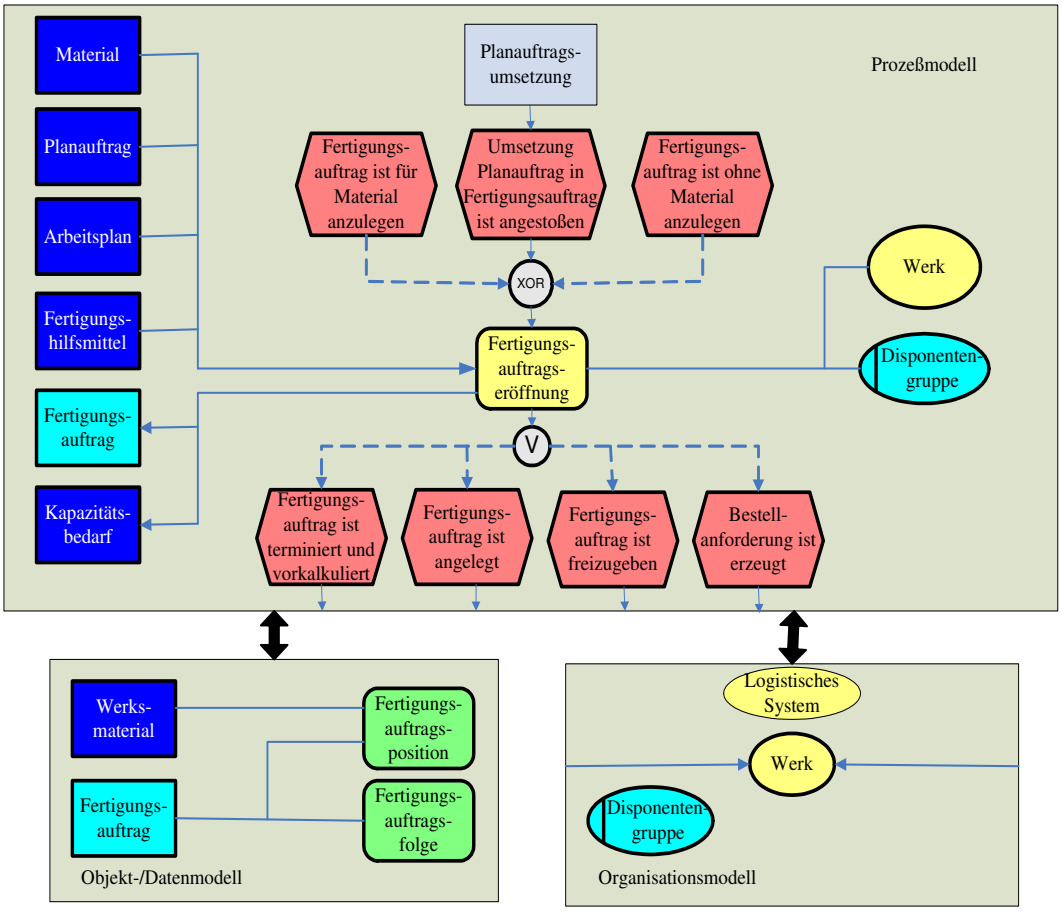

Abbildung 3.4: EPK für die Auftragsabwicklung [9]

Wenn der Kunde anruft und einen Auftrag vergibt, wird dieser zuerst definiert. Danach erfolgt die Überprüfung, ob der Kundenauftrag angenommen werden kann oder nicht. Zur Überprüfung gehören die technische Machbarkeit, die kaufmännische Überprüfung, die Kundenbonitätsprüfung und die Prüfung der Produktverfügbarkeit. Wenn alle Überprüfungen positiv verlaufen sind, wird der Kundenauftrag angenommen, andernfalls wird der Kundenauftrag abgelehnt.

3.4.3 Beschreibung der wichtigsten Elemente der Unified-Modeling-Language

Falls bereits vertiefte Kenntnisse in UML aus vorausgegangenen Lehrveranstaltungen vorhanden sind, kann dieser Abschnitt übersprungen werden.

Wir definieren zunächst die für die Objektorientierung fundamentalen Begriffe Objekt und Klasse. Objekt und Klasse

Definition 3.4.3 (Objekt und Klasse) *Unter einem Objekt verstehen wir*

einen Gegenstand mit klarer Abgrenzung und präziser Bedeutung. Ein Objekt ist also stets auch ein Konzept und geht mit einer Abstraktion einher. Eine Gruppe von gleichartigen Objekten heißt Klasse. Eine Klasse ist die Implementierung eines abstrakten Datentyps.

Objekte werden auch als Instanzen einer Klasse bezeichnet. Jedes Objekt ist Träger einer Identität. Objekte unterscheiden sich aufgrund ihrer Identität. Die Objekte sind dabei durch ihre Existenz und nicht aufgrund ihrer Eigenschaften unterscheidbar [15]. Das bedeutet, dass unterschiedliche Objekte existieren können, die in allen Eigenschaften übereinstimmen.

Die UML stellt Diagrammtypen bereit, die zur Modellierung und Visualisierung von statischen und dynamischen Elementen eines Softwaresystems verwendet werden. Wir beschränken uns auf Klassen-, Sequenz-, Aktivitäts- und Zustandsdiagramme, weil diese an verschiedenen Stellen dieses Kurses von Bedeutung sind. Gelegentlich wird aus Vereinfachungsgründen darauf verzichtet, dass die Diagramme vollständig konform zum aktuellen UML-Standard sind. Für eine vertiefte Darstellung von Elementen der UML verweisen wir auf [7].

3.4.3.1 Klassendiagramme zur Beschreibung von statischen Aspekten eines Softwaresystems

Die UML verwendet statische Strukturdiagramme, sogenannte Klassendiagramme, für das Typenmodell. Eine Klasse wird durch eine Menge von Attributen und Methoden beschrieben. Wir führen zunächst den Begriff „Attribut“ im objektorientierten Kontext ein.

Attribute

Definition 3.4.4 (Attribut) *Attribute dienen zur Beschreibung des Zustandes eines Objektes, stellen somit dessen Datenhaushalt dar.*

Methode

Anschließend definieren wir den Begriff der Methode.

Definition 3.4.5 (Methode) *Methoden sind einem Objekt zugeordnete Funktionen, die zum einen von außen mit Argumenten versehen werden können, andererseits die Werte der Attribute des jeweiligen Objekts verwenden und verändern können.*

Zur graphischen Notation werden in UML Rechtecke verwendet, die entweder nur den Namen der Klasse oder zusätzlich auch Attribute und Methoden umfassen können. Klassenbezeichner, Attribute und Methoden werden von zur Grundlinie des Rechtecks parallelen Linien voneinander abgetrennt. Attribute werden durch Attributnamen, Attributtyp, Initialwert und Zusicherungen beschrieben. Initialwerte sind Vorbelegungen eines Attributes (Defaultwerte). Diese werden automatisch verwendet, wenn dem Attribut bei der Initialisierung des Objektes kein Wert zugewiesen wird. Zusicherungen stellen eine Einschränkung des Wertebereichs von Attributen dar. Die Einschränkungen können unter anderem durch prädikatenlogische Ausdrücke angegeben werden. Für die Spezifikation

einer Methode wird ihre Signatur verwendet. Zusicherungen können zur Darstellung von Bedingungen verwendet werden, die bei Ausführung einer Methode erfüllt sein müssen.

Für Attribute und Methoden können Sichtbarkeitsmerkmale angegeben werden. Diese dienen dazu, das für die Objektorientierung typische Geheimnisprinzip (Information Hiding) zu realisieren. Durch die Sichtbarkeit wird der Zugriff auf Attribute und Methoden geregelt. Wir unterscheiden zwischen den Sichtbarkeiten Sichtbarkeit

- public, durch ein vorangestelltes + gekennzeichnet,
- protected, durch ein vorangestelltes # gekennzeichnet,
- private, durch ein vorangestelltes - gekennzeichnet.

Wenn ein Attribut oder eine Methode als public deklariert wird, kann auf das jeweilige Element von fremden Klassen aus zugegriffen werden. Im Gegenteil dazu legt eine private-Deklaration fest, dass das jeweilige Element nur innerhalb der eigenen Klasse sichtbar ist. Eine protected-Deklaration eines Attributes oder einer Methode bewirkt, dass das jeweilige Element nur für die vererbende Klasse sichtbar ist. Abbildung 3.5 zeigt als Beispiel für die Klasse „Los“ (englisch „Lot“) ein UML-Klassendiagramm. Wir beschreiben diese Klasse im nachfolgenden Beispiel genauer.

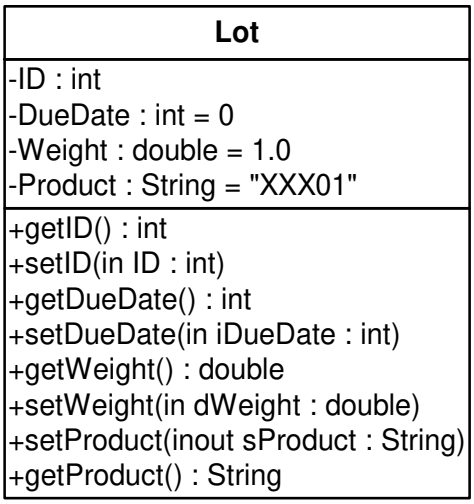

Abbildung 3.5: UML-Klassendiagramm für die Klasse „Los“

Beispiel 3.4.27 (Klasse „Los“) *Ausgehend von Kundenaufträgen werden in Produktionssystemen Lose gebildet. Jedes Los ist einem Produkt zugeordnet. Ein*

Arbeitsplan repräsentiert ein Produkt. Lose werden entsprechend dem Arbeitsplan auf Maschinen bearbeitet. Wir verweisen auf [11] bzw. auf die Ausführungen in Abschnitt 6.1.2.2 für eine umfassendere Darstellung von Stamm- und Bewegungsdaten für die Produktion. Jedes Los besitzt eine ID zur Identifizierung, außerdem einen geplanten Fertigstellungstermin sowie ein kundenspezifisches Gewicht, das die Wichtigkeit des Loses aus Kundensicht repräsentiert. Durch ein Attribut muss die Verbindung zum jeweiligen Produkt hergestellt werden. In Abbildung 3.5 geschieht das durch den Namen des Produkts. Es existieren aber auch andere Möglichkeiten (vergleiche hierzu Beispiel 3.4.29). Neben den bereits beschriebenen Attributen enthält die Klasse aus Abbildung 3.5 zunächst nur Methoden, die ein Lesen und Setzen dieser Attribute ermöglichen.

Wir stellen im Folgenden spezielle Typen von Klassen vor.

Typen von Klassen

- **Abstrakte Klassen**: Klassen dieses Typs fassen Attribute und Methoden von Unterklassen zusammen. Dadurch wird eine redundante Modellierung der Unterklassen vermieden. Von abstrakten Klassen können keine Objekte instanziiert werden. Abstrakte Klassen sind eine Möglichkeit zur Realisierung von Vererbungsbeziehungen. Unter Vererbung im objektorientierten Sinne verstehen wir einen Mechanismus, der Objekten einer Klasse Zugriff auf Attribute und Methoden einer bereits früher definierten Klasse gestattet, ohne dass diese neu definiert werden müssen [3].

- **Parametrisierbare Klassen**: Unter Generizität verstehen wir die Fähigkeit, Klassen durch einen Typ zu parametrisieren. Die Parameter der generischen Klasse können selber wieder Klassen sein. Typische Beispiele für parametrisierbare Klassen sind durch Containerklassen wie Stapel oder Listen gegeben. Containerklassen ermöglichen die Verwaltung von Objekten einer bestimmten Klasse, ohne dass die Containerklasse spezielles Wissen über den jeweiligen Objekttyp besitzt. Die Containerklasse wird mit einem Parameter T definiert. Eine Containerklasse „Liste“ wird dann wie folgt deklariert

  $$Liste < T > . \tag{3.1}$$

  Falls eine Containerklasse für einen bestimmten Typ benötigt wird, ist die Containerklasse mit diesem Typ zu instanziieren. Wir erhalten

  $$Liste < Integer > \tag{3.2}$$

  für eine Liste ganzer Zahlen. Die Sortier-, Such- und Löschoperationen auf einer Liste hängen nicht davon ab, welche Objekte eines bestimmten Typs T in der Liste vorgehalten werden.

- **Schnittstellenklassen**: Klassen dieses Typs enthalten lediglich Methodendeklarationen. Attribute und die Implementierung der Methoden fehlen. Die Definition der Methoden wird aber zwingend auf später verschoben. Schnittstellenklassen können als spezielle abstrakte Klassen realisiert werden. Als Beispiel betrachten wir einen sortierbaren Container.

Beispiel 3.4.28 (Sortierbarer Container) *Die Elemente des Containers sind Objekte, die auf bestimmte Methoden reagieren. Die folgende Schnittstelle ist zu implementieren:*

```
interface Comparable
{
        int compareTo(Comparable c);
}.
```

Der Container akzeptiert nur Objekte, die diese Schnittstelle implementieren. Die Annahmen des Containers über seine Objekte sind in dieser Schnittstelle enthalten. Dem Nutzer des Containers wird lediglich die folgende Schnittstelle zur Verfügung gestellt:

```
interface Container
{
        void put(Comparable c);
        void sort();
}.
```

Wir führen nun den Begriff „Klassendiagramm“ ein.

Klassendiagramm

Definition 3.4.6 (Klassendiagramm) *Eine Beschreibung der statischen Struktur der Objekte in einem System sowie ihrer Beziehungen untereinander erfolgt in einem Klassendiagramm.*

Vererbung

Bei der Modellierung der Beziehungen zwischen zwei Objekten in Klassendiagrammen unterscheiden wir zwischen Untertypen und Assoziationen. Eine Klasse K_1 ist Untertyp einer anderen Klassen K_2, wenn K_1 von K_2 abgeleitet wird, d.h. die Eigenschaften von K_2 erbt. Wir sprechen von einer **Vererbungsbeziehung**.

Assoziation

Im Gegensatz dazu stellt eine **Assoziation** eine allgemeine **Beziehung** zwischen zwei Klassen dar, über die Realisierung wird dabei nichts ausgesagt. Assoziationen werden durch Linien zwischen den beteiligten Klassen dargestellt. Gerichtete Linien werden durch einen Pfeil am Ende abgebildet. Der Zugriff auf Objekte der Klassen kann nur in Pfeilrichtung erfolgen. Wir sprechen an dieser Stelle von Navigationsfähigkeit. Am Assoziationsende kann eine Multiplizität angegeben werden, die beschreibt, wieviele Objekte an der Beziehung beteiligt sein können.

Aggregation

Eine **Aggregation** ist eine spezielle Assoziation, die dann entsteht, wenn ein Objekt, als Ganzes bezeichnet, sich aus anderen Objekten, Teilen genannt, zusammensetzt. Die Teile der Aggregation müssen dabei wesentlich für den Aufbau des Ganzen sein, da ansonsten lediglich eine Assoziation vorliegt. Zur Kennzeichnung von Aggregationen werden die Linien zwischen dem Ganzen und den Teilen

Komposition

mit einer Raute am Ganzen versehen. Eine **Komposition** ist eine spezielle Aggregation, in der die Teile zusätzlich physisch im Ganzen enthalten sein müssen. Das bedeutet insbesondere, dass ein Zerstören des Ganzen auch ein Zerstören der Teile nach sich zieht. Das ist bei einer Aggregation nicht notwendigerweise der Fall. Die Raute wird im Fall der Komposition durch eine schwarz gefärbte Raute ersetzt. Im nachfolgenden Beispiel 3.4.29 zeigen wir, wie Komposition und Aggregation mit Hilfe der Programmiersprache C++ realisiert werden können.

Beispiel 3.4.29 (Komposition vs. Aggregation) *Wir betrachten die Klasse „Los“ aus Beispiel 3.4.27. Jedem Los ist ein Arbeitsplan zugeordnet. Der Arbeitsplan ist unabhängig von der Existenz eines Loses. Aus diesem Grund sollen Arbeitsplanobjekte nicht zerstört werden, wenn das dazugehörige Losobjekt zerstört wird. Die Modellierung*

```
class Lot
{
....
    Product* p_my_Product; ....
};
```

ist somit sinnvoll. Die Alternative

```
class Lot
{
....
    Product my_Product; ....
};
```

d.h., ein Verzicht auf Zeiger ist nicht möglich, da in diesem Fall das Objekt der Klasse „product“ zerstört werden wird, falls das Losobjekt zerstört wird.

In Abbildung 3.6 zeigen wir ein Objektmodell, das wesentliche Bestandteile komplexer Produktionssysteme enthält, in Form eines Klassendiagramms. Wir erläutern die Abbildung im folgenden Beispiel.

Beispiel 3.4.30 (Objektmodell) *Ausgangspunkt des Objektmodells ist die Klasse D_Shop_Floor, die das Produktionssystem modelliert. Die Klasse D_Work_Area zur Abbildung von Produktionsbereichen ist mit der Klasse D_Shop_Floor durch Komposition verknüpft. Maschinengruppen und Maschinen werden durch die Klassen D_Machine_Group und D_Machine abgebildet. Die Rüstzustände einer einzelnen Maschine und die reihenfolgeabhängige Dauer der Umrüstzeit werden durch die Klassen D_Setup und D_Setup_Pair repräsentiert. Bestimmte Rüstzustände werden mit Prozess-Schritten, durch die Klasse*

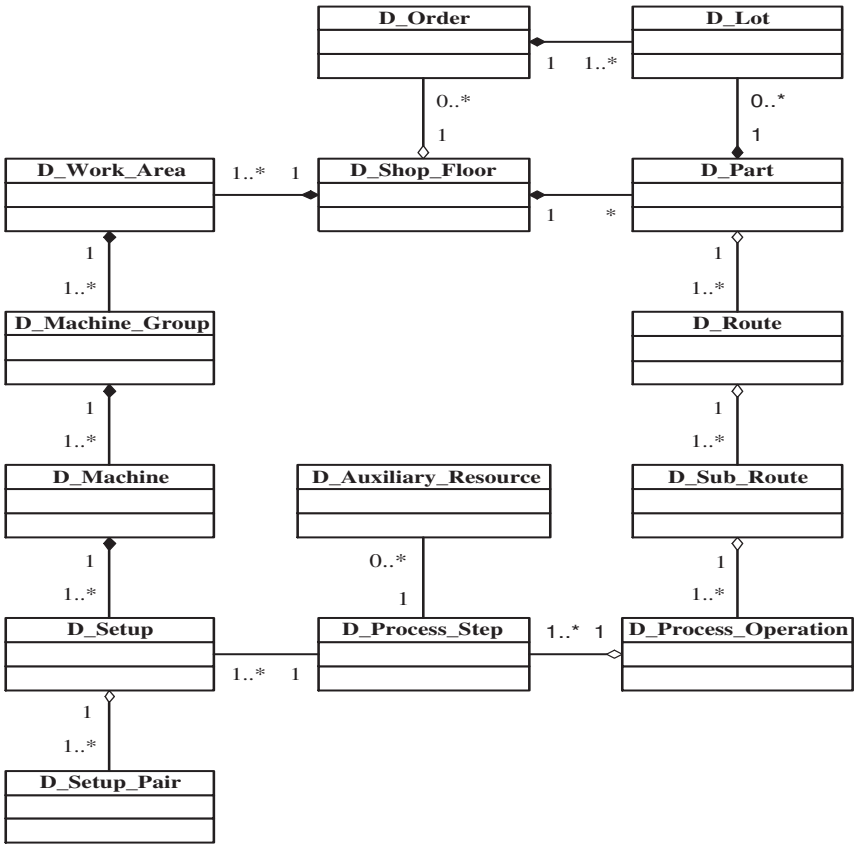

Abbildung 3.6: Aggregation und Komposition

D_Process_Step abgebildet, assoziiert. Gleichzeitig kann einem Prozess-Schritt eine sekundäre Ressource, durch die Klasse D_Auxiliary_Resource modelliert, zugeordnet werden. Prozess-Schritte werden zu Operationen, durch die Klasse D_Process_Operation dargestellt, zusammengefasst. Bestimmte aufeinanderfolgende Operationen sind Bestandteil von Teilarbeitsplänen, die durch die Klasse D_Sub_Route modelliert werden. Teilarbeitspläne werden zu Arbeitsplänen, die durch D_Route modelliert werden, aggregiert. Arbeitspläne sind wesentliche Bestandteile von Produkten. Produkte werden im Objektmodell durch die Klasse D_Part repräsentiert. Jedem Produkt wird die Menge der Lose dieses Produktes zugeordnet. Lose werden durch die Klasse D_Lot dargestellt. Da Aufträge Bestandteile des Produktionssystems sind, wird die Untersetzung von Aufträgen durch Lose durch die Komposition von D_Lot zu D_Order abgebildet.

Übungsaufgabe 3.7 (Klassendiagramm) *Erstellen Sie ein Klassendiagramm, in dem alle kundenspezifischen Daten abgebildet werden, die zur Verwaltung der Kunden und zur Abwicklung eines Bestellvorgangs benötigt werden. Versuchen Sie, die hier beschriebenen Abhängigkeiten und Konstrukte soweit wie*

möglich umzusetzen. Die Verwendung von Assoziationsklassen und reinen Datenklassen wird empfohlen.
Bei einem Kunden kann es sich sowohl um einen Geschäfts- als auch um einen Privatkunden handeln. Vorgänge von Privatkunden werden immer vom selben Sachbearbeiter bearbeitet, nur bei Geschäftskunden ist die Zuweisung eines Sachbearbeiters notwendig. Für beide Kundenarten werden Informationen zu den Lieferparametern, zu den Rechnungsparametern und zur Preisfindung abgelegt. Zu den Lieferparametern gehört die Lieferanschrift der Kunden, wobei ein Kunde auch mehrere Lieferanschriften haben kann. Die Rechnungsparameter umfassen Informationen zur Rechnungslegung. Da ein Produkt für unterschiedliche Kundengruppen, z.B. Großkunden, Privatkunden etc., unterschiedliche Preise haben kann, werden diese Informationen zur Preisfindung entsprechend abgelegt.
Bei einem Bestellvorgang des Kunden werden durch diesen Produkte des Unternehmens bestellt. Ein Bestellvorgang umfasst somit mehrere Bestellpositionen. Dem eigentlichen Auftrag des Kunden, bei dem die Produkte bestellt werden, kann ein Angebot des Unternehmens vorausgehen. Der Auftrag bezieht sich dann auf dieses Angebot, wenn der Kunde damit einverstanden ist.

3.4.3.2 Sequenzdiagramme zur Modellierung objektübergreifender Abläufe

Interaktionsdiagramm

Häufig ist es notwendig, die Interaktionen mehrerer Akteure zu modellieren. Dazu werden Interaktionsdiagramme verwendet. Wir führen zunächst den Begriff des Sequenzdiagramms ein.

Definition 3.4.7 (Sequenzdiagramm) *Diagramme, die der Beschreibung des Nachrichtenaustauschs zwischen unterschiedlichen Objekten im zeitlichen Verlauf dienen, heißen Sequenzdiagramme. Sequenzdiagramme sind spezielle Interaktionsdiagramme.*

Nachrichtenaustausch zwischen Objekten bedeutet dabei, entsprechende Methoden aufzurufen.

Sequenzdiagramm

Neben Klassendiagrammen sind Sequenzdiagramme die wichtigste Diagrammart der UML.

Zur Darstellung der am Nachrichtenaustausch beteiligten Objekte werden Rechtecke verwendet, die mit dem Namen des Objektes beschriftet sind. Jedem Objekt wird eine im Diagramm von oben nach unten verlaufende (gestrichelte) Lebenslinie zugeordnet, die gleichzeitig als Zeitachse fungiert. Auf den Lebenslinien werden langgezogene Rechtecke angeordnet, die angeben, wann und wie lange eine bestimmte Aktivität ausgeführt wird. Diese Rechtecke werden als „Focus-of-Control“ bezeichnet. Das Löschen eines Objektes wird durch ein Kreuz auf seiner Lebenslinie angedeutet. Ein Nachrichtenversand zwischen zwei Objekten wird durch einen Pfeil zwischen aufrufendem und aufgerufenem Objekt symbolisiert. Pfeile für Nachrichten verlaufen somit von links nach rechts im Sequenzdiagramm. Falls ein Objekt eine eigene Methode aufruft, spricht man von

Selbstdelegation. Die Nachrichtenpfeile werden mit dem entsprechenden Methodennamen bezeichnet. Vorbedingungen für den Aufruf einer Methode werden in eckige Klammern gesetzt. Die Rückkehr aus einer Nachricht wird durch gestrichelte Pfeile dargestellt. Bei asynchronen Nachrichten werden offene Pfeilspitzen verwendet.

Sequenzdiagramme werden in Situationen angewendet, in denen relativ wenige Klassen und viele Nachrichten auftreten.

Kontraktnetz

In Abildung 3.7 stellen wir ein Sequenzdiagramm für das Zusammenwirken von einem Manager und einem Agenten im Rahmen eines Kontraktnetzansatzes dar. Die grundlegende Idee des Kontraktnetzansatzes besteht darin, dass eine öffentliche Ausschreibung anstehender Aufgaben durch einen Manager stattfindet und die als Agenten bezeichneten Knoten sich für die für sie interessanten Aufgaben bewerben können [21, 10, 20]. Die einzelnen Phasen eines dem Kontraktnetzprotokoll folgenden Verhandlungsablaufs sind nachfolgend angegeben:

1. **Phase**: Eine Ausschreibung des Kontraktes durch den Manager erfolgt.

2. **Phase**: Die Ausschreibung wird durch alle Agenten evaluiert. Der einzelne Agent entscheidet, ob eine Bewerbung für die ausgeschriebene Aufgabe abgegeben werden kann, falls ja, wird eine entsprechende Mitteilung an den Manager verschickt.

3. **Phase**: Eine Bewertung der eingegangenen Bewerbungen und die Auswahl eines geeigneten Agenten wird durch den Manager vorgenommen. Die Bewerbung mit höchster Bewertung wird ausgewählt. Der diese Bewertung vertretende Agent erhält den Zuschlag.

4. **Phase**: Der Manager informiert den Agenten über den Zuschlag, der beauftragte Agent bestätigt abschließend die Bereitschaft zur Bearbeitung der Teilaufgabe.

Kollaborationsdiagramm

Neben Sequenzdiagrammen stellen Kollaborationsdiagramme ebenfalls Interaktionsdiagramme dar [7]. Wir verzichten an dieser Stelle auf eine Beschreibung.

Übungsaufgabe 3.8 (Sequenzdiagramm) *Erstellen Sie für den Bestellvorgang ein Sequenzdiagramm. Am Vorgang sind neben dem Kunden der Vertrieb und die Finanzbuchhaltung des Unternehmens beteiligt.*
Nach dem Eingang der Aufforderung zur Abgabe eines Angebots durch den Kunden wird dieses durch den Vertrieb erstellt. Wird das Angebot akzeptiert, so wird ein Auftrag erteilt und bestätigt. Nach der Versendung der Lieferung wird der Auftrag an die Finanzbuchhaltung zur Rechnungslegung weitergeleitet. Diese erstellt die Rechnung und wartet den Zahlungseingang des Kunden ab. Der Zahlungseingang wird dem Vertrieb mitgeteilt, der den Bestellvorgang abschließt.

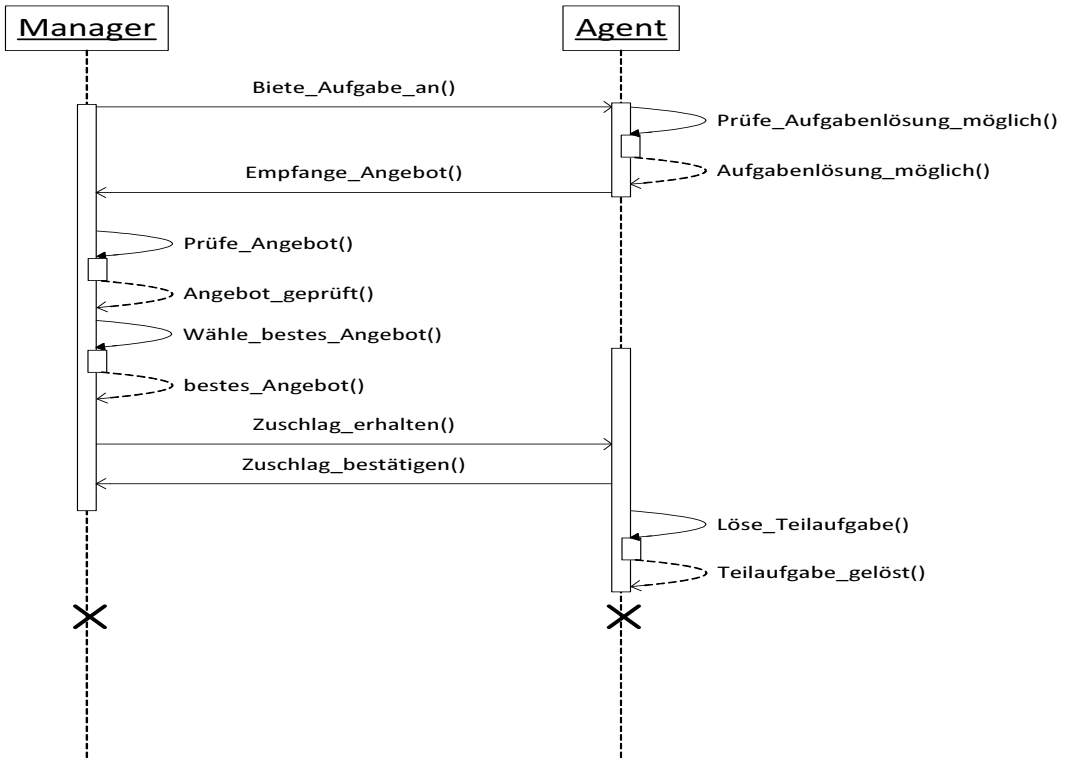

Abbildung 3.7: UML-Sequenzdiagramm für ein Kontraktnetzprotokoll

3.4.3.3 Zustandsdiagramme zur Modellierung objektinterner Abläufe

Zustands-diagramm

Aus Sequenzdiagrammen können Zustandsdiagramme entwickelt werden. Das Verhalten genau einer Klasse wird durch ein Zustandsdiagramm spezifiziert. Wir definieren zunächst den Begriff des Zustandsdiagramms.

Definition 3.4.8 (Zustandsdiagramm) *Ein Verhaltensdiagramm, mit dem die Reaktion bestimmter Objekte auf festgelegte Ereignisse beschrieben werden kann, heißt Zustandsdiagramm. Durch ein Zustandsdiagramm werden der Zustand und die Zustandsübergänge der beteiligten Objekte modelliert.*

Für Klassen, die nur Werte speichern oder nur als Schnittstelle dienen, d.h. das Verhalten weiter delegieren, sind Zustandsdiagramme nicht erforderlich. Wir betrachten das nachfolgende Beispiel für Zustandsdiagramme.

Beispiel 3.4.31 (Anwendung von Zustandsdiagrammen) *Die Modellierung realer Objekte, die umgangssprachlich als Automaten bezeichnet werden (Geldautomaten, Fahrstühle), kann mit Zustandsdiagrammen erfolgen. Für die Verhaltensmodellierung von Klassen, die Übertragungsprotokolle realisieren sowie von Klassen, die Benutzeraktionen abarbeiten, können ebenfalls Zustandsdiagramme eingesetzt werden.*

Rechtecke mit abgerundeten Ecken werden als Notation für den Zustand eines Objektes verwendet. Jeder Zustand kann durch ein eigenes Zustandsdiagramm verfeinert werden. Eintreffende Ereignisse lösen Zustandsübergänge aus. Ein Pfeil entspricht dabei einem Zustandsübergang. Der Pfeil dient somit der Ereigniskennzeichnung. Die einen Zustandsübergang hervorrufenden Ereignisse mit entsprechenden Bedingungen sowie entsprechenden Aktionen werden zur Pfeilbeschriftung verwendet.

Ein Zustand führt zum Ausführen von Aktivitäten mit einer gewissen Zeitdauer. Eine **Aktion** ist eine Operation mit Zeitdauer 0, die an Zustandsübergängen ausgeführt wird. Aktionen werden somit ausgeführt, wenn der Zustand erreicht wird und wenn der Zustand verlassen wird. In Abbildung 3.8 stellen wir schematisch den Übergang von einem ersten Zustand zu einem zweiten Zustand dar. Der rechts abgebildete Zustand ist generisch, er dient der Darstellung der verwendeten Notation. Es wird deutlich, dass ein Zustand durch eine Eintrittsaktion, eine Aktivität sowie eine Austrittsaktion gekennzeichnet ist. Auf die Beschreibung der Aktivitäten, Aktionen sowie Initialwerte wird häufig verzichtet. Im Zustand auf der linken Seite hingegen wird die Notation auf den konkreten Zustand „Kennworteingabe“ angewandt.

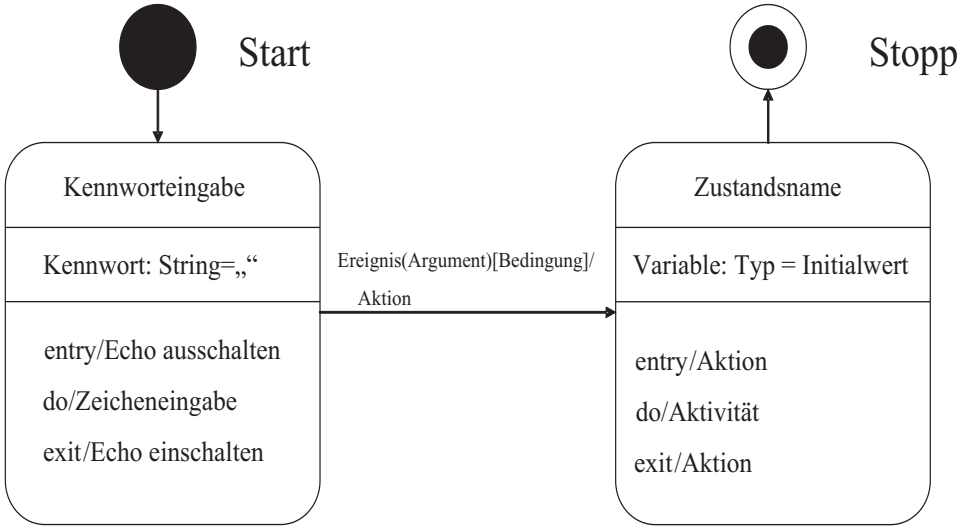

Abbildung 3.8: UML-Zustandsdiagramm [19]

In Abbildung 3.9 geben wir jeweils Zustandsdiagramme für den in Abschnitt 3.4.3.2 beschriebenen Manager und einen Knoten (Agenten) innerhalb eines Kontraktnetzansatzes an. Auf der linken Seite der Abbildung befindet sich das Zustandsdiagramm für den Manager, während auf der rechten Seite das Zustandsdiagramm für einen Knoten des Kontraktnetzes zu finden ist.

Übungsaufgabe 3.9 (Zustandsdiagramm) *Erstellen Sie ein Zustandsdiagramm für einen Kunden. Welche Zustandsmenge läßt sich hier in Verbindung*

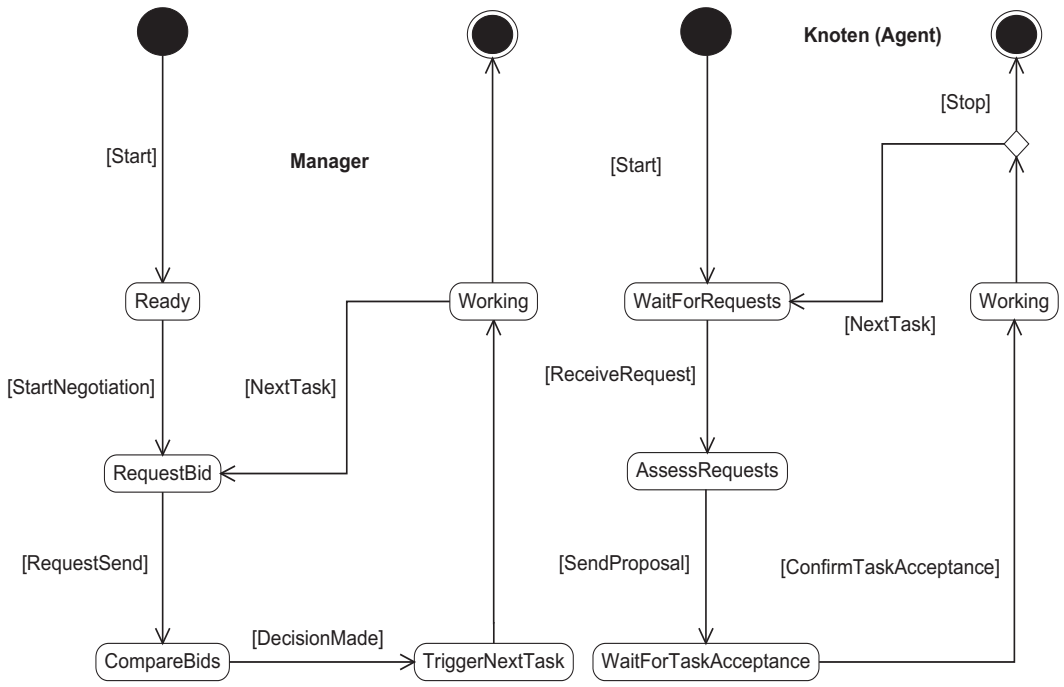

Abbildung 3.9: UML-Zustandsdiagramm für Manager und Agenten

mit einem Bestellvorgang abbilden? Beginnen Sie mit der Aufnahme des Kunden in den Datenbestand des Unternehmens. Vereinfachend können Sie annehmen, dass für einen Kunde nur ein Angebot oder ein Auftrag parallel bearbeitet werden kann.

3.4.3.4 Aktivitätsdiagramme zur Modellierung von objektübergreifenden Abläufen

Aktivitätsdiagramm

Während Zustandsdiagramme auf einzelne Objekte abzielen, ist es häufig notwendig, Abläufe zu modellieren, die mehrere Objekte beinhalten. Dazu dienen Aktivitätsdiagramme. Wir definieren zunächst den Begriff des Aktivitätsdiagramms.

Definition 3.4.9 (Aktivitätsdiagramm) *Ein Verhaltensdiagramm zur Modellierung von Abläufen wird Aktivitätsdiagramm genannt. Dabei können sowohl Vorgänge innerhalb eines einzelnen Systems als auch Abläufe beschrieben werden, die Interaktionen von Benutzern oder Systemen untereinander beschreiben.*

Aktivitätsdiagramme erlauben insbesondere auch die Beschreibung von parallelem Verhalten. Grundelemente dieser Diagrammart sind **Aktivitäten**. Darunter verstehen wir Zustände, in denen Vorgänge ablaufen. Das zweite Grundelement sind **Transitionen**. Sie erfolgen automatisch am Ende der Aktivitäten und stellen Zustandsübergänge dar. Sie werden durch Pfeile dargestellt. Synchronisa-

tionslinien bilden das dritte Element. Sie schalten, wenn alle Eingangstransitionen vorhanden sind. Als viertes Grundelement dienen Swimlanes (Schwimmbahnen). Sie teilen Aktivitäten derartig ein, dass die Bereiche einzelnen Objekten zugeordnet werden können.

Ein Beispiel für Aktivitätsdiagramme ist bereits durch Abbildung 2.17 gegeben. Dort wurden Aktivitätsdiagramme zur Darstellung von Choreographieszenarien verwendet.

In Kurseinheit 4 wird die während der Beschreibung von Abbildung 3.7 begonnene Diskussion von Softwareagenten im Rahmen eines Kontraktnetzansatzes wieder aufgenommen und vertieft.

3.5 Realisierung von Kopplungsarchitekturen

Kopplungsarchitektur

Nachdem wir in Kurseinheit 2 den Architekturbegriff für Informationssysteme eingeführt haben, möchten wir in diesem Abschnitt zeigen, wie die Kopplung von Anwendungssystemen im Sinne einer Konstruktion vollzogen werden kann. Wir führen dazu zunächst den Begriff der Kopplungsarchitektur ein.

Definition 3.5.1 (Kopplungsarchitektur) *Die Gesamtheit der für die Kopplung von Anwendungssystemen erforderlichen Elemente der Anwendungssystemarchitekturen sowie deren Beziehungen werden als Kopplungsarchitektur bezeichnet.*

Heterogenität

Kopplungsarchitekturen dienen der unternehmensweiten sowie der überbetrieblichen Integration von Anwendungssystemen. Bei der Entwicklung von Kopplungsarchitekturen müssen geeignete Maßnahmen zur Überwindung der technischen und fachlichen Heterogenität der zu koppelnden Anwendungssysteme ergriffen werden [17].

Beispiel 3.5.1 (Fachliche Heterogenität) *Anwendungssysteme unterscheiden sich bezüglich der Benennung und Repräsentation gleichartiger Attribute.*

Betriebsunabhängigkeit

Bei der Entwicklung von Kopplungsarchitekturen muss die Betriebs- und Entwicklungsunabhängigkeit der zu koppelnden Anwendungssysteme berücksichtigt werden. Wir definieren zunächst den Begriff der Betriebsunabhängigkeit.

Definition 3.5.2 (Betriebsunabhängigkeit) *Die Fähigkeit eines Anwendungssystems, zu einem beliebigen Zeitpunkt, inbesondere auch dann, wenn Wartungsarbeiten durchgeführt werden, Nachrichten eines anderen Anwendungssystems zu empfangen, heißt Betriebsunabhängigkeit.*

Entwicklungsunabhängigkeit

Wir definieren nun den Begriff der Entwicklungsunabhängigkeit.

Definition 3.5.3 (Entwicklungsunabhängigkeit) *Wenn Änderungen am Quellcode eines Nachrichten empfangenden Anwendungssystems keine Änderungen am Quellcode des die Nachrichten sendenden Anwendungssystems nach sich ziehen, sprechen wir von Entwicklungsunabhängigkeit.*

Im Folgenden unterscheiden wir zwischen dem Kern des Anwendungssystems und dem Kopplungssubsystem. Die Bestandteile des Anwendungssystemkerns sind für die Kopplung des Anwendungssystems nicht bedeutsam bzw. werden nicht ausschließlich für die Kopplung benutzt. Das **Kopplungssubsystem** umfasst diejenigen Bestandteile eines Anwendungssystems, die ausschließlich der Kopplung mit weiteren Anwendungssystemen dienen. Die Kopplungsarchitektur ist somit durch das **Kopplungssubsystem** sowie durch diejenigen Bestandteile des **Anwendungssystemkerns** gegeben, die auch für die Kopplung verwendet werden.

Kopplungssubsystem

Zur Verdeutlichung der Begriffe Kopplungssubsystem und Anwendungssystemkern betrachten wir das nachfolgende Beispiel.

Beispiel 3.5.2 (Kopplungssubsystem sowie Anwendungssystemkern) *Moderne ERP-Systeme (vergleiche für diese Klasse von betrieblichen Anwendungssystemen die Ausführungen in Abschnitt 5.3.6) ermöglichen anderen Anwendungssystemen den Zugriff auf ihre Daten u.a. über Webservices. Die Datenhaltung erfolgt dabei innerhalb des ERP-Systems in Tabellen einer relationalen Datenbank. Die unter Verwendung von Webservices realisierte Zugriffsschicht ist Bestandteil des Kopplungssubsystems, während die Datenbanktabelle, auf die zugegriffen wird, Bestandteil des ERP-System-Kerns ist, da sie auch außerhalb des Kopplungskontextes verwendet wird.*

Wir beschreiben in diesem Abschnitt der Reihe nach ereignis-, daten- und funktionsorientierte Kopplungsarchitekturen und ihre softwaretechnische Umsetzung in Anlehnung an [17].

3.5.1 Ereignisorientierte Kopplungsarchitekturen

Wir definieren zunächst den Begriff der ereignisorientierten Kopplungsarchitektur.

ereignisorientierte Kopplungsarchitektur

Definition 3.5.4 (Ereignisorientierte Kopplungsarchitektur) *Zur Übertragung von Ereignissen und den damit assoziierten Daten zwischen Anwendungssystemen werden ereignisorientierte Kopplungsarchitekturen verwendet.*

Die Übertragung der Ereignisse findet dabei in Form eines Nachrichtenaustauschs statt. Eine lose Kopplung der beteiligten Anwendungssysteme wird auf diese Art und Weise vorgenommen. Für den Nachrichtenaustausch müssen Nachrichtenformate sowie Kommunikationsprotokolle festgelegt werden.

Ereignisorientierte Kopplungsarchitekturen werden zur Abbildung der nachfolgenden Aufgabenbeziehungen eingesetzt [17]:

- Reihenfolgebeziehungen zwischen Aufgaben,
- partielle Identität von AO-Instanzen,

- partielle Gleichheit von AO-Typen.

Reihenfolgebeziehungen und die partielle Identität auf AO-Instanzebene werden durch ereignisorientierte Kopplungsarchitekturen standardmäßig unterstützt. Die partielle Gleichheit von AO-Typen kann weniger gut behandelt werden.

Im weiteren Verlauf der Ausführungen beschreiben wir die Grundprinzipien ereignisorientierter Kopplungsarchitekturen [17]. Wir betrachten dazu die Aufgaben $A_i, i = 1, 2, \ldots$. Die Aufgabe A_1 ist dadurch ausgezeichnet, dass das Nachereignis dieser Aufgabe das Vorereignis der anderen Aufgaben $A_i, i \neq 1$, bildet. Außerdem benötigen die Aufgaben $A_i, i \neq 1$, Ergebnisse der Aufgabe A_1, d.h., die AO-Instanzen sind partiell identisch. Wir bezeichnen das Anwendungssystem für die Aufgabe A_1 als Quellanwendungssystem, während angenommen wird, dass die verbleibenden Aufgaben durch ein oder mehrere Zielanwendungssysteme ausgeführt werden.

Das durch die Kopplungsarchitektur zu übermittelnde Ereignis tritt zunächst im Anwendungssystemkern des Quellanwendungssystems auf. Der Anwendungskern besteht aus Subsystemen zur Realisierung der Anwendungsfunktionen sowie aus Subsystemen zur Datenverwaltung. Empfänger dieses Ereignisses sind die bereits beschriebenen Zielanwendungssysteme. Das Kopplungssubsystem hat die Bestandteile

Bestandteile eines Kopplungssubsystems

- Ereignissubsystem,
- Empfängersubsystem,
- Datensubsystem,
- Kommunikationssubsystem,
- Heterogenitätssubsystem.

Das **Ereignissubsystem** weiß, welche Ereignisse des Quellanwendungssystems für die Zielanwendungssysteme von Bedeutung sind. Das Ereignissubsystem beobachtet den Quellanwendungssystemkern unter diesem Gesichtspunkt.

Das **Empfängersubsystem** verteilt die Ereignisse des Quellanwendungssystems auf die relevanten Zielanwendungssysteme. Die Zielanwendungssysteme werden entsprechend dem Typ des auftretenden Ereignisses ausgewählt. Falls eine Auswahl auf Basis des Ereignistyps nicht ausreichend ist, sind die nachfolgenden Varianten für Empfängersubsysteme zu unterscheiden:

- zustandssensitiv,
- inhaltssensitiv.

Unter einem zustandssensitiven Empfängersubsystem verstehen wir ein System, welches das Zielanwendungssystem in Abhängigkeit von vorher eingetretenen

Ereignissen auswählt. Im Gegensatz dazu verwendet ein inhaltssensitives Empfängersubsystem die mit den Ereignissen übertragenen Daten zur Festlegung des Zielanwendungssystems.

Das **Datensubsystem** überträgt zusätzlich zu den Ereignissen die mit den Ereignissen assoziierten Daten an das Zielanwendungssubsystem.

Das **Kommunikationssubsystem** überträgt die Ereignisse und die mit den Ereignissen assoziierten Daten vom Quellanwendungssystem in das Zielanwendungssystem. Dabei wird die technische Heterogenität zwischen den an der Kopplung beteiligten Anwendungssystemen durch die Kommunikation überwunden. Durch eine asynchrone, gepufferte Kommunikation wird erreicht, dass das Quellanwendungssystem unabhängig von der Verfügbarkeit des Zielanwendungssystems ist. Dadurch wird eine hohe Betriebsunabhängigkeit von Quell- und Zielanwendungssystem erreicht.

Das **Heterogenitätssubsystem** überwindet die fachliche Heterogenität zwischen Quell- und Zielanwendungssystem. Typischerweise werden die zu übertragenden Daten im Quellanwendungssystem zunächst in ein Übertragungsformat umgesetzt, welches dann im Zielanwendungssystem wiederum in das Format des Zielanwendungssystems übersetzt wird. Durch dieses Vorgehen wird eine höhere Entwicklungsunabhängigkeit erreicht, da die gekoppelten Anwendungssysteme nicht die internen Formate der anderen Anwendungssysteme kennen müssen. Auf der Seite des Zielanwendungssystems werden die Ereignisse und die damit verbundenen Daten durch das Kommunikationssubsystem entgegengenommen. Das Empfängersubsystem des Zielanwendungssystems legt die für die Verarbeitung der übertragenen Ereignisse notwendigen Funktionen im Zielanwendungssystem fest. Die übertragenen Daten werden durch das Heterogenitätssubsystem in die Formate des Zielanwendungssystems umgesetzt. Die dargestellte Situation ist in Abbildung 3.10 veranschaulicht. Anders als in dieser Abbildung dargestellt, sind allerdings prinzipiell mehrere Zielanwendungssysteme möglich. Die grau gefärbten Subsysteme bilden das Kopplungssubsystem, während die weißen Subsysteme den Anwendungssystemkern abbilden.

Wir betrachten das folgende Beispiel für ereignisorientierte Kopplungsarchitekturen im Produktionsumfeld.

Beispiel 3.5.3 (Ereignisorientierte Kopplungsarchitekturen) *Wenn ein MES oder ein ERP-System durch zusätzliche Planungs- und Steuerungsfunktionalität angereichert werden soll, kommen ereignisorientierte Kopplungsarchitekturen zum Einsatz. In diesem Fall werden Prozessmodelle, die wichtige Geschäftsobjekte wie Lose und Maschinen abbilden, im Hauptspeicher eines Rechners vorgehalten. Die Geschäftsobjekte werden ereignisgetrieben aktualisiert. Wenn beispielsweise die Bearbeitung eines Arbeitsgangs eines Loses abgeschlossen wird und dies über die Betriebsdatenerfassung an das MES gemeldet wird, wird gleichzeitig das Geschäftsobjekt zur Abbildung des Loses im Hauptspeicher aktualisiert. Eine erneute Belegung der Maschine, auf der das Los bearbeitet wurde, wird durch das MES angestoßen.*

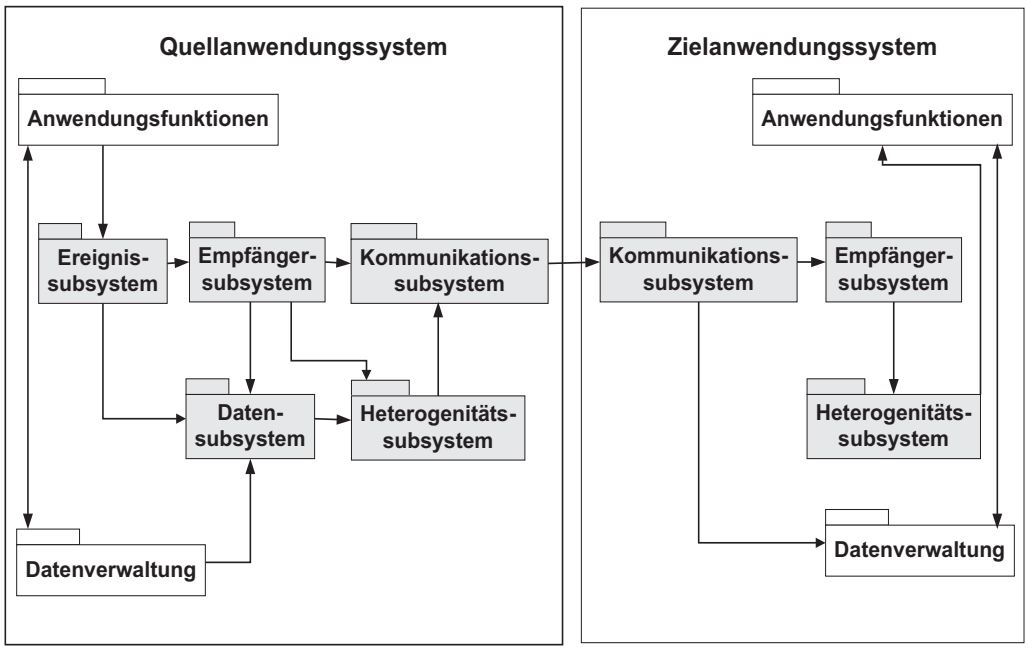

Abbildung 3.10: Ereignisorientierte Kopplungsarchitektur [17]

Die in Beispiel 3.5.3 beschriebene Nutzung ereignisorientierter Kopplungsarchitekturen wird im Multi-Agenten-System FABMAS, das in Abschnitt 7.1.5 dargestellt wird, angewandt.

3.5.2 Datenorientierte Kopplungsarchitekturen

Wir definieren zunächst den Begriff der datenorientierten Kopplungsarchitektur.

datenorientierte Kopplungsarchitektur

Definition 3.5.5 (Datenorientierte Kopplungsarchitektur) *Das Ziel datenorientierter Kopplungsarchitekturen besteht darin, die Nutzung gemeinsamer Daten mehrerer Anwendungssysteme durch die Kopplung der die Daten verarbeitenden Funktionen zu erreichen.*

Im Gegensatz zu ereignisorientierten Kopplungsarchitekturen realisieren datenorientierte Kopplungsarchitekturen eine enge Kopplung zwischen den an der Kopplungsarchitektur beteiligten Anwendungssystemen.

Wir unterscheiden zwischen der Kopplung auf AO-Typ- und AO-Instanzebene. Eine Kopplung auf Instanzebene dient der gemeinsamen Nutzung von Daten durch mehrere Anwendungssysteme. Eine Kopplung auf AO-Typ-Ebene ermöglicht die gemeinsame Nutzung von Datenspezifikationen durch mehrere Anwendungssysteme.

Datenorientierte Kopplungsarchitekturen ermöglichen die Abbildung der nachfolgenden Aufgabenbeziehungen [17]:

- partielle Identität von AO-Instanzen,
- partielle Gleichheit von AO-Typen.

Die partielle Identität von AO-Instanzen wird durch datenorientierte Kopplungsarchitekturen ermöglicht, die partielle Gleichheit von AO-Typen hingegen nur von solchen Kopplungsarchitekturen, die eine Abstimmung von Datentypspezifikationen zulassen.

redundante Datenhaltung

Für datenorientierte Kopplungsarchitekturen ist wesentlich, ob redundante Datenbestände verwaltet werden oder nicht. Im Folgenden beschreiben wir zunächst eine Kopplungsarchitektur mit **redundanter Datenhaltung**. Dabei nehmen wir an, dass mehrere Aufgaben vorliegen, die teilweise identische AO-Instanzen beinhalten. Einschränkend wird angenommen, dass lediglich eine Aufgabe die gemeinsamen AO-Instanzen verändert. Diese Aufgabe wird vom Quellanwendungssystem ausgeführt. Wir betrachten ausdrücklich nicht den Fall, dass mehrere Aufgaben versuchen, die gemeinsamen AO-Instanzen zu modifizieren. Dieser Fall führt zu Schreibkonflikten bei einem gleichzeitigen Zugriff (für eine Lösung derartiger Probleme wird auf [1] oder auf Lehrveranstaltungen zu Datenbanken oder verteilten Systemen verwiesen).

Die gemeinsamen AO-Instanzen werden redundant gespeichert, um eine hohe Betriebsunabhängigkeit zu erreichen. Eine redundante Speicherung hat aber den Nachteil, dass besondere Mechanismen zur Wahrung der Datenkonsistenz verwendet werden müssen [1, 4].

Die für den Fall der redundanten Datenhaltung notwendige datenorientierte Kopplungsarchitektur ist der ereignisorientierten Kopplungsarchitektur ähnlich. Aus diesem Grund beschreiben wir nur die Unterschiede. Die Architektur ist in Abbildung 3.11 dargestellt.

Der wesentliche Unterschied zwischen ereignis- und datenorientierter Kopplung mit redundanter Datenhaltung besteht darin, dass das Kopplungssubsystem im Zielanwendungssystem nicht mit dem Subsystem für Anwendungsfunktionen, sondern mit dem Datenverwaltungssubsystem verbunden ist. Die vom Quellanwendungssystem übertragenen Daten werden im Datenverwaltungssubsystem des Zielanwendungssystems abgelegt.

Dieser Kopplungsarchitektur liegt ein **Push-Prinzip** zugrunde. Das Ereignissubsystem überträgt die Daten entweder zu genau definierten Zeitpunkten oder bei Eintreten von fachlichen Ereignissen im Anwendungssystemkern.

Die zu übertragenden Daten werden vom Datensubsystem ermittelt. Es werden entweder alle Daten oder nur die geänderten Daten übertragen.

Im Zielanwendungssystem wird durch das Aktualisierungssubsystem eine Aktualisierung der Daten vorgenommen. Dabei werden die Schnittstellen des Datenverwaltungssubsystems genutzt. Die Reihenfolge der Datenänderung ist sowohl im Quell- als auch im Zielanwendungssystem identisch. Datenbanktransaktionen des Quellanwendungssystems sind im Zielanwendungssystem ebenfalls zu bilden.

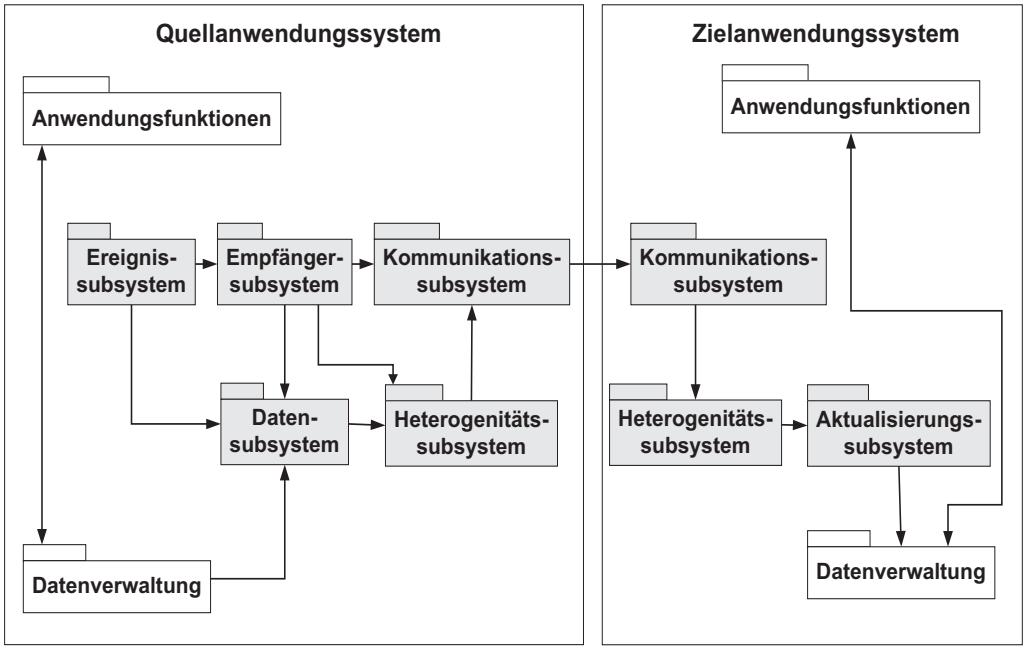

Abbildung 3.11: Datenorientierte Kopplung: redundante Datenhaltung [17]

Wir geben nachfolgend ein Beispiel für eine datenorientierte Kopplungsarchitektur mit redundanter Datenhaltung an.

Beispiel 3.5.4 (Datenorientierte Kopplungsarchitektur I) *Wir betrachten ein verteiltes Ablaufplanungsproblem. Das Produktionssystem setzt sich aus Produktionsbereichen zusammen. Jeder Produktionsbereich besteht aus Gruppen paralleler Maschinen. Jedem Produktionsbereich ist ein eigenes System zur Ablaufplanung zugeordnet, das Ablaufpläne für die Lose im jeweiligen Produktionsbereich berechnet. Da Lose häufig zwischen den Produktionsbereichen wechseln, ist es sinnvoll, die Daten aller Lose in allen Ablaufplanungssystemen vorzuhalten. Veränderungen im Losstatus werden zunächst nur an das Ablaufplanungssystem gemeldet, das dem Produktionsbereich zugeordnet ist. Dieses Ablaufplanungssystem informiert die Ablaufplanungssysteme der anderen Produktionsbereiche.*

nicht redundante Datenhaltung

Neben datenorientierten Kopplungsarchitekturen mit redundanter Datenhaltung spielen **datenorientierte Kopplungsarchitekturen mit nicht redundanter Datenhaltung** ebenfalls eine große Rolle. Diese Architekturvariante stellt eine zentralisierte, nicht redundante Datenverwaltung zur Verfügung.

Gemeinsame AO-Instanzen können anders als bei Kopplungsarchitekturen mit redundanter Datenhaltung durch mehrere Aufgaben verändert werden.

Ein eigenes Datenverwaltungssubsystem dient der Verwaltung der zentral vorgehaltenen Daten. Das Datenverwaltungssubsystem bietet eine standardisierte Datenschnittstelle an und behandelt konfliktäre Schreibzugriffe geeignet.

Die unterschiedlichen Kommunikationssubsysteme dienen der Übertragung von Abfragen der Anwendungssystemkerne an die zentrale Datenverwaltungskomponente sowie der Übertragung der Anfrageergebnisse. Die fachliche Heterogenität wird durch die beteiligten Heterogenitätssubsysteme überwunden.

Die beschriebene Situation ist in Abbildung 3.12 gezeigt. Es ist zu erkennen, dass das Subsystem zur zentralen Datenhaltung Bestandteil von beiden Anwendungssystemen ist. Das Subsystem kann entweder als eigenständige Komponente oder zusammen mit anderen Subsystemen eines der an der Kopplung beteiligten Anwendungssysteme in einer Komponente softwaretechnisch realisiert werden.

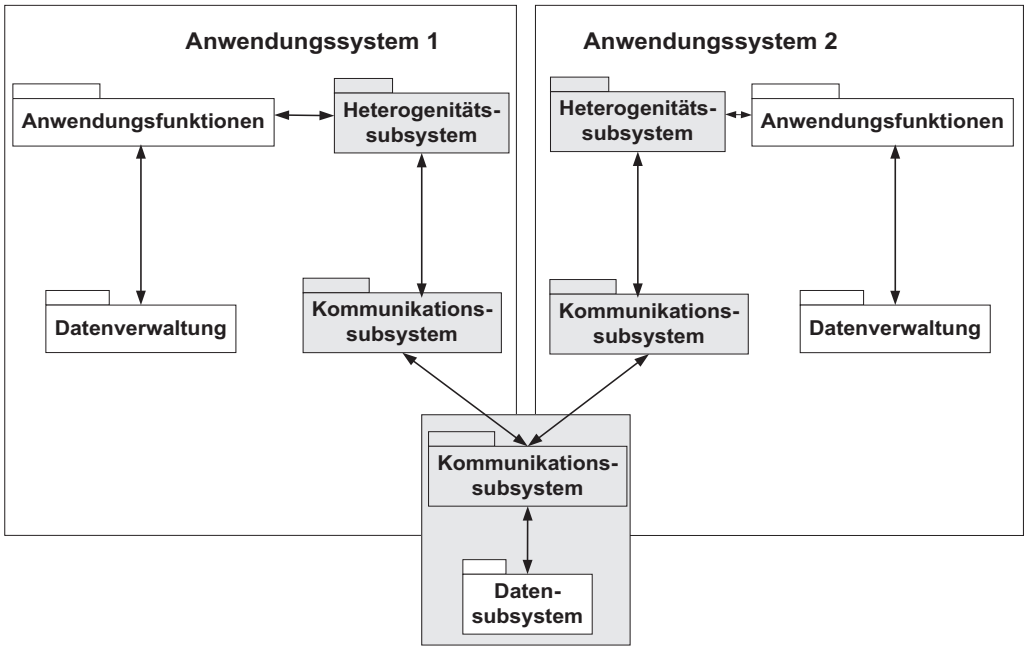

Abbildung 3.12: Datenorientierte Kopplung mit zentraler Datenhaltung [17]

Wir betrachten dazu das folgende Beispiel für diese Variante der Ausgestaltung einer Kopplungsarchitektur.

Beispiel 3.5.5 (Datenorientierte Kopplung II) *Wir erweitern dazu Beispiel 3.5.4. Die Losdaten werden in einem zentralen Datenverwaltungssubsystem vorgehalten. Neben den eigentlichen Losdaten werden auch die Ablaufpläne für die verschiedenen Produktionsbereiche gespeichert. Die Ablaufplanungssysteme lesen die Ablaufpläne aller Produktionsbereiche, um Informationen über zukünftige Losankünfte in ihrem eigenen Produktionsbereich in die Ablaufplanung einfließen zu lassen.*

Diese Architekturvariante entspricht der Datenhaltung in zentralen, unternehmensweiten Datenbanken. Das Datenbankverwaltungssystem ist in diesem Fall für die Gewährleistung eines Mehrbenutzerbetriebs zuständig.

3.5.3 Funktionsorientierte Kopplungsarchitekturen

Wir definieren zunächst den Begriff der funktionsorientierten Kopplungsarchitektur.

funktionsorientierte Kopplungsarchitektur

Definition 3.5.6 (Funktionsorientierte Kopplungsarchitektur) *Wenn Funktionen und damit assoziierte Daten durch mehrere Anwendungssysteme gemeinsam genutzt werden sollen, kommen funktionsorientierte Kopplungsarchitekturen zum Einsatz.*

Bei der Realisierung von funktionsorientierten Kopplungsarchitekturen müssen die nachfolgenden Sachverhalte geklärt werden:

- Sollen die Komponenten, die gemeinsame Funktionen realisieren, redundant oder nicht redundant zur Verfügung gestellt werden?
- Ist eine Ausführung der gemeinsam genutzten Funktionen in einem gemeinsamen Prozess sinnvoll?
- Wie werden gemeinsam genutzte Funktionen aktiviert?

Funktionsorientierte Kopplungsarchitekturen dienen der Behandlung der folgenden Aufgabenbeziehungen [17]:

- partielle Gleichheit von Lösungsverfahren,
- partielle Gleichheit von AO-Typen,
- partielle Identität von AO-Instanzen.

Die partielle Gleichheit von Lösungsverfahren wird durch funktionsorientierte Kopplungsarchitekturen in jedem Fall unterstützt. Die Unterstützung der anderen beiden Arten von Aufgabenbeziehungen hängt von der konkreten Gestaltung der Kopplungsarchitektur ab.

redundante Funktionen

Wir beschäftigen uns zunächst mit funktionsorientierten Kopplungsarchitekturen, bei denen die Komponenten zur Realisierung der gemeinsamen Funktionen **redundant** vorgehalten werden. Weiterhin soll die zu untersuchende Klasse von funktionsorientierten Kopplungsarchitekturen über eine redundante Datenhaltung verfügen. Die gemeinsam genutzten Funktionen werden in separaten Prozessen ausgeführt.

Wir gehen von einer Menge von Aufgaben aus, die ein teilweise gleiches Lösungsverfahren benötigen. Außerdem sollen identische AO-Instanzen vorliegen, die dann vom gemeinsamen Lösungsverfahren verwendet werden. Die zu beschreibende funktionsorientierte Kopplungsarchitektur soll die Gleichheit der Lösungsverfahren und die Identität der AO-Instanzen unterstützen. Im weiteren Verlauf der Darstellung wird davon ausgegangen, dass die Pflege der Komponente für die gemeinsam genutzten Funktionen sowie die Pflege der gemeinsam

genutzten Daten von genau einem Anwendungssystem vorgenommen wird. Wir bezeichnen dieses Anwendungssystem als Quellanwendungssystem.

Die beschriebene Situation ist in Abbildung 3.13 dargestellt. Sowohl das Quell- als auch das Zielsystem verfügen über eine Kopie der Komponenten, welche die gemeinsam benutzten Funktionen enthält. Die Funktionen werden jeweils in einem eigenen Prozess auf der eigenen Kopie der gemeinsam benutzten Daten ausgeführt. Dieses Vorgehen sichert eine hohe Betriebsunabhängigkeit.

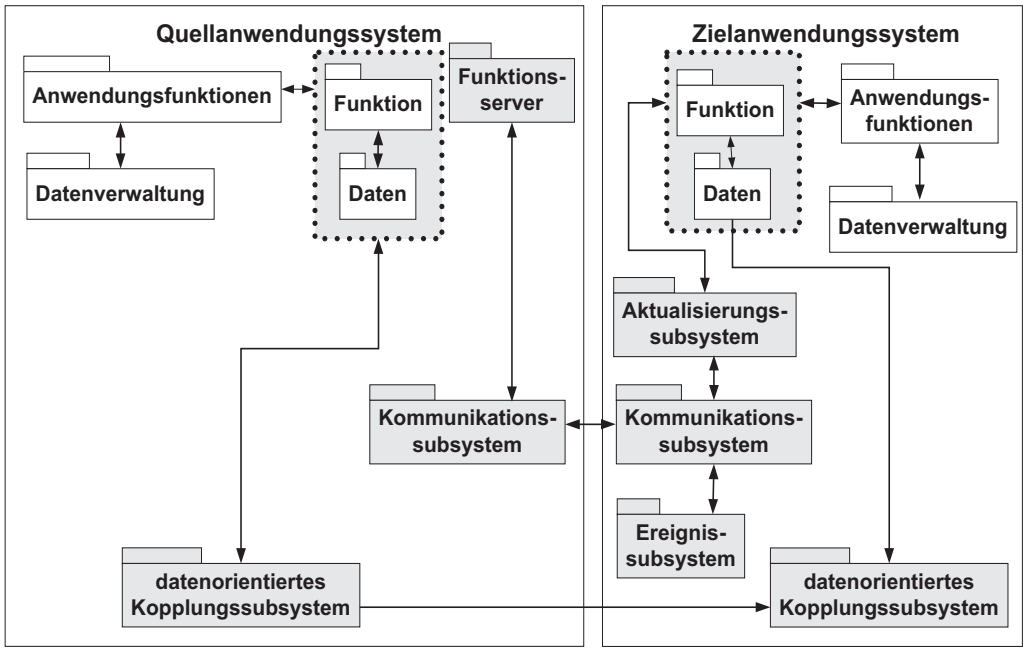

Abbildung 3.13: Funktionsorientierte Kopplung mit redundanten Funktionskomponenten, separaten Prozessen und redundanter Datenhaltung [17]

Das redundante Vorhalten von gemeinsam benutzten Funktionen in zwei Komponenten führt zu Problemen bei Änderungen an den gemeinsam benutzten Funktionen. Wir bemerken, dass die durch die redundante Datenhaltung auftretenden Probleme bereits im Rahmen der Beschreibung der datenorientierten Kopplungsarchitekturen behandelt worden sind. Wir weisen auf diesen Sachverhalt in Abbildung 3.13 durch ein zusätzliches datenorientiertes Kopplungssubsystem sowohl im Quell- als auch im Zielanwendungssystem hin.

Zur Übertragung von Änderungen an den gemeinsam benutzten Funktionen wird vom Zielanwendungssystem in bestimmten Abständen beim Quellanwendungssystem nachgefragt, ob neue Versionen der gemeinsam genutzten Komponenten vorliegen. Der Kopplungsarchitektur liegt also ein **Pull-Prinzip** zugrunde. Das Ereignissubsystem des Zielanwendungssystems bestimmt dazu die Zeitpunkte, zu denen beim Quellanwendungssystem nachgefragt werden soll und stößt die Anfragen an. Eine Anfrage beinhaltet den Namen der gemeinsam be-

nutzten Komponenten sowie die gegenwärtige Version im Zielanwendungssystem. Die Aktualisierung der gemeinsam benutzten Komponente im Zielanwendungssystem darf nicht zu Störungen im Betrieb des Zielanwendungssystems führen.

Funktionsserver

Der in Abbildung 3.13 dargestellte **Funktionsserver** ist ein wesentlicher Bestandteil dieser Kopplungsarchitektur. Der Funktionsserver verwaltet für das Quellanwendungssystem die zu replizierenden Komponenten. Falls das Zielanwendungssystem nach neuen Versionen der gemeinsam benutzten Komponente nachfragt, vergleicht der Funktionsserver die vom Zielanwendungssystem genutzte Version mit der aktuellen Version. Falls die aktuelle Version jünger als die vom Zielanwendungssystem genutzte Version ist, wird durch das Kommunikationssubsystem des Quellanwendungssystems die aktuelle Version an das Zielanwendungssystem übertragen.

Das Aktualisierungssubsystem im Zielanwendungssystem sorgt dafür, dass die bisherige Komponente im Zielanwendungssystem durch die neuere ersetzt wird. Wir betrachten das nachfolgende Beispiel zu dieser Kopplungsvariante.

Beispiel 3.5.6 (Funktionsorientierte Kopplung I) *Wir modifizieren die in Beispiel 3.5.4 beschriebene Situation. Dazu nehmen wir an, dass die Ablaufplanungsfunktionalität in den Ablaufplanungssystemen identisch ist. Veränderungen des Quellcodes in einem der Ablaufplanungssysteme müssen den anderen Ablaufplanungssystemen bekannt gemacht werden.*

nicht redundante Funktionen

Neben der bisher beschriebenen Ausprägung der funktionsorientierten Kopplungsarchitektur ist auch eine Ausprägung möglich, in der die zu koppelnden Anwendungssysteme eine gemeinsame Kopie der Komponenten sowie der Daten innerhalb eines Prozesses nutzen. Diese Architektur ist in Abbildung 3.14 dargestellt.

Kern dieser Kopplungsarchitektur ist ebenfalls wieder ein Funktionsserver, der die Funktionen verwaltet und diese den zu koppelnden Anwendungssystemen zur Verfügung stellt. Der Funktionsserver leitet die Aufrufe an die entsprechenden Funktionen weiter, koordiniert parallele Aufrufe und übernimmt die Zugriffskontrolle. Wir betrachten dazu das nachfolgende Beispiel.

Beispiel 3.5.7 (Funktionsorientierte Kopplung II) *Verfahren zur Bedarfsvorhersage werden typischerweise von menschlichen Planern und von verschiedenen Planungsverfahren verwendet. Aus diesem Grund ist es sinnvoll, die Bedarfsvorhersagefunktionalität in einen entsprechenden Funktionsserver auszulagern.*

Ein derartiger Funktionsserver kann unter Verwendung von komponentenorientierten Laufzeitumgebungen (CORBA, .NET, Enterprise-Java-Beans) implementiert werden. Laufzeitumgebungen werden im Abschnitt 4.2 über Middleware beschrieben.

Kommunikationssubsysteme auf Seiten der zu koppelnden Anwendungssysteme und des Funktionsservers dienen der Behandlung von Anfragen. Die zu

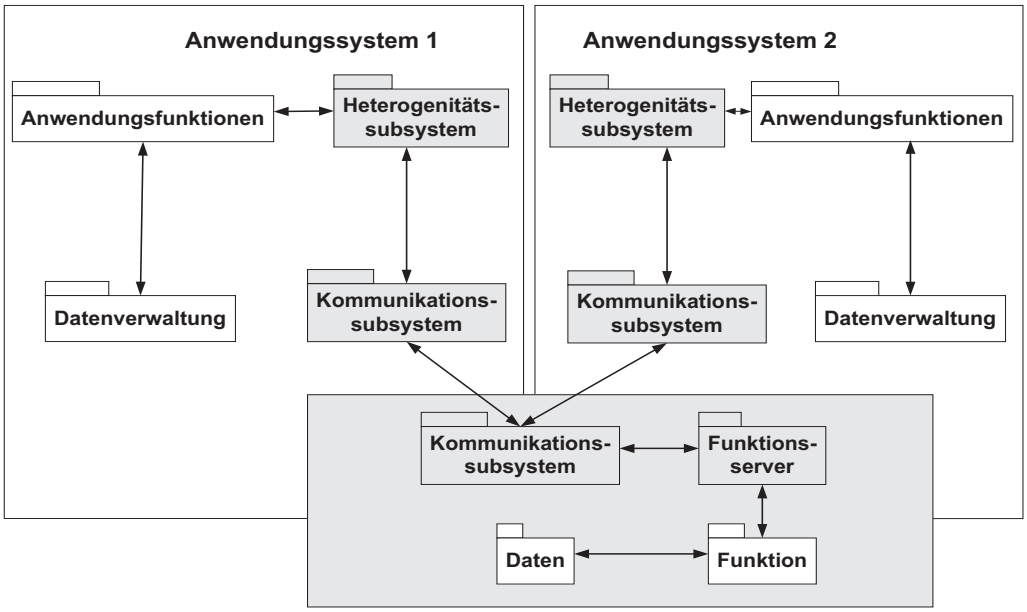

Abbildung 3.14: Funktionsorientierte Kopplung mit nicht redundanter Funktionskomponente, gemeinsamem Prozess und nicht redundanter Datenhaltung [17]

koppelnden Anwendungssysteme besitzen gegebenenfalls Heterogenitätssubsysteme, die zur Überwindung der fachlichen Heterogenität bei Anfragen und bei der Ergebnisbereitstellung dienen.

Wir bemerken, dass Webservices, die in Abschnitt 4.3 der Kurseinheit 4 beschrieben werden, zur Realisierung des Kommunikationssubsystems verwendet werden können.

Lösungen zu den Übungsaufgaben

Übungsaufgabe 3.1
Das Aufgabenobjekt ist im Fall der Angebotserstellung die Kundenanfrage, die durch das Vorereignis des Eintreffens dieser Kundenanfrage die Verrichtung der betrieblichen Aufgabe Angebotserstellung auslöst. Nach der Verrichtung dieser Aufgabe wird durch das Nachereignis das Angebot an den Kunden verschickt. Das Ziel der Aufgabe umfasst die Erstellung des Angebots für den Kunden auf Grundlage der von dem Kunden gewünschten Eckdaten wie Menge oder Liefertermin. Der Aufgabenobjekt-Typ umfasst folgende Attribute:

- Identifikation des Produkts ($Produkt$),
- Menge ($Menge$),
- gewünschter Liefertermin ($Termin$),
- Zahlungsweise des Kunden($Zahlungsweise$),
- Kundengruppe ($Kundengruppe$).

Eine Aufgabenobjekt-Instanz kann wie folgt aussehen:

- $Produkt$ = P0815,
- $Menge$ = 20,
- $Termin$ = 20.07.2008,
- $Zahlungsweise$ = Vorkasse,
- $Kundengruppe$ = Endabnehmer.

Zur Erstellung des Angebots sind durch das Lösungsverfahren der Preis in Abhängigkeit von der Menge, der Zahlungsweise, der Kundengruppe sowie der mögliche Liefertermin unter Berücksichtigung der Lagerbestände und der Kapazitäten des Produktionssystems zu ermitteln.

Übungsaufgabe 3.2
Wir betrachten weiterhin die Angebotserstellung und die Auftragsbearbeitung. Die Angebotserstellung ist ein Vorereignis für die Auftragsbearbeitung, da auf Grundlage des erstellten Angebots und der Zusage des Kunden die Auftragsbearbeitung vorgenommen wird.
Sowohl bei der Angebotserstellung als auch bei der Auftragsbearbeitung wird auf Kunden-, Produkt- und Bestandsdaten zugegriffen. Die partielle Gleichheit von Aufgabenobjekt-Typen liegt vor, wenn beispielsweise die Struktur der Kundendaten identisch ist. So gehören bei beiden Systemen der Name und

die Anschrift zu den Attributen des Kunden. Wird von beiden Systemen aus auf dieselbe Kundendatenbank zugegriffen, so handelt es sich um die partielle Identität von Aufgabenobjekt-Instanzen. Die partielle Gleichheit von Lösungsverfahren liegt vor, wenn beispielsweise in den unterschiedlichen Außenstellen des Unternehmens ein Webservice zur Angebotserstellung verwendet wird, der auf einem zentralen Server des Unternehmens läuft.

Übungsaufgabe 3.3
Es müssen zuerst das Wurzelelement Auftrag und die drei Kindelemente Bestellposition, Lieferdaten und Anmerkungen erstellt werden. Danach werden diese mit den entsprechenden Kindelementen und den dazugehörigen Zeichenketten anhand der Beschreibung ergänzt. Zu berücksichtigen ist dabei, dass es sich bei & um ein Sonderzeichen handelt, das entsprechend kodiert werden muss. Die Dokumentinstanz kann wie folgt erstellt werden:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<Auftrag>
  <Bestellposition>
    <Positionsnummer>1</Positionsnummer>
    <Produktname>P0815</Produktname>
    <Menge>20</Menge>
    <Einheit>Stück</Einheit>
  </Bestellposition>

  <Lieferdaten>
    <Kunde>
      <Name>König &amp; Partner</Name>
      <Anschrift>
        <Strasse>Fleyerstrasse</Strasse>
        <Nummer>100</Nummer>
        <PLZ>58097</PLZ>
        <Ort>Hagen</Ort>
      </Anschrift>
    </Kunde>
    <Lieferdatum>20.07.2008</Lieferdatum>
  </Lieferdaten>

  <Anmerkungen>
    <Text>Dieser Auftrag hat eine hohe Dringlichkeit!</Text>
  </Anmerkungen>
</Auftrag>
```

Übungsaufgabe 3.4

Beim Element Auftrag muss berücksichtigt werden, dass mehrere Bestellpositionen möglich sind. Dies erfolgt durch die Verwendung einer Wiederholung. Da bei der Anschrift sowohl eine Adresse als auch ein Postfach angegeben werden kann, muss die alternative Anordnungsmöglichkeit verwendet werden. Das Postfach besteht nur aus einem Textelement, während die Adresse aus zwei Textelementen besteht. Dies muss berücksichtigt werden, da immer nur jeweils ein Element als Alternative angegeben werden kann. Die Angabe der Lieferanschrift ist optional und muss dementsprechend gekennzeichnet werden.
Die Dokumententyp-Definition und die Dokumentinstanz werden wie folgt erstellt:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<!DOCTYPE Auftrag[
  <!ELEMENT Auftrag (Bestellposition+, Lieferdaten, Anmerkungen)>
  <!ELEMENT Bestellposition (Positionsnummer, Produktname, Menge, Ein-
heit)>
  <!ELEMENT Positionsnummer (#PCDATA)>
  <!ELEMENT Produktname (#PCDATA)>
  <!ELEMENT Menge (#PCDATA)>
  <!ELEMENT Einheit (#PCDATA)>
  <!ELEMENT Lieferdaten (Kunde, Lieferdatum)>
  <!ELEMENT Kunde (Name, Anschrift, Lieferanschrift?)>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Anschrift ((Adresse | Postfach), PLZ, Ort)>
  <!ELEMENT Lieferanschrift ((Adresse | Postfach), PLZ, Ort)>
  <!ELEMENT Adresse (Strasse, Nummer)>
  <!ELEMENT Strasse (#PCDATA)>
  <!ELEMENT Nummer (#PCDATA)>
  <!ELEMENT Postfach (#PCDATA)>
  <!ELEMENT PLZ (#PCDATA)>
  <!ELEMENT Ort (#PCDATA)>
  <!ELEMENT Lieferdatum (#PCDATA)>
  <!ELEMENT Anmerkungen (Text)>
  <!ELEMENT Text (#PCDATA)*>
]>

<Auftrag>
  <Bestellposition>
    <Positionsnummer>1</Positionsnummer>
    <Produktname>P0815</Produktname>
    <Menge>20</Menge>
    <Einheit>Stück</Einheit>
  </Bestellposition>
```

```
  <Lieferdaten>
    <Kunde>
      <Name>König &amp;amp; Partner</Name>
      <Anschrift>
        <Adresse>
          <Strasse>Fleyerstrasse</Strasse>
          <Nummer>100</Nummer>
        </Adresse>
        <PLZ>58097</PLZ>
        <Ort>Hagen</Ort>
      </Anschrift>
    </Kunde>
    <Lieferdatum>20.07.2008</Lieferdatum>
  </Lieferdaten>

  <Anmerkungen>
    <Text>Dieser Auftrag hat eine hohe Dringlichkeit!</Text>
  </Anmerkungen>
</Auftrag>
```

Übungsaufgabe 3.5

Für die Menge wird eine Attributliste modelliert, die aus dem Element Einheit besteht. Die Positionsnummer wird zu einem leeren Element, da der Wert als Attribut angegeben wird. Bei der Angabe der Werte für das Attribut ist zu beachten, dass diese nicht nur aus Zahlen bestehen dürfen. Die DTD und die Dokumentinstanz können wie folgt erstellt werden:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<!DOCTYPE Auftrag[
  <!ELEMENT Auftrag (Bestellposition+, Lieferdaten, Anmerkungen)>
  <!ELEMENT Bestellposition (Positionsnummer, Produktname, Menge)>
  <!ELEMENT Positionsnummer EMPTY>
  <!ATTLIST Positionsnummer
    id ID #REQUIRED
  >
  <!ELEMENT Produktname (#PCDATA)>
  <!ELEMENT Menge (#PCDATA)>
  <!ATTLIST Menge
    Einheit CDATA #REQUIRED
  >
  <!ELEMENT Lieferdaten (Kunde, Lieferdatum)>
  <!ELEMENT Kunde (Name, Anschrift, Lieferanschrift?)>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Anschrift ((Adresse | Postfach), PLZ, Ort)>
```

```
  <!ELEMENT Lieferanschrift ((Adresse | Postfach), PLZ, Ort)>
  <!ELEMENT Adresse (Strasse, Nummer)>
  <!ELEMENT Strasse (#PCDATA)>
  <!ELEMENT Nummer (#PCDATA)>
  <!ELEMENT Postfach (#PCDATA)>
  <!ELEMENT PLZ (#PCDATA)>
  <!ELEMENT Ort (#PCDATA)>
  <!ELEMENT Lieferdatum (#PCDATA)>
  <!ELEMENT Anmerkungen (Text)>
  <!ELEMENT Text (#PCDATA)*>
]>

<Auftrag>
  <Bestellposition>
    <Positionsnummer id = "p1"/>
    <Produktname>P0815</Produktname>
    <Menge Einheit = "Stück">20</Menge>
  </Bestellposition>

  <Lieferdaten>
    <Kunde>
      <Name>König &amp; Partner</Name>
      <Anschrift>
        <Adresse>
          <Strasse>Fleyerstrasse</Strasse>
          <Nummer>100</Nummer>
        </Adresse>
        <PLZ>58097</PLZ>
        <Ort>Hagen</Ort>
      </Anschrift>
    </Kunde>
    <Lieferdatum>20.07.2008</Lieferdatum>
  </Lieferdaten>

  <Anmerkungen>
    <Text>Dieser Auftrag hat eine hohe Dringlichkeit!</Text>
  </Anmerkungen>
</Auftrag>
```

Übungsaufgabe 3.6

Nach der Definition des Kundenauftrags erfolgt parallel die Prüfung der technischen Machbarkeit, der Kundenbonität, der Produktverfügbarkeit, die kaufmännische Prüfung und die Reservierung des Produkts. Dies erfolgt mit Hilfe einer Verzweigung über einen AND-Operator. Fällt eine der Prüfungen negativ aus, so wird der Kundenauftrag abgelehnt, was über einen OR-Operator abgebildet wird. Der Kundenauftrag wird angenommen, wenn alle Prüfungen positiv

ausfallen. Dazu wird ein AND-Operator verwendet. In Abbildung 3.15 ist die ereignisgesteuerte Prozesskette dargestellt.

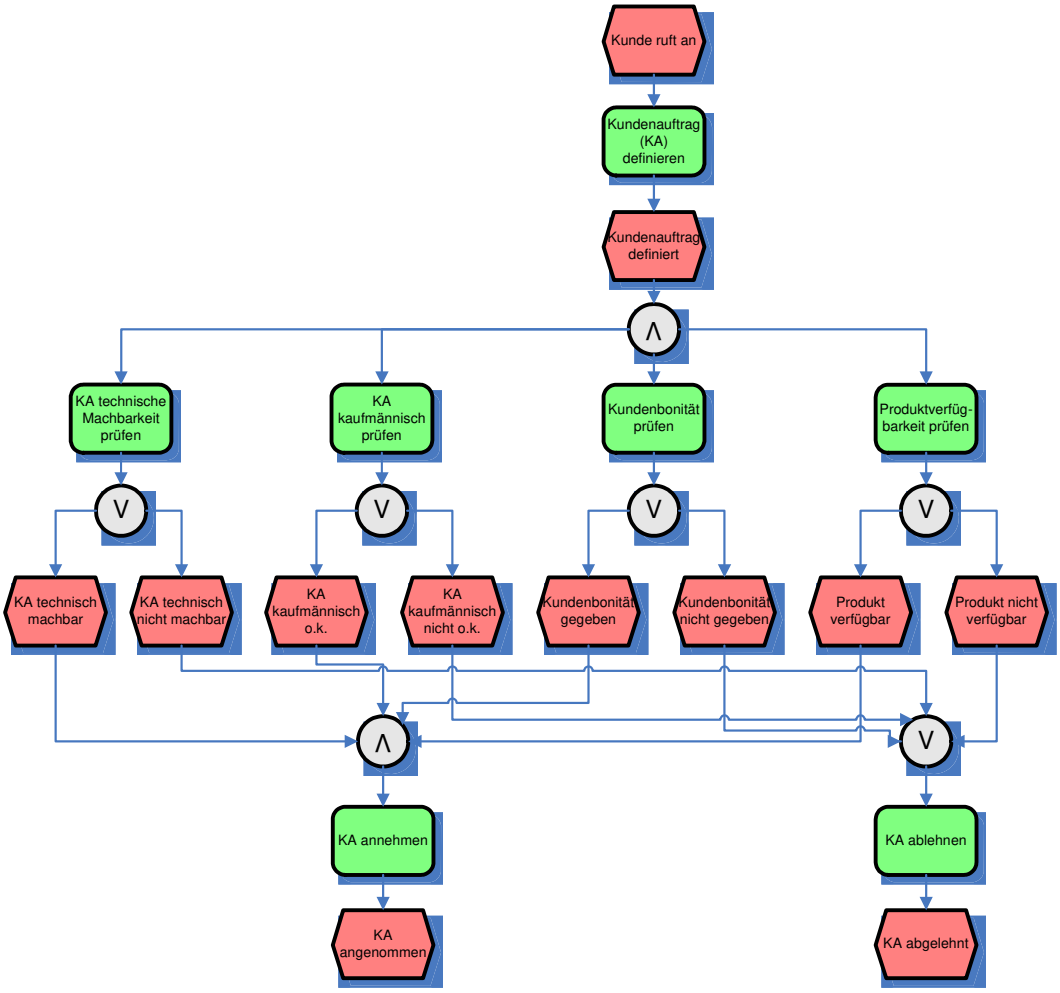

Abbildung 3.15: EPK zur Kundenauftragsannahme

Übungsaufgabe 3.7
Die Abbildung der Art des Kunden erfolgt über die Spezialisierung der Klasse Kunde. Die Klasse Kunde ist mit den Datenklassen Preisfindung, Lieferparameter und Rechnungsparameter über die Komposition verbunden. Zu den Lieferparametern gehört die Klasse Lieferanschrift. Ein Bestellvorgang kann über die Klassen Angebot und Auftrag spezialisiert werden. Zu einem Bestellvorgang gehören die bestellten Produkte. Für jedes bestellte Produkt wird über die Assoziationsklasse Bestellposition beispielsweise die bestellte Menge abgebildet. In Abbildung 3.16 ist das Klassendiagramm dargestellt.

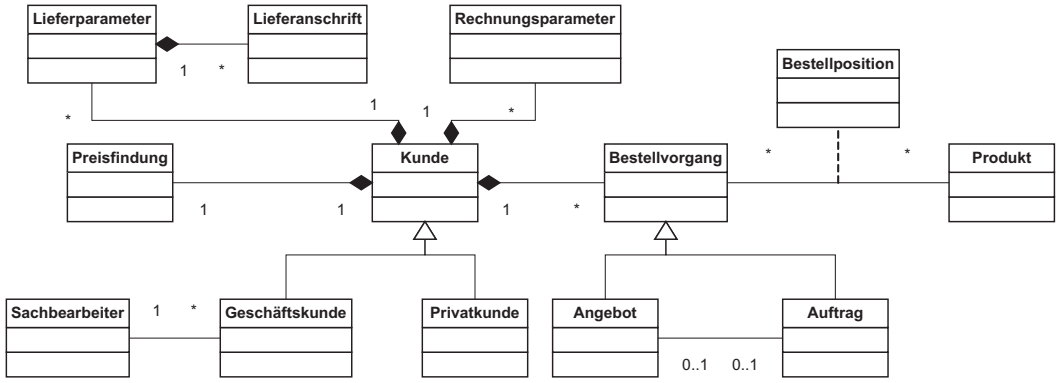

Abbildung 3.16: Klassendiagramm für kundenspezifische Daten

Übungsaufgabe 3.8

Das der Aufgabenstellung entsprechende Sequenzdiagramm ist Abbildung 3.17 zu entnehmen.

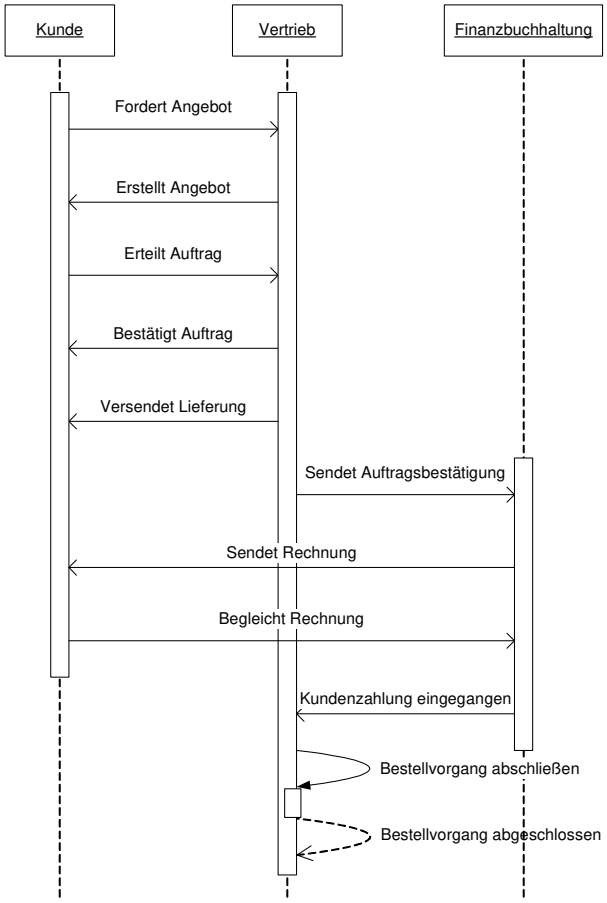

Abbildung 3.17: Sequenzdiagramm für einen Bestellvorgang

Übungsaufgabe 3.9
Der Kunde kann beispielsweise den Zustand verfügbar, wartet auf Angebot, wartet auf Auftragsbestätigung, wartet auf Lieferung, wartet auf Rechnung und nicht solvent annehmen. Die Zustandsübergänge, unter der Annahme, dass nur ein Angebot bzw. Auftrag parallel für einen Kunden bearbeitet werden kann, sind im Zustandsdiagramm in Abbildung 3.18 dargestellt.

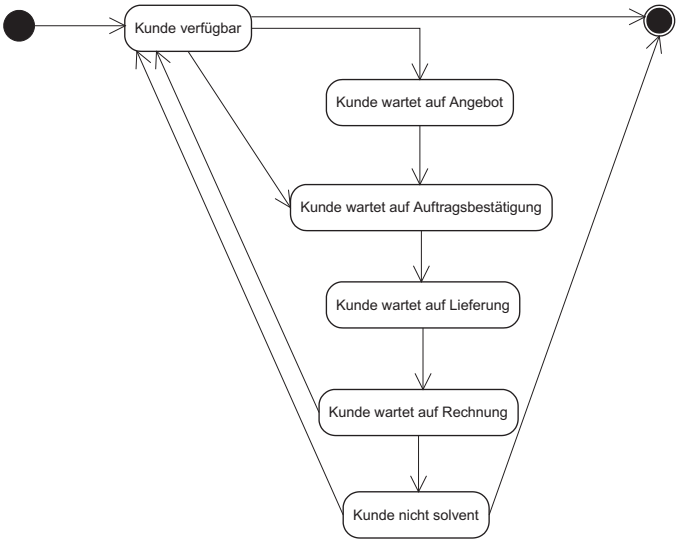

Abbildung 3.18: Zustandsdiagramm für Kunden